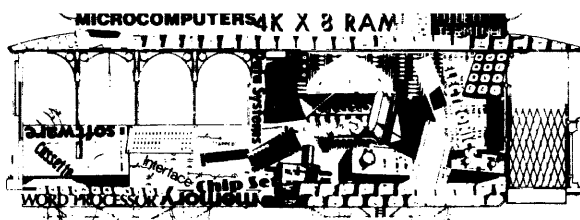# SECOND

WEST COAST

COMPUTER FAIRE

A Conference & Exposition
on
Personal & Home Computers

CONFERENCE PROCEEDINGS

Jim C. Warren, Jr., Editor

March   3 - 4 - 5,   1978   San Jose, California

MICROCOMPUTERS4K X 8 RAM

THE FIRST WEST COAST COMPUTER FAIRE

A Conference & Exposition on
Personal & Home Computers

Available for immediate delivery

CONFERENCE PROCEEDINGS

of the largest convention ever held

Exclusively Devoted to Home & Hobby Computing

over 300 pages of conference papers, including:

(Topic headings with approximate count of 7"x10" pages)

Friday & Saturday Banquet Speeches (16)
Tutorials for the Computer Novice (16)
People & Computers (13)
Human Aspects of System Design (9)
Computers for Physically Disabled (7)
Legal Aspects of Personal Computing (6)
Heretical Proposals (11)
Computer Art Systems (2)
Music & Computers (43)
Electronic Mail (8)
Computer Networking for Everyone (14)
Personal Computers for Education (38)
Residential Energy & Computers (2)
Systems for Very Small Businesses (5)

Entrepreneurs (6)
Speech Recognition &
    Speech Synthesis by Computer (14)
Tutorials on Software Systems Design (11)
Implementation of
    Software Systems and Modules (10)
High-Level Languages for Home Computers (15)
Multi-Tasking on Home Computers (10)
Homebrew Hardware (8)
Bus & Interface Standards (17)
Microprogrammable Microprocessors
    for Hobbyists (18)
Amateur Radio & Computers (11)
Commercial Hardware (8)

---- *plus* ----

Names & addresses of the 170+ exhibitors at the Computer Faire

---

Order *now* from:			
Computer Faire	Proceedings:	$12.00	($11.95, plus a nickel, if you prefer)
Box 1579	Shipping & Handling:	.68	(Write for shipping charges ouside U.S.A.)
Palo Alto CA 94302	Outside California:	$12.68	Payment must accompany the order.
(415) 851-7664	Californians Add:	.72	6% Sales Tax
	Inside California:	$13.40	Payment must accompany the order.

An 8½" x 11" Softbound Book

CONFERENCE PROCEEDINGS

Jim C. Warren, Jr., Editor

THE SECOND WEST COAST COMPUTER FAIRE

held in

The San Jose Convention Center

in

San Jose, California

March 3-5, 1978

PREFACE

As a widespread movement, "personal computing" began around January of 1975. It began as a hobby activity, involving only the dedicated computer hacker and elektroniker who had the time, talent, and patience to deal with the relatively sophisticated electronics that was available only in kit form, with -- at most -- minimal documentation, and virtually no software.

Within less than three years, we saw the entry into the marketplace of several fully assembled, ready-to-use microcomputers, priced as consumer products for the interested technocrat. In noticeably less time than that, we saw the availability of a variety of usable -- though certainly limited-capability -- systems software.

That is, by 1977, personal computing had moved beyond the dedicated computer hobbyist and was beginning to be accessible to the intelligent, logically-oriented novice.

Now -- March, 1978 -- we are seeing the first signs of true "computer power for the people", as I believe these *Conference Proceedings of the Second West Coast Computer Faire* illustrate.

In the First West Coast Computer Faire, that took place in April, 1977, we had slightly over a day of Conference activities regarding very-low-cost computers in education. This Second Faire has over two days of Conference sessions devoted to the topic.

Last year, we had two talks concerning the topic that is perhaps the ultimately "personal" application of computers -- computers for the physically disabled. This year, we have a full day of sessions addressing this topic, including demonstrations of several operational devices. Additionally, the commercial exhibits include several such demonstrations of prototype aids for the physically handicapped.

In the 1977 Faire, a Conference section addressed the potential of networking personal computers. This 1978 Faire -- less than a year later -- includes details of the protocols, and demonstrations of a functioning personal computing network facility.

Last year, there were few talks concerning the entrepreneur wishing to explore this new marketplace, and only one talk addressing microcomputing applications in business. This year, half-day sessions address each of these topics, presenting both ideas and the results of experience in these areas.

Though there have been something in the order of 30 other conventions addressing the topic of home and hobby computing, to date, the Computer Faire remains unique in the fact that it publishes the abstracts and full-text papers of most of the Faire speakers. We set this as a major commitment when we created the first Faire; we are continuing that commitment for the second Faire. These *Proceedings* are the result.

The papers herein were -- at most -- minimally refereed. As was true of the papers in the first *Proceedings*, they exhibit a wide range in quality. However, they also exhibit a timeliness that we feel is essential in a technical area moving as rapidly as personal computing is -- a timeliness that looks askance at the year-and-more turn-around time for obtaining publication in the many heavily-refereed, academically acceptable publications. Additionally, these *Proceedings* illustrate the viewpoint that one need not be "academically acceptable" to do interesting and challenging experimentation. They also illustrate the view that "novice" is a relative term, and that "state of the art" has many dimensions.

<div align="right">

Jim C. Warren, Jr.
Woodside, California
78 February 18

</div>

3

JIM WARREN, Faire Chairperson
    345 Swett Road
    Woodside, California 94062
&
Editor, *Dr. Dobb's Journal of Computer*
    Calisthenics & Orthodontia
    People's Computer Company
    Box E
    Menlo Park, California 94025

ROBERT REILING, Faire Operations Coordinator
&
Editor, *Homebrew Computer Club Newsletter*
    Homebrew Computer Club
    Box 626
    Mountain View, California 94042

RICK BAKALINSKY, designer
    Box 933
    Menlo Park CA 94025
&
willing co-pilot for flights of fancy
    1055 Pine 3, Sweet 1
    Menlo Park CA 94025

TABLE OF CONTENTS

DON'T SETTLE FOR ANYTHING LESS

Alan Kay
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto CA 94304

Biography

As a child, Alan Kay found himself equally attracted to the arts and sciences. In fact, he has never been able to discover any important distinction between the two. A short stint as an illustrator and professional musician was followed by the pursuit of mathematics and biology, occasionally interrupted by involvement in theatrical productions.

Eventually he discovered that the world of computers provided a satisfying environment for his blend of interests. A PhD (*with distinction*) from the University of Utah led to a research position at Stanford University and then to the Xerox Palo Alto Research Center where he is a Principal Scientist and Head of the *Learning Research Group*.

In 1967-69, while at the University of Utah with Ed Cheadle of Memcor Inc., he designed the *FLEX* Machine, the first higher-level personal computer. At Xerox he started the *Learning Research Group*, a ten-year project to produce *Dynabook*, the personal computer of the 1980's. He is the initial designer of *Smalltalk*, the programming system of the *Dynabook*.

Whenever he can he designs musical instruments, cooks, and plays tennis.

Selected Writings

FLEX Machine

FLEX, A FLexible EXtensible Language, Tech. Rep. 4-7, C.S. Dept. U. Utah, 1968
The Reactive Engine, PhD Thesis, C.S. Dept U. Utah, 1969

Early *Dynabook* and *Smalltalk*

A Personal Computer for Children of All Ages, ACM Nat'l Con., Boston, Aug 1972
A Dynamic Medium for Creative Thought, NCTE Nat'l Con., Minneapolis, Nov 1972

Vintage *Dynabook* and *Smalltalk*

Personal Computing, Con. 20 yrs of Com. Sci., U. Pisa, Italy, June 1975
Personal Dynamic Media, w/ A. Goldberg, Xerox PARC (1975)
    "   "   " , w/ A. Goldberg, exerpts: IEEE Computer, Mar 1977
Teaching Smalltalk, w/ A. Goldberg, Xerox PARC, June 1977
Microelectronics and Personal Computers, Scientific American, Sept. 1977

Adam Osborne, President
OSBORNE & ASSOCIATES, INC., 630 Bancroft Way, Berkeley, CA   94710

Summary

This paper examines the principal microprocessor achievements of 1977, and forecasts significant events for 1978. The emphasis is on semiconductor parts that have been developed rather than on home computing system hardware or software. The three most significant parts to be developed and shipped in 1978 are identified.

The recipient of the White Elephant Award for achievement and personal computing will be announced at the dinner. This award is described in the paper. In order to be consistent with the strange logic of the semiconductor industry, the White Elephant Award is an award for outstanding achievement rather than an award for lack of achievement, as the name might suggest.

Significant Developments from 1977

I would like to summarize what I believe to be the most significant microcomputer industry achievements of 1977, while looking at implications for 1978.

At the level of semiconductor components, 1977 was a remarkable year in terms of product announcements and a pretty good year in terms of products actually being shipped. Let us look at the significant semiconductor developments of 1977.

In 1977 the one-chip, 8-bit microcomputer became a reality. Mostek started to ship the 3870 - a one-chip F8 - in volume. Intel followed closely behind with the 8048 family of one-chip microcomputers. The 8048 family is remarkable for the presence of the 8748 series, which provides erasable programmable read-only memory on the microcomputer chip. This is a very significant industry first.

The 8041 and 8741 are variations of the 8048 that need to be specially identified. A casual reading of data sheets might lead one to believe that the 8041 and 8741 are simply variations of the 8048, aimed at some obscure corner of the market. Nothing could be further from the truth. The 8041 and 8741 are significant devices because they have clearly filled a need. Let us explore this need. The concept of the one-chip microcomputer was easy enough to grasp. Based on the high sales of the two-chip F8 configurations, the economics of having a very low-cost, high-volume, low part-count microcomputer were self-evident. But this one-chip microcomputer provides a small, isolated logic system that may well exist on its own. A more subtle and troublesome problem is the sub-logic function, characterized by the device controller. It is easy enough to identify device controllers such as floppy disk controllers, etc. Any microcomputer system will contain one or more of these peripheral devices, each of which needs its own interface logic. Unfortunately, this interface logic must usually be custom designed, resulting in support functions costing far more than the Central Processing Unit. This is a problem which is more significant than might at first appear, since microprocessors are being used in such a wide and varied set of circumstances. Thus, we are not simply talking about peripheral devices such as floppy disk printers and video displays - we are talking about an endless and probably unknown set of interfaces. The 8041 and 8741 address themselves to this sub-logic, interface market. Irrespective of what the CPU and the peripheral may be, an 8041 will generate the necessary interface intelligence, providing this interface intelligence can work within the speed, memory and I/O constraints of the 8041. To complete the effectiveness of the 8041, the 8741 allows you to generate interfaces (initially in low volume) by using an erasable programmable read-only memory to hold programs as they are developed.

We select the 8741 as the most significant part to be introduced and shipped in 1977.

The next area of significant development has been the 16-bit microprocessor. Fairchild introduced the 9440 and started to ship this microprocessor, while Data General introduced and started to ship the MicroNova. Both the MicroNova and the 9440 are one-chip implementations of Data General Nova Central Processing Units. The MicroNova is an implementation of the Nova 3, while the 9440 is an implementation of the Nova 1200.

Specialized processors have also begun to appear. Advanced Micro Devices has introduced the Am9511, which is an arithmetic processor. This very significant device finally makes it practical to use microprocessors in intensive computation applications. The Am9511 brings trigonometric functions, logorithms, exponentials and multiprecision arithmetic to microcomputer systems. We select the Am9511 as the second most significant part to be introduced during 1977.

The next area where we have seen very significant developments is in support circuits for microprocessors. A wealth of parallel I/O devices, serial I/O devices, DMA controllers, priority interrupt controllers and peripheral interface circuits were introduced. We believe the most significant interface circuit to be introduced and shipped is the Z80 SIO device. The Z80 DMA device should also be mentioned, but Zilog is not yet shipping it.

In 1977, Mostek became the first company to start shipping 16K-bit dynamic RAMs in volume. Here again is a development whose significance can easily be overlooked. Why get excited about just another memory device? Very large, low-cost memory devices, as they appear in the future, are likely to revolutionize more industries than any other single development. I single out the music industry - the recording and reproduction of sound - as the one likely to experience devastating changes in the future.

Although 1977 was a year for announcements and product releases, 1978 is likely to see even more dramatic new microprocessor-related products. Specifically, 1978 will be the year of the 16-bit microprocessor - with the announcement and delivery of Intel 8086's, Zilog Z8000's, and sub-

stantial deliveries of TMS9900's and Fairchild 9440's. Given these developments, what impact, if any, can we expect on personal computing?

The answer, surprisingly, is very little. Even now, three years after the first home computers appeared, there is a crippling shortage of software, even to support 8080-based microcomputers. If a manufacturer were to switch in 1978 to a new 16-bit microprocessor, it is likely to be three or four years before this new microcomputer system has any reasonable amount of software support. Thus, the "software prop" is likely to keep existing microcomputers in commercial production for many, many years to come. This "software prop" will be reenforced by the fact that, for many applications, the existing 8080-based microcomputer systems are more than adequate in terms of computing power; any switch to more powerful microcomputers would have little tangible economic advantage. Even for those applications where more computing power is needed, there is always the alternative of moving to new 8080A Central Processing Units that are faster - and therefore more powerful - rather than moving to entirely new microprocessors and instruction sets.

It is easy to fall into the trap of looking upon new microprocessor products as "new waves" which replace everything that came before them. I believe this is a very inaccurate visualization of reality. It is more accurate to think of new microprocessor products opening up new markets - for which older microprocessor products were inadequate. Once some particular level of microprocessor product has been adopted, it will be used for a long time to come because the cost of re-engineering to take advantage of new, more recent developments is simply not realistic. That is to say, new personal computers were manufactured when 8080A Central Processing Units and support circuits made them economical in the first place. Since 8080A Central Processing Units and support circuits were adopted in personal computers, they will be the mainstay of personal computing for many years to come. The fact that an 8086 will be available in 1978 does not mean that three years from now all 8080A-based systems will be obsolete. Far from it. The 8086 is going to have to make its own new markets, and will have little impact on established markets for past microprocessors. Therefore, if you are looking at the personal computing industry and deciding when to jump in,

your answer is: as soon as you find products you can use. Do not wait until next year for better products which may appear, because next year you will be waiting for the following year, and you may finish up waiting forever.

The fact that new developments will not cause old developments to become obsolete is made more certain by the huge customer base for personal computing products which already exist. The personal computing market buoyancy is attested to by the present show, and by the success of so many other shows around the country. This success has resulted from a combination of eager customers and willing visionaries who had the foresight to see what was coming and the vigor to help it on its way. My principal purpose tonight is to recognize the individual who I believe has done more in the past year to further personal computing than anyone else. The name of this individual will be announced at the dinner and not in this paper. To this individual, I plan to present a singularly apt award. In order to be apt, this award must recognize the perversities of the semiconductor industry. Instead of rampant inflation, this is an industry of rampant deflation. Instead of protecting every new product from competition, this industry runs out to find a second source, who is given all necessary secrets to compete effectively. Since everything is back-to-front in this industry, it is only appropriate that an award for achievement be given a name more aptly associated with lack of achievement. Therefore, the annual award which I plan to present will be known as the White Elephant Award. But, instead of representing the biggest waste of effort, my White Elephant award will recognize the best-spent effort. The award consists of an 8741 chip, which is my choice for Chip-of-the-Year, mounted on a suitable plaque with a microscopic White Elephant cemented onto the surface of the chip. I plan to award this trophy annually, using the Chip-of-the-Year for each year's trophy. I furthermore plan to choose the chip and the recipient of the award entirely on my own, without letting my judgment be clouded by committees

or input from the personal computing community.

In recognition of the individuals who made the selected chip possible, the award will list these individuals in addition to the person receiving the award.

DINKY[1] COMPUTERS ARE CHANGING OUR LIVES

Portia Isaacson, The Micro Store
634 S. Central Expressway, Richardson, TX 75080
(214) 231-1096

Computers can now (or will soon be) found in cars, sewing machines, tombstones, typewriters, and pinball machines. The age of the abundant computer is here. As it completely unfolds we will think we have entered a land of science fiction. Dinky computers will permeate virtually all aspects of our lives. Computers will be used in old ways by people and businesses who couldn't afford them before and in many exciting new innovative ways that we couldn't even have thought of before.

Computers have been around for some time. Why all the fuss now about change? The answer is simple. We now realize that computers can be useful to individual people. A few years ago the price of a computer dropped past a threshold that caused a lot of people to understand that the computer was a personally useful tool. A few people understood before, but now that idea is so popular that it has some of the aspects of a religion. The idea of the personal computer certainly has a large and active following.

The changes brought about by dinky computers will be many and not all will be good. Change will be rampant in the computer industry. But few institutions or individuals will escape without change. Businesses both large and small, the U.S. economy, labor, women, the handicapped, the data processing professional, government, the U.S. Postal Service, and our educational system are among those that will be changed by dinky computers.

Business, Labor, and the Economy

Small businesses can make use of dinky computers in a variety of ways--most of them scaled down versions of the same applications in big businesses. Applications common to most small business include: general ledger, accounts payable, accounts receivable, payroll, and inventory control. Some businesses will find a use for word-processing in the generation of letters and reports. Mailing list maintenance and label generation are popular computer uses. A small business

might find a computer useful in scheduling people or equipment. Some businesses will have applications specialized to their own business such as a personnel agency's maintenance and search of an applicant data base or a savings and loan company's calculation of amortization schedules. Innovative applications might include sales forecasting, electronic mail for ordering, building security, energy conservation, games as sales techniques, and graphics in advertising displays.

A typical configuration for a small business computer system including 32K bytes of memory, dual floppy disks and a continuous forms printer costs less than $5 per day when amortized over three years. Small businesses commonly find that a computer costing less than $5 per day can replace one or more employees and can give the management more timely and accurate information than they were getting before. In general, the effect of the computer on the small business is to improve productivity while reducing costs primarily by reducing the number of employees in relatively unskilled positions. By reducing overhead an increasing number of small businesses will find themselves viable. This experience is not unique to small businesses but is the same as that of large corporations which preceded them in the use of business computers. Future applications could include conferencing and working at home.

The effects of the managers' use of the dinky computer will be many. The productivity of clerical employees will be increased. The effect of an easily accessible private computer will be to improve budgeting and project control techniques. Electronic mail will decrease the need for unskilled labor and decrease the use of the post office.

The same $5 per day business computer system found so helpful in small businesses will also be useful to the manager in the large corporation. Now a manager at nearly any level can afford his or her own private computing resource. One of the first applications will be word processing for the preparation of letters, memos, and reports. Other immediate applications include: budgeting, project control, maintenance of specialized data bases, sales forecasting,

---

[1] Of the many words Ted Nelson has given us, this is one of the best.

scheduling, reminders, mailing or routing list maintenance, and electronic mail.

The overall effect of the use of low-cost computing in business will be an increase in national productivity and an improved economic position for the U.S. in the world marketplace. The U.S., as the undisputed leader in low-cost computing technology, will be able to use this technology as a principal weapon in any future economic war.

The labor force will experience both positive and negative effects of low-cost computing. On the positive side, there will be reduced need for people to do boring work. However, there will be a reduction in the demand for relatively unskilled labor such as clerical, mail service, and bookkeeping. Since most of these jobs are now filled by women, women will be hardest hit by the reduced demand for unskilled labor. Countering the increasing demand for programmers will be the fact that entry-level programmers will be in plentiful supply since low-cost computing will make computer education, even self-education, widely available.

Now computer-inventiveness is in the public domain. Before only large corporations and well-endowed universities could invent products containing computers. Now the man or woman on the street has economic access to computers and can use them in inventions. I'm sure they will. The same inventive talent that brought us the automobile and the electric light will bring us "intelligent" computer-based products that are now beyond our imagination. The businesses springing up around these inventions will employ people and further improve the U.S. economic position.

The Computer Industry

As the demand for dinky computers goes up, the demand for gargantuan computers will come down. It will often be found that new applications, or portions of new applications, are more economical on small computers. The traditional corporate demand for bigger and bigger computers will slacken as fewer new applications are developed for it. Additionally time-sharing use of the big corporate computer will be replaced by small computers in instances that are not locked in by data bases or applications software.

The corporate data processing center will lose control of the data processing function as more and more departments own their own computers. The DP center will do less new development since new projects will be done at the department level if possible. The DP center may find a new role when departments realize that they want to access the central data base and communicate with the computers of other departments. DP's new role will be in planning the distributed

data base and communications networks. This role will not be easy since departments will realize that information is power. The struggle over how to distribute the data base will be a power struggle between departments with DP caught in the middle.

Now that computers can be owned by individuals or dedicated to the use of an individual in a corporation, there is little need for time-sharing. In fact, time-sharing was invented as an attempt to give the illusion that each user had his or her own computer. Now that each user can have his or her own computer, time-sharing is no longer needed and the overhead required by sharing makes it uncompetitive. Present time-sharing customers will, of course, stay with time-sharing if they are locked in by software or data bases. Additionally, there are a few applications that may need resources too great for today's dinky computer.

The big computer will not go down without a fight. We can expect to see significant price cuts in order to keep the gargantuan machine alive. But ultimately the giants will be kept only to run programs too hard to change. Most new architectures will be based on unshared computers, shared large disks, and shared fast peripherals connected into networks. The heyday of distributed computing will have arrived.

The new computer industry will see many opportunities. Computer manufacturing and distribution will be feasible small businesses. The new small companies with low overhead will keep the price of computing low; and, in fact, may provide the solution to the problem of the present near-monopoly in the industry. There will be a new economics associated with mass produced software. A complex software package may sell for just a few dollars because it will be sold thousands of times. Individuals may be able to capitalize on their efforts in software creation through royalty payments in much the same way as authors of books do now.

The data processing professional will be faced with many changes. The data processing department will need maintenance programmers, communications and network experts, and data base designers. Programming will be done in user departments where application knowledge will be at a premium. So programmers who don't fit into the new DP department will find themselves in user departments specializing in a particular application area. This specialization will certainly limit their mobility.

Although lower-cost computers will mean more computers and a great demand for programmers, the greater demand will be offset by a greatly increased supply of entry-level programmers and the fact that programming will be easier. Schools at all levels will be able to offer computer training since the

hardware is now affordable. Many people will even teach themselves how to program. The new dinky computers are interactive and much easier to program than big batch computers. All this could lead to a decrease in the salary-level of entry-level programmers. Ultimately this must affect other levels.

As the public becomes more and more knowledgeable about computers, the job of the data processing professional will seem much less glamorous and mysterious and much more just an ordinary job. This will have more than just an ego deflating effect on the profession. A computer-literate public will demand that the programming job be done properly with the good of the public an objective. We can expect to see a public demand for legislation to control computer usage and programmer qualifications. As the public becomes more aware that they are becoming increasingly dependent on unproven computer technology, our profession may find itself in the fish bowl of public controversy.

Government

Government at all levels will experience most of the problems and opportunities of businesses. In addition, government will face some unique changes. The increasing use of electronic mail will bring about further declines in the use and efficiency of the U.S. Postal Service. Government may be able to reduce the demand for energy by encouraging the use of computers to control and conserve energy usage in homes and industry. Crime can be decreased through the use of computerized security systems. The cost of political campaigns may be decreased by applying low-cost computing to the data processing tasks involved in a campaign. Government must help solve the problem of protection for the author's rights in mass-produced software. Increasing displacement of unskilled labor by computers will be a difficult governmental problem. New legislation may be required to control computer technology. Finally, our government will be faced with the new ghetto of the computer "have-nots."

The Individual

All the changes previously mentioned affect us to some extent individually. There are other effects, however, that deserve mention.

The computer brings us a new form of entertainment. It is entertainment through the simulated experience. Often called computer games, this form of entertainment can offer very challenging and highly involving activities. The most popular game of this class is Star Trek. It lets one pretend to be captain of a star ship charged with defending the universe against klingons. The strategies and events are intricate and demanding requiring quick and correct decisions. Computer games are often intellectually stimulating as well as just plain fun. Although the computer games encourage socialization to an even less extent than television (there are no commercials), at least they involve the player in the activity unlike passive television-watching.

Besides games, the computer offers other opportunities for entertainment and creativity via computer-generated art and music. For several years a few artists and musicians have experimented with the computer as a tool for creativity and expression. Now the computer as an artist's tool is available to many.

The low-cost computer coupled with video disk technology could do much to increase the availability and flexibility of personalized education. These new technologies make high-quality computer-assisted instruction techniques affordable by educational institutions, libraries, corporations, and individuals. The place of education may become much more flexible. The role of the educational institution may change to primarily that of preparing courseware and certification of knowledge or skill levels.

Computers can be used in many ways to improve the lives of the handicapped. A person without arms or legs could control a wheelchair by voice commands. A blind person might use a typewriter, computer terminal, or calculator that speaks each letter or number. A deaf person might use a telephone that visually displayes messages. A speech-impaired person might use a speech synthesis device that spoke what was entered at a keyboard. The possibilities are exciting and many.

In the gizmo age we will be surrounded by "intelligent" devices ranging from the self-dialing phone to the self-flushing toilet. Most of these devices will be helpful and friendly, but not all. The computer-generated junk phone call is with us. A computer-based device can place calls, play a recorded message, record a response, and even accept touch-tone input of a credit card number for a purchase. The unlisted number doesn't help since the device could place calls to all the numbers having a certain prefix--a very inexpensive way of placing calls to a part of town corresponding to a certain economic level. The devilish device could remember that you didn't answer and call until you do. It could even remember that you hung up and pester you until you listen. Unfortunately, junk telephone calls are a fraction of the cost of junk mail. A bill has already been introduced in the Congress to control this nuisance made possible by dinky computers.

What will be next?

Low-cost computing will add fuel to the already threatening invasion of individual privacy. Abundant dinky computers mean data bases too numerous to control. An individual won't have a chance of knowing whose keeping what records about him or her. Cheap computers will mean increased feasibility of surveillance of individuals by government or business. The IRS might be able to check, in detail, every tax return. Isn't that exciting!

Conclusion

We've surely only glimpsed the brave, new world being created by dinky computers. The next few years will be more exciting and probably less believable than most science fiction. I want to be there as it happens. Perhaps I can help . . . . .

BEGINNER'S GUIDE TO COMPUTER JARGON
John T. Shen
Computer Scientist & Consultant
Naval Ocean Systems Center
271 Catalina Blvd
San Diego, CA 92152

Human nature being what it is, we're always trying to develop tools to make problem-solving easier. We also try to develop tools to do monotonous and mechanical jobs so we'll have more free time to do the things we enjoy. So, unlike humans, the tools we develop make fewer mistakes, work without getting tired and don't go on strike.

One of the best tools we've created is that creature called "computer." But what is a computer? What's the difference between a computer and a microcomputer? What do we mean by multiprocessing? And what do we mean by large scale integration (LSI)?

A computer is an electronic tool that can accept information supplied by a human or another machine. A computer also accepts instructions regarding what to do with the information supplied. The computer then performs the operations on the given information. After the instructions are performed, the computer supplies the results to the person who requested them, or to another machine which may need the results to carry out other operations.

A basic computer is usually composed of an input and output (I/O) unit, memory (or "storage") unit.

The input unit accepts the information to be operated on from people or other machines, and the output unit makes the results available in terms a human can understand.

The memory unit stores information until needed by one of the other units, such as the arithmetic and logic unit, the control unit or the I/O unit.

The arithmetic and logic unit (ALU) does the arithmetic and logic operations necessary to sort or search for particular items or perform mathematical procedures.

The control unit manages all the other units. For example, the control unit decides when the I/O unit will accept information and when the information should be sent from the I/O unit or the memory unit to the ALU for processing. The control unit also decides what operations to carry and in what sequence. When an operation is completed, the control unit decides whether the results should be sent to the I/O unit or the memory unit.

The technology for building today's computers is very different from the technology that built computers 10 to 20 years ago. Until the late 1950's, computers were built from electronic tubes, mechanical relays, resistors and capacitors. We call these computers the "first generation".

From the late 1950's to the early 1960's, computers were built from discrete transistors, resistors and capacitors. We call these computers the "second generation".

In the early 1960's, a new technology arose, called integrated circuits (IC), where many components are fabricated on a chemical substrate called a "chip", which is about 1 centimeter square. In the early 1960's only 100 transistors could be packed on a chip. Computers implemented with 100-transistor chips are called the "third-generation".

Later, new fabrication techniques were developed, so that today we can pack 1000 or more transistors on a single chip. We call these computers "fourth generation".

From the first to the fourth generation, the physical size of a computer with the same computing power shrank drastically. The cost also decreased impressively, and the computing speed increased several magnitudes.

A computer designed for use in many fields of business and science is called a "general-purpose computer". A computer designed for a specific purpose, such as monitoring a patient's heart condition, is called a "special-purpose computer".

A small computer is called a "minicomputer". A very small computer is

a "microcomputer". The central processing unit
(CPU) of a microcomputer is called a "micro-
processor".

If a computer has more than one CPU, and if
the CPU's are operating in parallel, the computer
is called a "multiprocessing computer" or a "multi-
processor".

But just having a computer will not solve
your problems. You need a way to instruct the
computer to solve a problem. One way is to write
a "program" (a set of instructions or steps that
tell the computer exactly how to solve a problem.

Since English is our native language the
languages used for writing programs are usually
English-based, examples of English-based languages
are FORTRAN, COBOL and ALGOL. Some of the pro-
gramming languages are mathematically-oriented,
such as APL. Both types are "human under-
standable". They are called "high-order languages".

But all computers are built on the simple
language of "yes" and "no" or (1's and o's), which
is the "machine language". To use high-order
languages, we must build translators to act as
interpreters between man and machine.

The programs programmers write to solve
their particular problems are called "application
programs". The large program developed by the
computer manufacturer for managing the computer
resources such as I/O devices, memory spaces and
CPU time is called the "operating system".
Programs that facilitate the easy use of I/O
devices and peripheral memory are called
"utility programs".

The application programs, language trans-
lators, operating system and the utility pro-
grams are called "software".

When a language translator completely trans-
lates a program before the execution of that
program, the translator is called a "compiler".
If a translator translates one statement of a
program at a time and executes that statement
immediately, the translator is called an
"interpreter".

When a portion of a control unit is
electrically programmed into a device call
"read-only memory' (ROM), or when some of the
software is electrically programmed into one or
several ROMs, the programming is called "micro-
programming".

EVERYTHING YOU NEVER WANTED TO ASK
ABOUT COMPUTERS BECAUSE YOU DIDN'T
THINK YOU'D UNDERSTAND IT ANYWAY,
             OR
A TALK FOR PEOPLE WHO GOT TALKED INTO
    COMING HERE BY SOMEONE ELSE

       Copyright Jo Murray, 1978

              By
           Jo Murray
       2325 Leimert Blvd.
       Oakland, Ca. 94602

    This talk will trace the history of
computers, beginning with Charles Bab-
bage, who might have given us computers
more than 100 years ago if he had only
known more about electricity. Bab-
bage was aided by the Countess of Love-
lace, the daughter of that soulful and
anti-machinery poet Lord Byron. Then
there was George Boole, another nine-
teenth century figure who gave us the
symbolic logic that lets computers de-
cide what to do next. The history
continues into the current century with
a few references from such technical
publications as Alice Through The Look-
ing Glass, a brief description of
vacuum tubes, transistors and the sil-
icon chips that run the computers at
the Faire, and no formulas whatsoever.

    If you're one of those people who
have been thinking that all computers
do is print out bills in funny look-
ing numbers and letters and then con-
fuse your orders with your next door
neighbor's, then this is the place for
you.
    You know, you just kept thinking
that if you ignored them long enough,
they'd go away. Then the humans who
could do thinks like talk on the tele-
phone and read names and addresses in
longhand instead of making you put them
in block letters in little squares
would come back.
    But so far they haven't. And if
you're like me, one day you decided
it is convenient to have computers
that know whether there are seats on
planes and telephones you can use to
call across the country and little
calculators you can hold in your hand.
That's not even considering all the
wonderful machinery here today.
    Not all of the electronic marvels
we have today are computers, strictly
speaking. But it is getting very hard

to ignore them.
    Actually, when computers were first
invented, just about everybody did ig-
nore them. Hard as it is to believe,
there was a man back in the early nine-
teenth century who invented all the
principles of computers. His name was
Charles Babbage, and he got a lot of
help in interpreting his ideas from the
Countess of Lovelace. You may know her
better as the only daughter of the
poet Lord Byron. Other people call her
the first computer programmer.
    Her parents separated just after she
was born, and she never saw her father,
but he did write her letters and he
apparently referred to her in some of
his poems, although he didn't seem en-
tirely pleased with her intellectual
talents. A lot of people think he may
have had her in mind in the passage in
Don Juan which reads:

    "'Tis pity learned virgins ever wed
    With persons of no sort of education
    For Gentlemen although wellborn and
        bred
    Grow tired of scientific conversation.
    I don't choose to say much upon this
        head,
    I am a plain man and in a single
        station
    But, Oh ye Lords of ladies intel-
        lectual,
    Inform us truly, have they not hen-
        pecked you all?"

    Other people thought a little more
of her intellectual talents. When she
first met Babbage, she was with a group
that was described as looking at his
machine as if they were a bunch of
savages looking at a gun for the first
time. But Lady Lovelace apparently
grasped the principles of Babbage's
machine the first time she saw it.
She even predicted that some day the
engines would be used to write music.
    Babbage got the idea because he was
fed up with the mathematical mistakes
of the times. In his day, sailors and
astronomers and anyone else interested
in math carried huge books of tables.
But they were calculated by hand and
they were set in type by hand, and the
number of mistakes was incredible.
One book of calculations for sailors
was so bad that the captain of one
ship, who got it as a gift and didn't
realize it was more vaulable for its
beautiful binding than its accuracy,

was never heard from again.

So Babbage sat down and invented his difference engine, as he called it. It worked by calculating tables, using the difference between two numbers. The idea was that if one pound of meat cost five shillings, two pounds would cost ten shillings and three pounds would cost fifteen shillings. So instead of multiplying three times five to find out how much three pounds of meat would cost, you could look at a table that was constructed by adding five each time.

The machine was also capable of making tables involving squares of numbers, using the principle of the second derivative, or "difference."

This principle was not new: this was the way most mathematical tables were constructed at the time. What was new was the idea of having a machine do it, and the idea that a machine could be constructed so it would never make mistakes.

Babbage used punch cards, which had been developed in France to control looms so they would weave patterns in cloth, to feed his machine information. He even devised an ingenious system, based on logarithms, so the machine would stop and ring a bell if the attendant gave it the wrong card. The engine printed out the answers itself to eliminate the possibility of mistakes in typesetting. He then went on to design a more sophisticated machine, which he called the analytical engine, that would do almost everything computers do today.

Babbage's principles were so close to today's that Howard Aiken, who helped build one of the early modern computers, once said, "If Babbage had lived 75 years later, I would have been out of a job."

Babbage, who held himself in very high esteem, apparently agreed. He wrote that if anyone later developed a similar machine, "I have no fear of leaving my reputation in his charge, for he alone will be able to fully appreciate the nature of my efforts and the value of their results." When Aiken came across these lines, he said he felt like it was a voice personally addressing him from the grave.

But Babbage had two things going against him, and his engines were never completed.

For one thing, electricity had just been discovered. It was only in 1831 that Michael Faraday discov-

ered a moving magnet would induce an electrical current in a coil of wire.

For another thing, toolmaking was a relatively new art. Clocks were about the most complicated machines that existed, and they were all individually made by hand. Since hecould not use electricity, his machines needed an enormous number of gears. He had to have a workman make most of the tools he needed to make the precise gears, and then his chief workman got mad and quit and ran off with the tools. But that was another problem.

Babbage did gain a sort of prominence for his time. There were just enough people who appreciated his genius that when he died in 1871, the Royal College of Surgeons of England preserved his brain, which is still there today. But the surgeons who examined it to see if brain looked different from anybody else's couldn't find anything especially remarkable about it.

After Babbage and the examination of his brain, computers faded from the scene for awhile. There were just a number of minor steps that all had to be taken before the first modern computers could be built during World War II.

For one thing, Babbage wanted his engine to be smarter thanmost computers are today. He wanted it to be able to do sophisticated things like add 204 and 311. Both of his engines--which were never completed--were to do this using the decimal system in the same sort of way that the odometer on a car works. When one row of figures reached 10, it was to automatically cause the wheel of figures in the next row to turn.

Today's computers don't even try to count as high as 10. They're like the Red Queen in _Alice_ _Through_ _the_ _Looking_ _Glass_. You remember when she asked Alice, "What's one and one and one and one and one and one and one and one and one and one?"

"I don't know," Alice replied. "I lost count." And then the Red Queen yelled, "She can't do addition."

Well, the difference between us and computers is somewhat like the difference between Alice and the Red Queen. Just about anybody here can add 204 and 311 in their heads if they put their minds to it. But if I stood here and said "one" over and over again, first for 311 times and then for 204 times, I doubt that anybody could tell me exactly how many times I said "one."

Well, this is what computers do.

They say one, one, one, one, one, one, one, etc. and they keep track of it. Or they subtract "one" two or three hundred or thousand times. And if you think about it, multiplication is simply a matter of adding numbers and division is simply a matter of subtracting the divisor over and over.

Actually, it's a little more complicated than this, but this is basically how they work. The don't know any numbers but ones and zeros and they count them over and over again.

This is known as the binary system, and when you take the binary system and electricity, you can do some incredible things with computers.

The reason we use 10 as a base is probably because we have 10 fingers. Babbage used base 10, too. But the modern computers don't have 10 fingers or even 10 rows of digits like the early machines. They just have electrical switches. They're either on or off. They either have current flowing through them or they don't. If they're on, the computer counts them as one. If they're off, the computer counts them as zero. This gives you a numbering system that's very easy for computers, even though it's difficult for humans.

Probably people who work with computers a lot can look at binary figures and read them as easily as we can read the decimal system. But I can't. So I'm going to refer to this little card to explain the binary system.

The digit in the righthand column represents the number of ones. The other columns are not powers of 10, but powers of 2.

Binary		Decimal
1		1
10		2 (2^1+0)
11	is the	3 (2^1+1)
100	same	4 (2^2+0+0)
101	as	5 (2^2+0+1)
110		6 (2^2+2+0)
111		7 (2^2+2^1+0)
1000		8 $(2^3+0+0+0)$

You can add them just like you add in the decimal system, except that as soon as the total is 2, you have to carry a digit to the next column.

Binary	Decimal Equivalent
1	1
$\frac{10}{11}$	$\frac{2}{3}$
1	1
$\frac{11}{100}$	$\frac{3}{4}$

You can also use the on-off switches, or the zeros and ones, to represent letters. You can say A=0, B=1, C=01, and so on. By the time you get up to five digits, you have 32 different combinations and that's enough for the entire alphabet. From there, you can write anything.

So now you've got all this material in the computer represented by ones and zeros. But you still have to do something with it. That's where George Boole comes in.

Boole was an Englishman who lived during Babbage's lifetime. Boole lived from 1815 to 1864, and Babbage did most of his work from 1812 to 1842, but I haven't come across any evidence that the two knew each other.

Boole developed something called Boolean algebra, which is really more symbolic logic than algebra. He also was one of the first people who argued that logic should be a branch of mathematics, not philosophy, and he certainly had some good reasons for it. Today you find a fair number of philosophy majors working with computers, and it's not as odd as it first sounds. The computers work on the same principles of logic that philosophy departments teach.

Boolean algebra is the type of logic where you have those little puzzles that look as if they came out of algebra books, such as "If A is true, B is not."

Nobody found much practical use for this until this century when a man named Claude Shannon was working on his master's thesis, and discovered you can change these logical statements into sets of ones and zeros and let the computer use the rules of Boolean lgebra to make its own decisions about .1at to do next. This is the sort of logic that should tell the computer it doesn't have to send you a bill if you don't owe the store any money. The computers that haven't been programmed very well are the ones that send you a bill, anyway.

They say one way to tell if you'd make a good computer programmer is to take a puzzle like this one from Litton Industries. If you can figure this out and think it's fun, you'd probably make a good programmer. If you're ready to throw up your hands in despair, you'd better stay away from programming.

"If Sara shouldn't, then Wanda would. It is impossible that the statements: 'Sara should' and 'Camille couldn't' can both be true at the same time. If Wanda could, then Sara should and Camille could. Therefore Camille could. Is this conclusion valid?"

Now that you know whether you should be a programmer or not, we'll go on.

Another name you hear a lot is that of John Von Neumann. He's the one who figured out that you could put the entire program into binary form. Just why he decided to do this I'm not sure because if there was anybody who didn't need computers, it was Von Neumann. One story about him is that one of his fellow researchers had stayed up until 4:30 a.m. doing five problems with a desk calculator. Then he decided to play a trick on Von Neumann. Von Neumann came in the next morning, and his friend asked for help in solving the problems. In five minutes, Von Neumann had worked out four of the problems in his head. The other person, who still didn't say he already knew the answers, then announced the fifth answer. Von Neumann apparently was quite perturbed that someone could figure out a better solution to the problem than he could until they told him what was going on.

The early computers worked on vacuum tubes, and that soon got to be a problem. Vacuum tubes get very hot and they burn out. They're like a light bulb. It doesn't matter how good they are; sooner or later they're going to burn out. The ENIAC, which was the first totally electronic computer, had 17,000 vacuum tubes. And it wasn't long before computers were getting so big that if you made them any bigger, it would take 24 hours a day just to replace the vacuum tubes that had burned out.

Fortunately, about this time--in 1947 to be exact--the transistor was invented. Transistors do the same thing as vacuum tubes, but they're much tinier and they never burn out. It is possible to destroy a transistor by dropping it or by running too much current through it, but you really have to work at it.

This solved a lot of problems, but it basically got computers down from the size of a small house to about the size of a smal living room. You could, by the late 1950s, use transistors to make radios small enough to pick them up and carry them around with you, but computers still needed too many transistors to be very portable.

By this time, you had to do your work under a microscope, but scientists kept on working. In the late 1960s, something called an integrated circuit was produced. Dr. Robert N. Noyce, the president of Intel Corp. in Santa Clara, is generally credited with being one of the co-discoverers of it. What the integrated circuit means is that you can put the entire electronic circuit on a single piece of material, usually silicon.

These are so minute that it's hard to believe. This is a silicon chip. What's even more amazing is the fact that it's just the little gray spot in the middle that does all the computations. The rest is here because you can't connect wires to something as small as the chip. But these gold lines eventually connect with little hairlike silver wires that lead into the silicon. I don't know how many transistors are on this particular chip, but some have 100,000. It may soon be possible to put a million transistors on something this size.

To give you an example of the differences in size the integrated circuit has meant, it's possible to put the entire UNIVAC computer, which was the first commercial computer, on one of these.

If your family was one of the first in the neighborhood to have a television, you may remember the UNIVAC which was a guest of sorts of "People Are Funny." It used to spew all its cards out in front of the camera and Art Linkletter would pick them up and read off the names of two people who would get to go on a blind date together.

The UNIVAC is now in the Smithsonian Institution, and it's the little chips like these that are taking over the world. Probably every piece of machinery at the Faire here depends on these silicon chips for its operations.

The silicon chip starts with a very unexotic raw material: sand. A shovelful of sand can supply the basic raw material for an entire computer.

Silicon companies take sand and pure silicon "seeds," which are sold by only three companies in the world. They

use these and "grow" cylinders of material which look like shiny, gray mirrors.

Once you get the silicon, the hard part comes. You have to put the transistors on it. The way you do it is sort of a cross between batik and photography.

In batik printing, you first draw the design on a piece of cloth. Then you decide which parts you want to turn out a particular color and cover everything else with wax. You dip it in a vat of dye, let it dry and scrape the wax off. The next time you cover everything but another color with wax, dye it again, scrape the wax off again, and keep on going until the picture is finished.

To make a silicon chip you do almost the same thing except that layers of silicon oxide take the place of the wax and tiny lines of metal form the picture. The lines are so small and so thin that they are put on the chip through a photographic process in much the same way that shining a light through a negative produces a picture on a sheet of photographic paper. In this case, the negative is called a photomask.

To make a photomask, you need a master diagram of the chip. These drawings start out several feet square and are reduced to the size of a chip, again through a photographic process. The masks, which are made out of glass, are made from these drawings.

Each chip needs eight to ten different photomasks, but there are 90 to 100 steps involved by the time the chip is cleaned and new layers of oxide are formed on it and scraped off between photography sessions. And people are already working on ways to eliminate the photomasks and write the diagrams directly on the chip. So far, though, the machines that do this cost over a million dollars.

All along the way, it's a very delicate process. People who make chips don't even let you take pencils inside the laboratory because they produce dust when they write on paper. The water used to wash the chips between the different processes has to be so pure that companies sometimes have their own water purification plants.

One firm--Monolithic Memories in Sunnyvale--says its water is 100 times purer than distilled water. And when the plant finishes with it, it's still 100 times purer than the regular city water.

When you start talking about chips this size, you find that computers are almost becoming self-perpetuating. It would be impossible to make them this small if you didn't already have computers to help do it. Computers are used to test the models of the circuits, they draw the layouts for the photomasks and they control the manufacturing equipment.

But they can't do it all by themselves yet. When the whole thing is finished, that's when they call in the humans. The humans look through a microscope to check all of the circuitry on the tiny chips. The people at Monolithic Memories tell me that after a few weeks of training, people learn to check one in about a minute. The reason they need people is that there are so many different structures and so many differences in the size and the color of the lines that are still acceptable that there's no way to program a computer to remember them all.

There's still not a computer that can remember as much as even the average human brain.

INTRODUCTION TO PERSONAL COMPUTING
A BEGINNERS APPROACH

Robert Moody
2233 El Camino Real, Palo Alto, Calif. 94306
Phone: Home (408) 225-3341, Work (415) 327-8080

I Introduction

Computers are now within everyone's reach!
Whether you are a housewife, small businessman,
student, professional, musician, or in real es-
tate -- computers are being made and sold at
prices you can afford, and they will set you
free in ways you never dreamed of! A personal
computer will open possibilities; it will al-
most certainly change your lifestyle.

Most people, when they visualize a computer,
they think of monstrous machines that tower over
us, seeing all and doing all. It's really not
that way. Everybody has been communicating with
a computer in one way or another most of their
adult lives and not really known it. For ex-
ample, all your tax returns are processed by a
computer, most amjor department stores handle
their buying, stocking, and billing by computer.
All your credit card buying is handled by com-
puter, every check you write is processed by
computer. Dentist, doctor, hospital, gas, elec-
tric, phone bills are handled by computer. Why!!!

Take a city the size of San Jose for inst-
ance. Just think of the tremendous amount of
manpower it would take to try and post all the
checks that were written in one day, or the
amount of phone calls to be logged in one day.
Handling that amount of information — having
that kind of computing power — is at your finger
tips today.

The most important think that I want to
convey to you is: computers are not just for
geniuses!!! You don't have to be a special
gifted person to own or operate one.

This presentation, I hope, will enlighten
you a little as to what are these things,
personal computers, how you can talk and ask
questions about them, what makes them up, and
what you can do with one.

II Buzz Words

As you know, all types of industries have
their own language that they use. We as a group
and a new up-coming industry, are no exception.
As you scan down the second page of the handout
I have given you, there is a short list of these
buzz words, and a simple explanation of each.

BASIC: Beginners All-purpose Symbol Instruc-
tional Code: a mode of language that you will
use for communication between you and your com-
puter.

Bit: The smallest unit of measure in a computer
word; several bits make up a byte, or computer
word.

Bug: The cause of a malfunction, usually in a
program. They're called "bugs" because they can
be hard to find.

Byte: The space which a letter or digit (one
character) takes up in a computer. Space in a
computer is measured in bytes. A megabyte is a
million bytes.

Computer: A machine for handling repetitive in-
formation. Basically it can calculate, compare,
alter, send and receive information very rapidly.

Core: See memory.

CRT: Your computer's "TV screen", showing you
what's IN there. The CRT is your computer's way
of talking to you. It is also something refer-
red to as a display unit, or terminal.

Data: The information that gets WORKED OVER,
when your program runs. Data is all the in-
formation you have your computer use, everything
that is sent into your computer to store and
retrieve.

Disk: A Mass Storage device, either floppy or
hard disk.

Display: Same as CRT.

Floppy Disk: A mass storage device, which uses
a flexible platter to store a large amount of
information.

Fortran: Another type of computer language.

Hard Copy: Computer output ON PAPER, for per-
manent storage.

Hard Disk: Much like a floppy, a hard disk
stores a tremendous amount of information, but
its platter is much larger and not so portable.

Input: The information that goes IN to your
computer system. The computer's "food".

Interface: A connector that "translates" be-
tween two parts of a system. You generally need
one interface for each peripheral, to hook it to
your computer.

Mass Storage: Any way of keeping a lot of information OUTSIDE your computer, but available to it. This is your computer's "memroy". Most common kinds of mass storage are <u>tape</u> and <u>disk</u>.

Micro, Microcomputer, or Microprocessor: Same as computer. The "micro" came in when we learned to make them physically tiny — they are about the size of pencil erasers.

Output: What your computer system produces.

Peripherals: The devices attached to your computer, such as the <u>display</u>, <u>keyboard</u>, <u>printer</u>, etc.

RAM: <u>R</u>andom <u>A</u>ccess <u>M</u>emory. This storage device is used by your computer to change the data you have put into it, then it is transferred to a mass storage device.

Storage: The part of your system that <u>remembers</u> information, as opposed to the parts that "think".

Software: A list of instructions to the computer, telling it what to do and when to do it.

Terminal: A unit for conversing (input or output) with your computer. It has a keyboard, plus Display or Print-out.

TVT: "Television typewriter". A keyboard and electronics specially designed to turn your TV into a TERMINAL.

III   What Makes Up a Computer?

In your handout you will find a block diagram of a computing system. As you notice, there are not many modules or components that are needed to do the job. The technology today has made this possible with microelectronics. With this tremendous reduction in physical size of transistors and diodes it enables us to compact a large amount in a very small area. Also, the power requirement is small as well.

The first and most important module is the CPU, or <u>C</u>entral <u>P</u>rocessing <u>U</u>nit. This is the brain of the personal computer. The CPU does all the actual work of calculation, comparison, alteration, receiving and sending data. Everything else that we attach is a function from the CPU.

There is a wide variety of CPU's on the marketplace today. Depending on who you talk to, one is better than the other. Don't let this discourage you now; all that you need to know at this point is that they work. You can get into particulars later.

The CPU cannot operate by itself; along with it you need some memory. RAM or Random Access Memory does the job. This portion of the computer allows the CPU to activate a section of the data at one time. For instance, the RAM could be compared to an active filing system. The data is stored in some kind of order and the CPU only pulls the data it needs and keeps the

rest for later.

Now that the CPU has changed, calculated, or compared this data, it needs to store it someplace. This is where mass storage comes in. Here again there is a wide variety and different types to choose from, but basically it is compared to an inactive file. This file holds much more data than the RAM in your computer does. You as a computer operator put in and take out these files at will, making one active and another inactive.

To be able to accomplish the task of moving all this data around, you have to be able to talk to your computer and have it talk back to you. A peripheral device called a Terminal is necessary. This is usually made up of a keyboard, much like a typewriter keyboard, and a display unit. The Terminal is attached to your computer through an interface device. There are two basic types: serial and parallel.

The serial I/O, or input-output, interface takes the data you are typing in and sends it one bit at a time to the RAM. Parallel, on the other hand, sends all the bits that make up the computer word or byte, and puts them all in at one time. This might seem very confusing to you, but all that is necessary to know is what kind of interface is incorporated on the peripheral you are attaching to your computer, and match it up with the proper I/O device.

Now that I have thoroughly confused you, we will push onward.

IV   Programming

Now that all this hardware has been assembled, we have all the physical things that are needed, but there is something that is lacking: software. What this thing does is allow the computer to try and make some sense out of what you are trying to tell it, and vice versa, have you understand what it is trying to tell you. Fundamentally there are two types: systems and applications.

Systems software is what the computer uses as its language. It takes the information in, in this special language, and acts on it. What you do as an operator is use this language to develope an application.

There is a lot more to programming than just a couple of sentences I have used to describe it. I could ramble on about all the different types of systems software, but I have to spend more time on the biggest question that I'm asked, and that is "I would like to have one of these things called personal computers, but what the hell do I do with it?"

V   What Can I Do With It?

The most asked question that I receive is: "what can I do with this thing, now that I have it?" The uses for personal computers are endless; the list below only shows a few uses — let your imagination go and you will see!

Education
Accounting and billing
Invoicing
Sorting
Receipts
Taxes
Addressing
Budgets
Forecasting
Reducing the physical
   SIZE of FILES (custo-
   mer files, corres-
   pondence, product
   files, etc.) - <u>No</u>
   <u>more file cabinets</u>!!!
Playing games
Engineering design-aid
Cashflows
Maintaining "tickle-
   files" (calendar-
   reminder systems)
Filling out checks
Playing the stock market
FILING
Remembering all trans-
   actions
Making and keeping card
   catalogs
"Simulating" results of
   one decision versus
   another, so you can
   see their effects

Recordkeeping
Inventory management
Routine correspondence
   and form letters
Filling out forms
Calculation of all
   kinds
Receiving and placing
   phone calls
MATCHING any informa-
   tion with any other
   information
Polls and surveys
Indexing
Cataloging
Solving problems
Printing out results
Maintaining LISTS,
   especially:
   MAILING LISTS
   SHOPPING LISTS
   ITEMIZATIONS
   STOCKLISTS
   PACKING LISTS
Sales analysis
Travel and route
   planning
Scheduling
Ticketing
Distribution
Editing

<u>Lists</u> — Computers are lovely at working with lists. The nice thing is that you can change anything in the list -- even one letter "8" for example -- without affecting the rest. It's like a stored-away blackboard. As items become obsolete -- the way they're always doing in a shopping list, say -- you just tell your computer to "drop" them from the list. Your computer does the rest: YOU DON'T HAVE TO KEEP WRITING THE LIST OVER AGAIN.

THIS GOES FOR ANY COLLECTION OF INFORMATION IN YOUR COMPUTER — YOU DON'T HAVE TO MAKE THE WHOLE THING OVER JUST TO CHANGE ONE PART.

This makes your computer a powerful tool for such unexpected things as TEXT EDITING. You can work a whole manuscript over without having to erase or "fix" anything, physically. The time savings alone are immense — in writing, ed-iting, and composition! But imagine USING NO PAPER until you've got the <u>final version</u>!

With all this, your Personal computer adds up to two things: IMPROVEMENT AND EXPANSION. Improvement: Your computer will do FOR you what you're already doing, in a fraction of the time. Expansion: You do things with it you never could have before.

<u>Improvement</u> — If you;re in mail-order, small business, or an office profession, a "micro" computer is for YOU! It can keep your <u>mailing</u> <u>lists</u> and print <u>address</u> <u>labels</u> at a rate of four

per second, all accurate and up-to-date. It can store all your records on products, sales, cust-omers and ad results. It will compare or change any part of these you'd like, without touching the rest. In other words, it gives you SALES ANALYSIS and MARKET PROFILES which only "multi-millioners" have had until now!

Your computer <u>keeps</u> <u>inventories</u>.
It does <u>accounting</u> and <u>billing</u> of all types.
It does <u>order</u> <u>entry</u> (live).
It can hold and index your "<u>own</u> <u>private</u> <u>library</u>" of whatever information interests you.

For example, it can help in health care by keeping track of each patient's medical history and alerting you to patterns you might have mis-sed. Or give you a quick way to look up the newest therapies by <u>what</u> <u>they</u> <u>remedy</u> -- you type in the condition and your computer gives you back a list of indicated drugs or treatments.

If you are a pharmacist: you may presently be keeping "card files" of each customer's contra-indications and other medication history. With a computer, you'd no longer have to look each of these up for each prescription! You'd just type in the customer's name and Rx, and UP would come the pertinent information on him! It would be shown on a little TV screen -- just like at the airlines. You could also tell your computer to remember a list of "potentiating" drug combinations and warn you if it detects one in a customer's combination. YOU DON'T HAVE TO THINK IT OUT OR LOOK IT UP EACH TIME, ANY MORE.

Your computer can run FLEXIBLE FORM LETTERS for you! You tell it to change this word or that, add or delete a paragraph, date a letter next Monday, and address it to, say, all the people in your "Best Prospects" file. OUT will come a stack of letters, individualized and ready to mail! Because of the "add-on" equip-ment which adapts computers for particular jobs, we arrive at your computer's greatest promise: expansion

<u>Expansion</u> — You're about to see what <u>new</u> <u>types</u> <u>of</u> <u>business</u> are possible with a computer running in your own home! These are all high-profit, high-service businesses -- yet they take little SPACE or TIME. Best of all, they're <u>VERY</u> <u>low</u> in STRESS! Because, let's face it, one of the chief causes of STRESS seems to be EMPLOYMENT. It's hard on both <u>worker</u> and <u>boss</u>. So why be <u>either</u>? GET A COMPUTER.

Here are some new businesses which one per-son alone can set up with a computer. The first are all based on one thing computers do very well: INFORMATION-MATCHING! Basically this gives you a CLASSIFIED AD SERVICE — either gen-eral or very special (loke clothing only). And it can be "live", available right over the phone. So, LIVE CLASSIFIED ADS are a very LIVE BUSINESS OPPORTUNITY! It works like this: people call to say they want this or have that, and you tell 'em who has or wants it. You do this by signal-ing your computer what's wanted. In a few

seconds it gives you a LIST of people who have or want that thing! You read that to your caller.

Your computer will display any desired details you've given it in the past, as part of the list. The point is, YOU don't have to REMEMBER it all anymore. So your "data bank" can be much larger than you could ever handle mentally, and you're able to give far broader service. At the same time you're freed up for creative work!

Moreover, some phone companies are learning to cooperate by "On-Call Billing" of calls to your Service number. The caller is simply charges a higher rate for calling that number. These charges are then forwarded or credited by the Phone Company directly to you. In other words, as a business you get paid when you're called.

Where On-Call Billing isn't available yet, you can tie-in your fee with transactions, like the New York "BUYLINES" do. This is more work, but that's what your computer does anyhow.

Some new satellites are making phone lines unnecessary, and getting clearer transmission to boot. Your "phone system" can become very powerful, with help like this. By "being in" with a system ahead of time, you can make the most of it.

New Business –

Real Estate – with up to the minute listings over a wide area.

Dating Service – getting people together. With speedy, detailed descriptions that YOU don't have to look up! (Your computer will spot the most "ideal matches".)

Apartment and Building Rental Service – where subscribers can find places for rent, or list places they have. There's a need for this kind of service, since real estate agencies are more interested in SALES. If you have an agency however, a computer lets you expand painlessly into rentals. With a computer you can make your service "long distance" too, using classifieds from other cities.

Swapping Service – You put people in touch with each other, who need and have things they want to trade. Instead of money, they give you "due bills" which you charge 10% each on. These act as "certificates" for exchanging the goods or service. For example, you know a dentist who wants a used car; and a car dealer who wants his house painted. Your COMPUTER completes the circle: it displays the name of a painter who needs dental work! You have due-bills from each already, and just put the people in touch. And you get paid with each due-bill — so it doesn't matter to you whether they go ahead with the swap or not.

Equipment Rental Service – here your computer merely keeps tabs on who has equipment they are willing to rent, and of who wants to rent some. Again, YOU DON'T HAVE TO STOCK ANYTHING. Furniture, housegoods, and even cars can be rented.

Shopper's "Where To Find It" Service – this is like a detailed "Department Store Directory" available by phone. Your computer keeps track of what stores carry which items, and prices too if you wish! You can also notify of specials and sales.

Instant Babysitting – you run a "register" of babysitters who are available on short notice. This can be used for many other types of jobs and services too, of course.

"Phone Rummage" Sales – Like a rural version of the What's For Sale service. The big items can even be cataloged into your system, so callers can ask, say, for a piano and you'll know where one's for sale. This can be applied to flea markets, too.

Clothing Clearinghouse – this answers an acute need. Everybody –– especially females and children –– had a predicament until now: they need different nice things to wear, and they have good clothes they don't want. Yet they're unable to sell or find a used thing! (If you'd seen the pennies a store offers you for a perfectly good $300 gown you've worn once, you'd see why!) This is a natural for your computer. You just keep pumping it full of the clothing descriptions and clients' names and phone numbers, and your computer will find every match there is! – even down to desired color and size! After you've brought two parties together they can work out any terms they wish about the clothing. You get paid by the phone company, or by clients for the numbers you give them. You DON'T HAVE TO KEEP ANY CLOTHING ON HAND. You need NO SPACE and NO MERCHANDISE to conduct this business.

Now for some businesses that are NOT of the "INFORMATION MATCHING" type. Notice that these still take almost NO SPACE to operate.

Service Bureau – doing small business work for other people or firms. Accounting, billing payroll, mailings, correspondence, taxes and inventory are top candidates. You can do for others, anything we've named so far.

Answering Service – fully automated. We think that with a little ingenuity you could run a whole answering service using nothing but a phone and your computer. I mean, go off to the Health Club while your computer routes messages all day!

Calling Service – Your computer, properly rigged, can do much of your routine phone calling for you. You can even tell it to call someone from across the room! (And IT will remember the number – all you have to know is the name). Some people will use a computer to place automatic "buy-sell" orders on the stock market or commodities. You could be earning in two places at once! You'd be at your job or business, while your computer is playing the

stocks for you!

Horoscopes - your computer can cast horo-
scopes, and then print them out for you to sell.
It's able to mix "individual" information such
as birthdate and place, with the "mass" informa-
tion about astrological types. (You choose and
give it the "mass" information ahead of time.
It can then print out horoscopes in as many cop-
ies as you wish, with even the peoples' names
and addresses on them for mailing. You see such
horoscopes advertised nationally. Those ads net
hundreds of thousands of dollars!

Tarot Readings - this is the same idea as
horoscopes, only the "general" information you
give your computer beforehand is from books on
the Tarot instead of astrology. Your computer
would "draw" the cards for you! Tarot readings
are just getting popular, and more novel to the
public than horoscopes. You could see them
well!

Writing Service - your computer is the best
Assistant Editor anyone ever had. How it saves
TIME! And PAPER! More on this in a moment.

Classified Bulletin - publish it monthly or
bi-weekly. It has nothing but classified ads of
all kinds. The ads are run FREE and people pay
you when there's a sale. This sounds risky, but
for many reasons, IT WORKS. The Classified Bul-
letin is a much better deal for people than the
newspaper -- all you need is adequate coverage.
People's ads run longer, run free, and cover
better geographical area. You could be running
a "live" classified service and a printed ver-
sion, at the same time. Each serves as public-
ity for the other. You could also support your
classified bulletin with paid ads from local
businesses.

Other new businesses possible with your
computer:

"Local Events Calendar
Reservations Service
Opinion Polls and
  Surveys
Lottiers and other
  Fortune Games
Betting Systems - win-
  ning at horses, black-
  jack, etc. It can be
  done, in fact it IS.
  And there are some
  books on it.
Gold Handicapping

News Service
Ticketron
Problem Solving for
  others
Special Interest Clubs
Selling Time on your
  computer. You charge
  other people for
  using it.
Your Own Publication -
  magazine, classified,
  or newsletter, edit-
  ing with the computer.

You're getting an idea of your computer's
business uses. We bet you'd also like to know

How a Computer Can Help You in Your Home

Now with all of these uses, you must be
starting to guess what one of these "mighty
midgets" can do in your home! First: you can
have everything controlled automatically, whe-
ther you're there or not.

Your home computer can literally become
your "private secretary". Make you feel like
the President, as it keeps track of notes, dates,

reminders, appointments... It will alert you at
the right time -- give you a message that
"school's letting out early today," or that So-
and-so's coming to visit, or that you have an
appointment downtown at 2:00. Keep lists, bal-
ance your checkbook, be a library, manage your
finances and answer the door. YOUR MIND GETS
UNBURDENED — a computer's really useful for that!
Maybe even prevents wrinkles, if those come from
feeling harassed!!!

Right now, you'll "tell" your computer what
you want by typing messages on its keyboard.
Don't worry how it "understands" them. It does.
You just type your instructions, and it'll do
the rest.

Now you'll even be able to SPEAK your com-
mands to it, and it will carry them out! For
example, there's an alarm clock on the market
right now that stops ringing if you yell at it.
(If you just groan or talk, it rings less.) And
it recogni es YOUR voice — no one else's!

Any electrical device can be set up to do
this. We've always liked the idea of LIGHTS
that keep themselves on only when someone's
around. (You wouldn't have to talk to them,
they'd know you were there by your temperature.)

As each of these technical devices comes
onto the sales market, you'll be ready. You can
just ADD them to your system.

Your computer can also help with home pro-
tection. It's hard to beat a trained attack dog
of course. But he can't call the Fire Depart-
ment if he smells smoke, or signal your "beeper"
when the baby wakes up. And you don't have to
walk your computer. For burglary, as a matter
of fact, a little machine is planned that you
attach to your door. It's activated while you-
re away, to SOUND like a fero ious dog roaring
and hitting the door, if anyone starts tampering
there. Your computer can do even better, espec-
ially if you have a Hi-Fi set.

It could even be wired to water plants or
feed fish.

AT HOME AS IN BUSINESS, your computer's
YOUR PARTNER IN JUDGMENT!! And IT HELPS WITH
ALL KINDS OF PLANNING. You can do budgets,
schedules, and calendars, party management, in-
ventory, and keep your own "mailing lists" and
phone directory. You can do super MENU-PLANNING
with a computer, too. It tells you what's in
the refrigerator, what you're out of, what needs
using up, and what you'll need to buy for a cer-
tain recipe. Or it tells you what recipes you
can do with what's on hand. Frankly, all this
would be a big order for your little computer at
present. But it's coming (Also, the longer
you spend with your system, the more versatile
it gets.)

Remember how easily your computer handles
LISTS. Well, you can add or delete items all
week, and review it anytime you want. Only
those items you changed will be different. And
the rest of the time your lists are tucked away
safe and CLUTTER FREE!

Now more on TEXT EDITING, because it's so well suited to WORK-AT-HOME! (The sources of editing work to do at home are so varied that we won't go into them here. The commonest is TYPing, of course.)

You type your material into the computer, which "stores" it in its memory. Then you call up the material section by section, and work it over. YOU USE NO PAPER. You don't have to erase, or "fix", or buy correction fluid. You don't have to make carbons. Any change you call for is made instantaneously. After the work is EXACTLY THE WAY YOU WANT IT, you press the "PRINT" button. THEN your system spits out finished pages. The newest systems do this a whole page at a time, and silently like a Xerox.

Something else your computer can do for you

Indexing

We don't know if you're into indexing or not. Some are, some aren't. If you have a lot of ideas or keep a diary or want to save clippings and the like, though, INDEXING is for you. It makes a compact filing system, ready to go -- and your computer's the one to do it! It will keep track of all the headings, key words, page or issue numbers, and the alphabetizing of everything. Here's how it works, basically: Say you have a clipping or an idea you want to "index". Say it's about a car that runs on sunpower. You just tell your computer the key words it's about - like FUELLESS CAR - and then tell your computer where you've put the writeup on it. Like, "file of October '76, Item Number 4."

Then, say a few months later you want to look up the article again. You just tell your computer the KEY WORDS - type in FUELLESS, CAR - and your computer tells you exactly where to look for the writeup! (You might have collected several items on this subject. Your computer would then tell you about all of them.) It's like your own private library.

If you'd collected a lot of other material on cars, then you could see it all by just typing in "CAR". Or maybe you're interested in anything that runs without fuel. Then you'd just type in FUELLESS, and get back more than cars. See?

INDEXING is like a great scrabble game. It also tends to make you smarter, for some reason. (Certain studies prove this.) Maybe that's because indexing develops both your logic and fine judgment at the same time.

There are several ways in which indexing can be built into a business. Cutting and selling newspaper clippings is who's likely to pay most for what subjects. Likewise, which publication. Your computer does a superb job of this. Another business possibility is abstracting the literature in a given field. The abstract can be sold, and so can the indexes!

Some people just "take" to indexing - it's another one of those areas for unpredicted

flair — just like programming. So, you mightn't have guessed what fun for YOU lies hidden in a computer!

NOW... your home computer can act as Doorman, Guard, Secretary, Librarian, Mother's Helper, Consultant, New Business Partner, Companion, and just general Unpaid Labor Force.

But bet you never thought of it as an ENTERTAINER!

You're hearing a lot about "TV Games" for this Christmas. They're just "little black boxes" you hook up to your TV. You play games with them through a little keyboard, using the TV screen as a "display".

Know what those "little black boxes" are? They're microcomputers, that's what! Someone else has programmed them, is all.

Music

One of personal computing's greatest promises is in MUSIC. Imagine yourself going to your piano or organ, and playing as slowly as you wish! ...Maybe one finger at a time, or making it up as you go. All the while, your computer is "listening". It's remembering every note. (And no-one else has to hear you.. you can wear headphones!)

When you're finished with your "composition", you tell your computer "play it back - to tempo". Your computer has the piece played back, as fast as you want it to go.

There you are — you've made your own player piano or "player organ"! And what's it playing? YOUR piece! Speak of ENTERTAINMENT--!! This is an indescribable experience, if you've always longed to make music, but never got the chance. A COMPUTER IS YOUR CHANCE TO MAKE MUSIC, AT LAST -- WITHOUT HAVING TO PLAY!!

Present-day organs are moving in this direction, but they still require a lot of skill. You have to be able to "keep up with the rhythm", for example. With your own computer, you won't have to: the rhythm keeps up with you!

How Your Organization Can Use a Computer

A computer can be a real asset for a SMALL ORGANIZATION. It's indispensible in event planning, maintaining calendars and schedules, bookkeeping, assigning and tracking committee work, reporting, calculations, issuing ads and announcements, doing memos, and keeping the minutes.

These can all be INDEXED to let you find the exact material you were thinking about, stored in the computer four years ago!

Your computer is also the center of attention at social events, as it handles ticketing, door prizes, drawings and countless crowd-participation games. Imagine the attraction THAT is, with you as the sponsor!

School

Speaking of community, what if you took a

computer like this to SCHOOL? It could make you
a hero. And since you programmed it, of course,
it's ready to do whatever you planned.

As a teacher you could use it as a TUTORIAL
AID and to do INSTANT GRADING, for example.
Perhaps above all, you could use it to TEACH
PROGRAMMING. And the students would have to
write programs to DO something, so they start
coming up with interesting applications. (Again
programming boosts mathematical ability — it's
not the other way around, as often supposed.)
The kids find themselves exposed to possibili-
ties like all these we're talking about. They
catch on quick what to do with them!

Perhaps one of the most important things
kids can discover is how to earn a living in a
way that feels creative to them. And again we
see the LITTLE computers playing a BIG part in
exactly this discovery!

Meanwhile, the little computer can help
with HOMEWORK. And it can feature in all sorts
of projects and reports, ones that get writeups
in local papers. This often happens with stud-
ent computer projects, because they tend to be
so original. Also, one student can do the work
of many, this way. And likewise, many scien-
tific projects become possible with a computer,
which were impossible before.

VI The Next Step

Where do you go from here? Well, my sug-
gestion is more reading. Take what I have given
you today and expand on it. There are a number
of books out in publication now that will enable
you to understand better all this that is going
on and in much more expanded form than I have
done.

Make use of your local computer store! The
owners and operators are more than willing to
give you more of an understanding and help you
over the hard bumps of decision-making process
of which one to buy and what to hook up to it.
Also look into classes at your local high
school and college. There are any number of
classes to take to further your knowledge in the
field.

As a final note, remember: Computers are
not just for geniuses, you don't have to be a
special gifted person to own or operate one. It
is a tool that you can use, that will help in
every day life.

ELECTRONICS FOR THE HANDICAPPED

Dr. Robert Suding
Research Director for the Digital Group
Box 6528
Denver, Colorado 80206

Abstract: "Handicapped" can refer to either physically impaired or mentally impaired. There is a need for individual hobbyists to become involved in aiding these handicapped people. Present technology supports a large number of exciting applications. Hobbyist magazines are begging for articles on how to implement systems for the handicapped. An international organization of hobbyists interested in handicapped applications was formed at PCC '77(Atlantic City Convention). For more information or to join contact:
Computers for the Handicapped
c/o Warren Dunning
5939 Woodbine Ave.
Philadelphia, PA 19131

MICROCOMPUTER COMMUNICATION for the HANDICAPPED

By Tim Scully
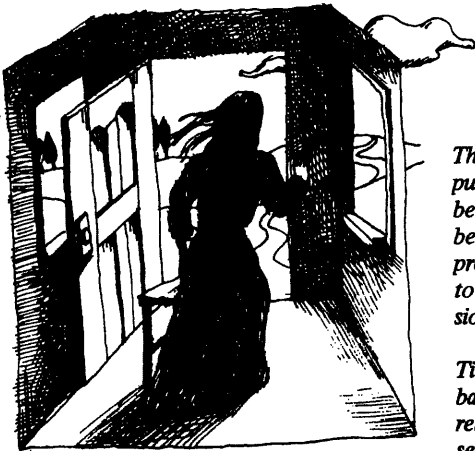
This article is more technical than many published in People's Computers, *but we believe that the general discussion will be interesting, informative, and thought provoking to all, even those who choose to skip the program listings and discussion.*

Tim Scully has been designing biofeedback equipment and doing biofeedback research for many years. Tim is a Research Fellow of the Humanistic Psychology Institute; he is now working towards his doctorate in psychology. His dissertation project involves researching and developing biofeedback systems and techniques for use in drug rehabilitation.

Tim is also teaching a computer class to fellow inmates at a Federal penitentiary. Although prison resources are scarce and he is not allowed to solicit donations, he is hopeful of somehow eventually acquiring a computer system for the prison.

The potential of microcomputers as tools for the handicapped is enormous and exciting: we encourage dissemination of such information. For this reason we are making copies of this article available. To receive a reprint, send a stamped, self-addressed envelope (24¢ for business size, 35¢ for 8½ by 11 inch) to People's Computers.

How would you communicate if you couldn't talk, didn't have the use of your hands, and could only somewhat control the movements of one knee? This is the problem which Robin, a young lady in her 20's has lived with all her life. She has cerebral palsy.

I met Robin in 1976, and this is the story of how a microcomputer communication system came to be built for Robin. The general concepts applied in the development of Robin's communication system may prove helpful in the development of microcomputer systems for other handicapped people.

When I first met Robin, her communication was accomplished by use of a word wheel. She could understand speech and she could read, but she needed help in 'talking'. Her word wheel was made from an electric clock motor and a bicycle spoke, with the bicycle spoke attached where the second hand of a clock would normally be mounted. A sheet of cardboard was mounted behind the spoke, with the letters of the alphabet on it, arranged in a circular pattern. The spoke pointed to the letters, one at a time, as it rotated. Robin could move her knee to one side and hit a kneeswitch mounted on her wheelchair, thus stopping the motor so that the spoke would freeze, pointing at the letter she had chosen.

The spoke rotated at one revolution per minute, so spelling proceeded at about one letter per minute! The person Robin was conversing with often had to write the letters down, to keep from forgetting them, as a message slowly built up. To speed up the communication process, a few words were written next to each letter of the alphabet, so that when the spoke stopped it would point at a group of words as well as a letter. The person with whom she was conversing would have to guess which of these Robin intended. It took considerable patience to hold a conversation with Robin, and not very many people took the time;

When I first saw Robin's communication system, I thought of replacing her word wheel with a microcomputer and video

display, using a vocabulary of words stored in the computer's memory in place of the sheet of cardboard. A little over a year later, that system now exists and is being installed on Robin's wheelchair.

HOW IT WORKS

The present system is an expansion of the word wheel concept which uses a TV display with 16 lines of text. The top line is reserved for the display of a 'menu' of items (words, letters of the alphabet, punctuation symbols or control codes) from which Robin can choose. The second line is kept blank and the bottom 14 lines provide space for the display of a message of about 200 words.

As items are displayed on the menu, Robin can choose one by hitting the kneeswitch mounted on her wheelchair. In some modes of operation several items will appear on the menu at once, in which case the item at the left is the current item, the one which can be selected by hitting the kneeswitch.

On start-up, the system blanks the TV screen and then offers the SPELLING? mode by putting that word on the menu. This item remains on the menu for a time 'T1' (an adjustable time delay). If the kneeswitch is hit during that time, the SPELLING? mode is entered, otherwise the next menu item is displayed: PUNCTUATION?. If that item isn't chosen either, after another delay equal to T1, then the system will begin displaying the names of groups of words: A-BONE, BOOK-CROWN, CRY-FINGER, FINISH-HIDE, HIGH-LOT, LOUD-OUGHT, OUR-ROSE, ROSE ANN-STAY, SQUARE-TWENTY and TWO-YOURSELF, one group at a time. Each group of words contains about 120 words in alphabetical order. The name of each group is made up from the first and last words in the group.

If Robin doesn't pick any group of words, the computer then offers an ESCAPE? from the groups of words. If this isn't chosen, the names of the groups are offered again. If the ESCAPE? is chosen, the system returns to near the beginning of the program and offers SPELLING? again. This ESCAPE? to the beginning is offered from every mode of system operation.

If Robin does pick a group of words, HIGH-LOT for example, then the names of subgroups in that group begin being displayed, one at a time: HIGH-HONOR, HOPE-HUNT, HURRY-IMPORTANT, IN-INTERESTING, INTO-I'VE, JENNIFER-JUMP, JUST-KISS, KITCHEN-LAKE, LAND-LEAST, LEAVE-LIE, LIFE-LITTLE, LIVE-LOT and then ESCAPE?. If Robin picks a subgroup, such as LEAVE-LIE, then the words in that subgroup are displayed across the top line of the TV, with two spaces between each word:
LEAVE LED . . . LIBRARY LIE

If Robin hits the switch at this moment, LEAVE will be transferred down to the first available space in the message area of the TV screen and the menu will begin all over again by offering SPELLING?. If the first word, LEAVE, isn't chosen, then after the usual time delay T1, the list of words on the menu will shift one to the left, so that LED is on the extreme left and it becomes the current item. This process continues until a word is chosen or until the end of the subgroup, LIE. If LIE isn't chosen, ESCAPE? is offered, and if it isn't chosen, the complete list of 11 words in the subgroup is displayed across the menu and the cycle begins again.

By this system of groups of words, subgroups, and finally words, it is possible for Robin to look through a list of 1200 words in a short time, find the one she wants and add it to a message she is assembling on the TV screen. The computer automatically adds a space after each word chosen, so it isn't necessary for Robin to worry about spacing between words—she can just choose one word after another. All letters and words are upper case, so she doesn't have to shift.

When a sentence is complete, and when she wants punctuation symbols, Robin can select the PUNCTUATION? mode. The first item offered on entering this mode is CONTROL? and if that isn't chosen, then after the usual time delay, the punctuation symbols will be spread across the menu in much the same way that the words in a subgroup were displayed:
. ' ? ; : ! 0 1 2 . . . 9 # $ % & ( ) * + −

These items leave the screen at the left, one at a time, if they are not chosen. If one is chosen, the computer backspaces once (to undo the automatic spacing) and adds the chosen symbol to the message on the screen. Then the system starts over by offering SPELLING? again.

The CONTROL? mode offers Robin a few useful commands, one at a time, if it is chosen: BACKSPACE?, ERASE LAST WORD?, SPACE?, ERASE SCREEN?, and NEXT LINE?. These control codes operate immediately if selected. Then the system starts over by offering SPELLING? again.

The SPELLING? mode exists to allow Robin to spell words not found in the 1200 word vocabulary stored in the computer's memory. To speed up the process of spelling, letters of the alphabet are not offered in alphabetical order. Instead they are offered in the order of their probability of use in English. Except at the beginning of a word, the likelihood of a letter appearing in a word depends on the last letter chosen.[†] If we are in the middle of a word, and the last letter chosen was 'A', then the most likely next letter is 'E', the second most likely is 'B', etc.

Robin's system has 27 different alphabets stored in it. The first alphabet has the letters organized so that those most likely to appear at the beginning of a word will be displayed first. This is the alphabet which appears when the SPELLING? mode is first entered. The letters are spread out along the menu line as usual, with the first offering on the left. If no letter has been chosen by the time all of them have moved off the screen to the left, the usual ESCAPE? offering is made and the alphabet redisplays.

If a letter is chosen, it is added to the message area of the screen, and ESCAPE? is offered on the menu. If Robin decides to stay in the spelling mode, the computer then displays one of the 26 remaining alphabets—which one is determined by the letter she just chose. When she picks a letter from this new

† Mr A Ross Eckler suggested the bigram spelling scheme used in Robin's system. He supplied me with letter use frequency tables which he credited to F Pratt, *Secret and Urgent: The Story of Codes and Ciphers*, Blue Ribbon Books, 1942 pp 258-259.

alphabet, it is added to the message, immediately after the first letter (the system automatically backspaces to undo its automatic spacing). This process continues until she has completed spelling a word. Then she picks ESCAPE?, which returns her to the beginning of the program, which offers the SPELLING? mode, and a space is left after the word she has just completed.

This spelling scheme allows comparatively rapid spelling of words because Robin only has to wait for a few letters to display before the one she wants is likely to become the current item. The automatic spacing also speeds up communication.

Now that we've looked at what Robin's system does, let's examine the hardware and software which do the work.

SYSTEM DESIGN

Robin's system was designed around the special limitations of her situation and my own situation. I met Robin through a United States Probation Officer, who was supervising me while I was temporarily free on appeal bond. I was waiting for the Court of Appeals to decide if it would uphold my conviction for conspiracy to manufacture LSD (back in 1968 and 1969). As it turned out, the Court did uphold my conviction, and I'm now serving a 10 year Federal prison term at McNeil Island Penitentiary in Washington.

My personal problems limited the system design to the use of a commercially available computer kit because of the difficulty of sending materials into prison. Robin's family had only a limited budget, and Robin's capabilities formed the remaining design limits.

In 1976, the budget we had (about $1,300) was just about enough to buy a computer kit with keyboard, cassette tape system, video monitor and 8K of memory, so this is the size system we planned on. The average word in English is about 5.5 characters long and we initially planned on a vocabulary of about 1,000 words, which uses up 5,500 bytes of memory. This left about 2,500 bytes for the program to control the system together with storage for spelling and punctuation symbols.

That's not enough memory for the use of a high level language such as BASIC, so the program had to be written in assembly language. Since my previous assembly language experience was with the 8080A, this was the CPU chosen for Robin's system.

We wanted the system to be expandable. In the future, Robin may want to add more memory, a printer, a speech synthesizer or other additional peripherals. For maximum flexibility in expansion, the S-100 bus structure was chosen because of the wide range of commercially available plug-in circuit cards. The computer also had to be small and light enough to mount under the seat of Robin's wheelchair. In order to modify the menu and message areas of the video display independently, the computer needed a memory-mapped video display. These constraints pointed us toward the Polymorphic Systems' Poly 88 System 4 kit.

The Poly 88 uses a 5 slot S-100 chassis, which makes it small and fairly light in weight. The Poly video card is memory mapped and displays 16 lines of 64 characters each—just right for Robin. The features of the Poly CPU card were also useful: it has 512 bytes of RAM together with a monitor program in ROM. A cassette tape interface card works together with tape loading software in the monitor ROM to handle program storage and loading.

The vocabulary for Robin's system is stored in RAM because we expect her vocabulary needs to change once she can communicate more freely. The problem with storing vocabulary in RAM is that RAM is volatile—the memory and thus the vocabulary are erased every time the computer is unplugged. So a battery back-up card was added to the system. This card keeps the program and vocabulary stored in RAM even though the computer may be unplugged for hours at a time while Robin's wheelchair is moved from place to place. Robin's computer uses the Seals Electronics BBUC card with NiCad batteries.

We had, at one point, considered battery powering the entire system, but ended up rejecting the idea. A large and heavy battery would have been required for reasonable life, and this would bring the

total weight of the wheelchair and system up so high that Robin's mother wouldn't be able to lift it in and out of their family van for trips to school and other errands. As it is now designed, Robin's system has to be plugged into a wall outlet to operate, but the battery back-up card keeps memory alive while the system is unplugged so that it is instantly ready to start upon being plugged in.

HARDWARE MODIFICATIONS

A few additions and modifications were made to adapt the commercially available hardware to Robin's application. The Poly 88 chassis has only two controls: an on/off switch and a reset pushbutton. This is because it is designed to use a keyboard for functions which a control panel might perform. The reset pushbutton starts the ROM cassette tape loading program. I added a second pushbutton which activates a vectored interrupt and jumps to the beginning of Robin's program. This makes it possible to start up Robin's system without the keyboard. A schematic for this simple addition is shown in Figure 1.
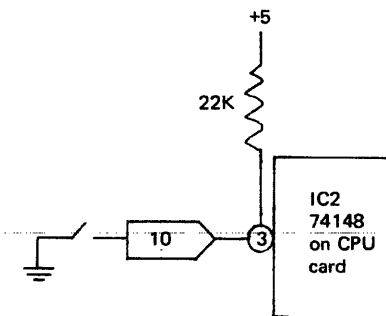
Figure 1

As a computer powers down, it can scramble data stored in memory by sending out false write commands. To eliminate this problem, the memory in Robin's system was partitioned so that an 8K block of RAM, containing the main program and stored vocabulary, could be write protected. This left only the 512 bytes of RAM on the CPU card unprotected (and the memory mapped video display, of course). The small CPU RAM area is used for all scratchpad functions and is one of the features of the Poly CPU card which encouraged its selection.
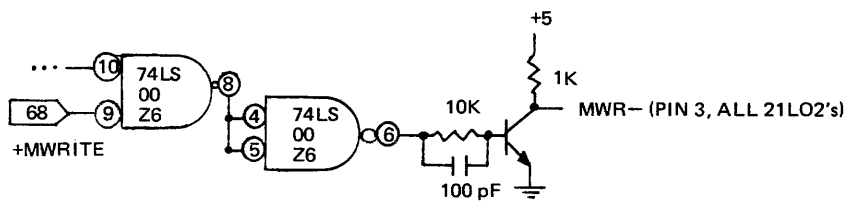
Figure 2
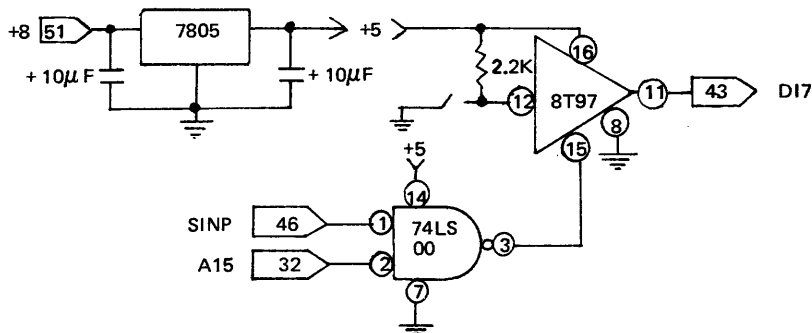
Figure 3

The RAM card used in Robin's system is an Industrial Microsystems IuS #000231 8K card which uses 21L02-4 chips. This card was modified slightly so that a toggle switch could be added to the computer's front panel which protects/unprotects the main 8K RAM. When loading new programs from cassette tape, RAM is unprotected. Otherwise it is protected. A schematic of this circuit is in Figure 2.

The final hardware modification for Robin's system was the addition of an input port for her kneeswitch. Figure 3 shows the schematic for this circuit, which was built on a small scrap of Vectorboard and mounted on the Poly 88 chassis.

SOFTWARE DESIGN

The program for Robin's system is listed, with comments, on the following pages. It was kept as brief and simple as possible to leave as much space in memory as possible for the storage of vocabulary. The vocabulary is stored as ASCII, with one character per byte of memory. ASCII doesn't use the eighth bit of an eight bit word, so I used the eighth bit as a 'beginning of word' flag. The first character of any character, word or phrase

stored in memory has the eighth bit true, and all following characters (if any) have the eighth bit zero. This scheme allows the words in Robin's vocabulary to be packed tightly in memory. The only extra bytes of memory used are flags inserted at the end of each subgroup (FDH), group (FEH) and at the end of the vocabulary (FFH).

The main program uses one subroutine from the Poly 4.0 monitor ROM. That routine, WH1, outputs a character to the video display. It uses a location in the CPU board RAM, POS, to store the next position it will print into and it recognizes several control codes:

0DH = carriage return and line feed
0CH = erase screen and send cursor home (upper left corner of screen)
0BH = send cursor home without erasing screen
18H = erase current line

The starting address for the memory area mapped by the video display is F800H. Thus, if the control code 18H is in the A register when WH1 is called, it will stuff F800H into POS. WH1 saves all registers on entry and restores them on exit.

The monitor ROM on Robin's CPU board is a slightly modified version of the 4.0 monitor: at address 0008H a JMP 2000H

has been inserted so that vectored interrupt VI6 jumps to the start of the main program. This allows a single pushbutton to start Robin's system.

Robin's software was hand assembled because I didn't have an assembler program to run on her system. The program listings were typed by hand and may contain a few errors.

TEXT AND EDITOR PROGRAMS

The TEXT and EDITOR programs written for Robin's system are both very short. TEXT was used to enter the messages, alphabets and vocabulary into her system's memory from the keyboard. EDITOR is used to modify her vocabulary and to add to it after the original entry. Here is what they do in detail.

TEXT is entered with a starting address in HL. The TV screen is erased and the system waits for text to be entered from the keyboard. Any unshifted letter is printed on the TV screen as a lower case letter but is stored in memory (beginning at the starting address in HL) as upper case ASCII with the eighth bit zero. The keyboard for Robin's system is a Teletype-like keyboard and does not have lower case letters, so the 'unshifted' letters are actually upper case, but TEXT translates them for display purposes.

Any letter of the alphabet typed while the CTRL key is held down (except Z) is printed on the TV screen as a capital letter and is entered into memory as upper case ASCII with the eighth bit turned on. This allows the first letter of any word or phrase to be identified. The Poly monitor program uses CTRL Z as a command to enter its front panel mode, so this is the one exception to the rule stated above. Shift O jumps to the EDITOR program, at the current address. Rubout erases the last character entered.

TEXT is also capable of inserting the control codes which identify the end of alphabets, subgroups and groups.
CTRL shift L = insert FBH
CTRL shift N = insert FDH
CTRL shift O = insert FEH

EDITOR is a somewhat longer and more complex program which allows the user to examine the text stored in the system's memory. It also allows modifications of
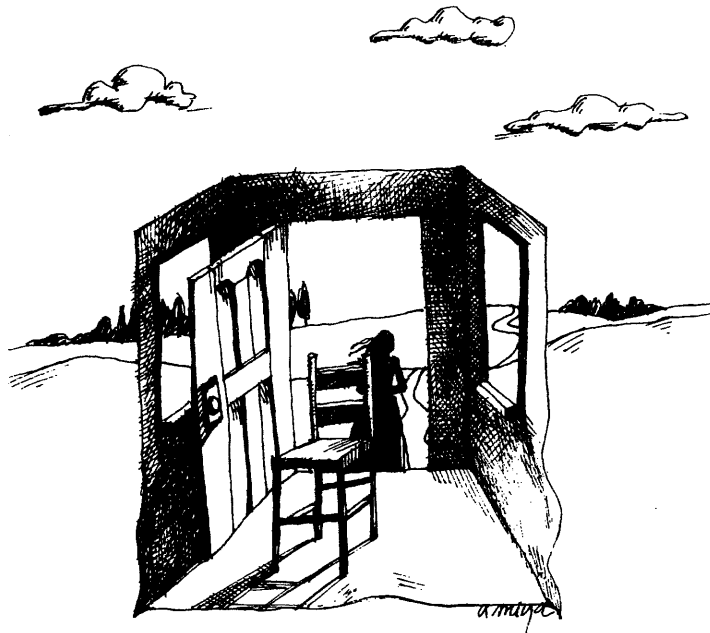
that text by insertions and deletions. If a deletion is made, all of the rest of the text (at addresses greater than the deleted address) is moved down one memory location to close the gap. If an insertion is made, all of the rest of the text is moved up one location to make room for the addition.

EDITOR is entered with a starting address in HL. Upon entry it will display a 'line' of text, beginning at that address. At the left end of the line, the current starting address will appear, in hex, followed by a space. Then the contents of memory are printed, up to and including the first 'control code' found. Any letters stored in memory with the eighth bit high will print on the TV as capitals, while those with the eighth bit low will print as lower case. The control codes will print as special symbols:
FBH = { FDH = } FEH = ∿ FFH = ■

The EDITOR recognizes several commands, as listed below:
carriage return = display next line
line feed = display previous line
space = redisplay the current line, shifted one character to the left
   NOTE: insertions made by EDITOR will go just in front of the first character on the display. The space is used to move along the current line so that insertions (or deletions) can be made in the middle of a line.

Shift O = jump to TEXT with HL equal to the starting address of current line.
CTRL shift L = insert capital L
CTRL shift M = insert capital M
CTRL shift O = insert FEH
CTRL shift N = insert FDH
rubout = delete first character of current line
any unshifted letter = insert that letter with eighth bit low
CTRL any letter except L, M, or Z = insert that letter with eighth bit high.

The EDITOR and TEXT programs use several more subroutines from the Poly 4.0 monitor ROM. Either program is entered with a starting memory address in HL. The monitor program allows register pairs to be pre-loaded from the keyboard while it is operating in the 'front panel' mode. For a detailed explanation of this procedure, see the Poly system manual Volume 2 p58-65. The other subroutines used are:
WHO = fetches a character from the keyboard and returns it in A. No other registers are affected.
DEOUT = print the two byte number in DE as a four character hex number.
MOVE = move —BC bytes from the area starting at (HL) to the area starting at (DE) — only works for moving to lower addresses.

Robin's main program turned out to be shorter than expected. Including the

alphabets and punctuations symbols, it is 1250 bytes long. Even with 1,200 words of vocabulary in memory, there is still room for TEXT and EDITOR to remain in memory so that Robin's family can revise her vocabulary as needed.

A FEW WORDS ABOUT WORDS

It may be helpful to briefly mention how the initial vocabulary for Robin's system was chosen. The first 1,100 words were supplied by Robin's tutor, from lists of the first words taught in English. The remaining words were chosen by Robin and her family. These include the names of people, places, articles of clothing, foods and other objects which Robin comes in contact with.

As practical experience with the system is accumulated, revisions may be made in the initial vocabulary and possibly in the main program. For example, it may turn out that Robin will feel more comfortable spelling words than looking them up in the stored vocabulary. If this is the case, we may try adding a set of look-up tables for prefixes, roots and suffixes to speed up communication.

FUTURE DIRECTIONS

The basic system built for Robin can be expanded and modified to fit a wide range of possible situations. For example, the kneeswitch could easily be replaced by an electromyograph (EMG), an instrument which measures the electrical signals associated with muscle tension. An EMG can easily detect levels of muscle tension which are too weak to control a switch mechanically. This is a practical alternative to the kneeswitch for people who are only capable of very limited movement, such as an eyelid twitch. There are many hospitalized patients who experience extreme frustration because they are conscious but cut off from communication. Microcomputer communication systems of some kind may eventually become standard hospital equipment, and could help to make such patients' lives much more rich and meaningful.

There is a wide range of possible options for expanding Robin's system. It would be easy to add a printer, for example. She could assemble a message on the TV screen as usual, and then select a

'print' command which would cause the message area of the TV screen to be copied on paper. This would allow her to write an essay or a letter.

An S-100 compatible card is available from D. C. Hayes Associates (the 80-103A Data Communications Adaptor) which would allow her to select and dial a telephone number and send messages over the telephone to anyone having a computer terminal. A number of computer networks are now being used as communication networks, and it is reasonable to expect a network for handicapped people to develop in the near future.

Computers can be used to generate and control sounds. Several companies now offer S-100 compatible speech synthesis cards. It would be possible for Robin to learn to speak out loud, using one of these cards. Although the necessity of learning a new 'language' of phonemes would initially make this a slow communication process, the potential exists for this to be a very rapid communication mode.

Several companies now offer S-100 compatible circuit boards for music synthesis. It would be possible to write a program which would allow Robin to compose music and instruct the computer to perform it for her. Computer graphics are also possible. With a higher resolution video display, it would be possible for her to draw pictures with fine detail, and with a suitable printer, make 'hard copies' of these on paper.

There are several CMOS microprocessor CPU chips available now. Although CMOS memory and peripheral chips are still somewhat more expensive than TTL and NMOS chips used in Robin's system, it is already practical to build a microcomputer system similar to Robin's which would consume much less power. Such a system would be more expensive, but would be capable of battery operation, increasing portability.

S-100 compatible circuit cards are readily available which allow a computer to control electrically operated devices in its surroundings. It would be easy to expand a system like Robin's to allow her to turn on and off lights, appliances, etc. A system which can communicate can be a flexible control system too.

If you decide to try to build a microcomputer communications system for a handicapped person, I'd like to hear from you. I may be able to help with advice, and Robin might benefit from your ideas. My mailing address is:
Tim Scully
35267-136 CH
P O Box 1000
Steilacoom, WA 98388

Programs

ROBIN'S MAIN PROGRAM

This is Robin's main program

Address	Label	Bytes	Instruction	Comment
2000	START	3E0C	MVI A,0CH	erase screen
2002		CD240C	CALL WH1	
2005		CD4922	CALL PATCH	
2008	ENTERL	22820C	SHLD CM	save current message address
200B		00	NOP	
200C	REENTRY	215622	LXI H,SPELM-1	
200F		CDFF20	CALL MESSAGE	offer spelling
2012		CA8F21	JZ SPELL	
2015		215F22	LXI H,PUNM-1	
2018		CDFF20	CALL MESSAGE	offer punctuation
201B		CAD321	JZ PUNCT	
201E		217F25	LXI H,BEGIN	start of vocabulary
2021		11FE00	LXI D,00FEH	set flags
2024		CD8120	CALL MENU	offer groups
2027		11FD00	LXI D,00FDH	set up flags
202A		2A840C	LHLD CI	fetch chosen item's address
202D		CD8120	CALL MENU	offer subgroups
2030		11FC01	LXI D,01FCH	set up flags
2033		2A840C	LHLD CI	fetch chosen item's address
2036		CD8120	CALL MENU	offer words
2039		2A840C	LHLD CI	fetch chosen item's address
203C	XMIT	7E	MOV A,M	fetch character
203D		CD240C	CALL WH1	print it
2040		23	INX H	next
2041		7E	MOV A,M	fetch it
2042		E680	ANI 80H	check for start of next item
2044		CA3C20	JZ XMIT	if not next item, keep printing
2047		3E20	MVI A,''	space after completed item
2049		CD240C	CALL WH1	
204C	ELOP	2A0E0C	LHLD POS	fetch current message address
204F		C30820	JMP ENTERL	go back to offer spelling again

The following subroutine is used by MENU to save the 'flags' which start out in DE (the flag in D is 0 unless MENU is asked to display words or individual characters, the flag in E is the indicator telling MENU what to display, eg. FEH = groups, FDH = subgroups, etc.), it also saves the data in HL as its starting address and saves the current value of POS in the 'current message' storage.

Address	Label	Bytes	Instruction
2052	ENTER	EB	XCHG
2053		22860C	SHLD FLAGS
2056		EB	XCHG
2057	SENTER	22800C	SHLD CS
205A		2A0E0C	LHLD POS
205D		22820C	SHLD CM
2060		2A800C	LHLD CS
2063		C9	RET

The following subroutine erases to top two lines of the video display without disturbing the message displayed on the bottom 14 lines. It ends with the cursor at 'home'.

```
2064  3E0B      NEW     MVI A,0BH
2066  CD240C            CALL WH1         send cursor home
2069  3E18              MVI A,18H
206B  CD240C            CALL WH1         erase current line
206E  3E0D              MVI A,0DH
2070  CD240C            CALL WH1         go to next line
2073  3E18              MVI A,18H
2075  CD240C            CALL WH1         erase it too
2078  3E0B              MVI A,0BH
207A  CD240C            CALL WH1         send cursor back home
207D  C9                RET
```

MAJOR SUBROUTINES: SMENU AND MENU

SMENU and MENU, which follow, are the major subroutines for displaying items on the menu (the top line of the video display). MENU is entered with flags in DE and a starting address in HL. The flags tell MENU to display groups, subgroups, words or individual characters. The starting address tells MENU where to find the first item to display. An exit from MENU is accomplished when an item is selected by use of the kneeswitch. Upon exit from MENU, the starting address of the chosen item will be in CI.

```
207E  11FB01    SMENU   LXI D,01FBH
2081  CD5220    MENU    CALL ENTER      set flags for spelling
2084  CD6420    ITEM    CALL NEW        save addrss & flags
2087  22840C            SHLD CI         erase menu
208A  7E        DISPY   MOV A,M         save current item address
208B  CD240C            CALL WH1        fetch character from memory
208E  23                INX H           and display it
208F  7E                MOV A,M         next
2090  E680              ANI 80H
2092  CA8A20            JZ DISPY        check for msb=1
2096  AF                XRA A           if not, keep printing
2096  BA                CMP D           are we finished with group or
2097  C26421            JNZ WORD        are we printing with words or letters?
209A  14                INR D           if so, go on with words or end
209B  3E2D              MVI A,'_'       otherwise, set flag
209D  CD240C            CALL WH1        print '_'
20A0  2B                DCX H
20A1  23        SEARCH  INX H           and look for end of group or
20A2  7E                MOV A,M         subgroup
20A3  BB                CMP E           by checking for a flag like E
20A4  DAA120            JC SEARCH       keep looking until found
20A7  2B        BACKUP  DCX H           then backup
20A8  7E                MOV A,M         and print it
20A9  E680              ANI 80H
20AB  CAA720            JZ BACKUP
20AE  C38A20            JMP DISPY
```

The next four locations store the timing constants for two time delays: T1 and T2. T1 is the time each item on the menu is displayed and T2 is the minimum time the kneeswitch has to be closed before it is considered intentional (so that accidental twitches will be ignored).

```
20B1  5050      DW 5050H    T1 time constant
20B3  5050      DW 5050H    T2 time constant
```

SUBROUTINE: SWITCH

The subroutine SWITCH looks for a switch closure for time T1 and then returns with zero in D if the switch was never closed. If the switch closes, but not for at least T2, the routine just starts over, extending T1. If the switch closes for at least T2, then after the switch is released, it returns with one in D.

```
20B5  1600      SWITCH  MVI D,0
2087  E5                PUSH H          set up 'never closed flag'
20B8  2AB120            LHLD T1         fetch time constant
20BB  E5                PUSH H
20BC  C1                POP B           put it in BC
20BD  E1                POP H
20BE  DB80      IN      IN 80H          look at switch
20C0  E680              ANI 80H         it's only one bit
20C2  CADA20            JZ CLOSED
20C5  22900C            SHLD 0C90H      waste time
20C8  2A900C            LHLD 0C90H      to make timing loop longer
20CB  22900C            SHLD 0C90H
20CE  2A900C            LHLD 0C90H
20D1  0D                DCR C
20D2  C2BE20            JNZ IN          check switch every time
20D5  05                DCR B
20D6  C2BE20            JNZ IN
20D9  C9                RET             keep timing
20DA  E5        CLOSED  PUSH H          time up, no contact
20DB  2AB320            LHLD T2         fetch time constant
20DE  E5                PUSH H
20DF  C1                POP B           put it in BC
20E0  E1                POP H
20E1  22900C    WAIT    SHLD 0C90H      waste time
20E4  2A900C            LHLD 0C90H
20E7  0D                DCR C
20E8  C2E120            JNZ WAIT        keep timing
20EB  05                DCR B
20EC  C2E120            JNZ WAIT
20EF  DB80              IN 80H          time up?
20F1  E680              ANI 80H         check switch
20F3  C28520            JNZ SWITCH      it's only one bit, the msb
20F6  14                INR D           start over if not still closed
20F7  DB80      UP      IN 80H          set flag for contact
20F9  E680              ANI 80H         check switch again
20FB  C0                RNZ
20FC  C3F720            JMP UP          wait until it is released
                                        meanwhile looping
```

SUBROUTINE: MESSAGE

The subroutine MESSAGE is used to display a number of short messages on the menu. Message is entered with an address in HL equal to one less than the starting address of the message to be displayed. It will display the message found, up to and including a terminating '?'. Upon exit from message, the zero flag in the PSW will be one if the offered item was chosen and zero if it was not chosen.

20FF	000000	MESSAGE	NOP NOP NOP	I deleted something here
2102	CD6420		CALL NEW	erase menu
2105	23		INX H	
2106	7E		MOV A,M	
2107	CD240C		CALL WH1	print
210A	FE3F		CPI '?'	check for end of message
210C	C20521		JNZ MESSAGE +6	
210F	2A820C		LHLD CM	
2112	220E0C		SHLD POS	restore POS
2115	CDB520		CALL SWITCH	
2118	3E01		MVI A,1	
211A	BA		CMP D	
211B	C9		RET	

SUBROUTINE: COMP

The subroutine COMP is used by MENU to check the switch.

211C	CDB520	COMP	CALL SWITCH	
211F	3E01		MVI A,1	
2121	BA		CMP D	
2122	C22C21		JNZ NEXT	if no contact, offer next choice
2125	2A820C		LHLD CM	
2128	220E0C		SHLD POS	restore main text POS
212B	C9		RET	

MORE ROUTINES USED BY MENU

The following chain of routines are used by MENU to find and display the next item, check for the last item in a list, offer ESCAPE? and recycle to the beginning of the list if nothing is chosen. The details of these operations vary depending on what items are being offered: groups, subgroups, words or characters.

212C	EB	NEXT	XCHG	save current address
212D	2A860C		LHLD FLAGS	while restoring flags
2130	EB		XCHG	
2131	7B		MOV A,E	
2132	FEFD		CPI FDH	are we displaying groups or subs?
2134	D24121		JNC CHECK	if so, check for end
2137	2A840C		LHLD CI	
213A	23	FIN	INX H	skip current word or letter
213B	7E		MOV A,M	
213C	E680		ANI 80H	and keep skipping until the
213E	CA3A21		JZ FIN	start of the next, then check

2141	1C	CHECK	INR E	the last item will be followed
2142	7E		MOV A,M	by a flag = to E + 1
2143	BB		CMP E	
2144	D25721		JNC LAST	
2147	1D		DCR E	restore flag in E
2148	FEFB		CPI FBH	if no control code found,
214A	DA8420		JC ITEM	keep displaying
214D	7B		MOV A,D	
214E	FEFD		CPI FD	
2150	DA5721		JC LAST	skip control code
2153	23		INX H	
2154	C38420		JMP ITEM	
2157	CD8221	LAST	CALL ESCAPE	if last item was displayed, offer
215A	2A860C		LHLD FLAGS	escape and then loop back
215D	EB		XCHG	
215E	2A800C		LHLD CS	
2161	C38420		JMP ITEM	and start displaying over again

SUBROUTINE: WORD

WORD, The next subroutine, is used by MENU. If groups or subgroups are being offered, it is entered only after the complete offering has been printed and it jumps to COMP to check the switch. But if individual words or characters are being offered, WORD keeps printing words or characters across the menu space, with two spaces between each, until the end of the subgroup or until the end of the line.

2164	7B	WORD	MOV A,E	check flag
2165	FEFD		CPI FDH	
2167	D21C21		JNC COMP	and split if groups or subs
216A	3A0E0C		LDA POS	check position in menu
216D	FE3C		CPI 3CH	if we are near the end of the line,
216F	D21C21		JNC COMP	
2172	7E		MOV A,M	stop printing & split or if we are
2173	BB		CMP E	at the end of the subgroup, split
2174	D21C21		JNC COMP	
2177	3E20		MVI A,' '	otherwise,
2179	CD240C		CALL WH1	print two spaces
217C	CD240C		CALL WH1	
217F	C38A20		JMP DISPY	and add more to menu

SUBROUTINE: ESCAPE

The subroutine ESCAPE offers a return to the SPELLING mode and is used often.

2182	C5	ESCAPE	PUSH B	
2183	214F22		LXI H,ESC-1	set up for message
2186	CDFF20		CALL MESSAGE	
2189	C1		POP B	
218A	C0		RNZ	return if no escape
218B	E1		POP H	clean up stack
218C	C30C20		JMP REENTRY	and reenter SPELLING?

SUBROUTINES USED BY SPELLING MODE

The SPELLING mode uses this chain of subroutines. The first alphabet offered is different from the other 26, and the routine doesn't backspace before printing the first letter, so there is one routine for the first letter and another for all the others. ESCAPE? is offered after each letter is printed and before a new letter is offered. A look-up table is used to pick the right alphabet to offer after the first letter has been printed.

Addr	Hex	Label	Instruction	Comment
218F	211523	SPELL	LXI H, ASTART	address of initial alphabet
2192	CDB721		CALL FIRST	print first letter
2195	CD5720		CALL SENTER	to restore POS
2198	CD8221	TALE	CALL ESCAPE	offer escape
219B	21C722		LXI H, STAB	start of look-up table
219E	78		MOV A, B	fetch last letter printed
219F	BE	LOOK	CMP M	and look for it in table
21A0	CAA921		JZ FOUND	
21A3	23		INX H	each table entry
21A4	23		INX H	is three bytes
21A5	23		INX H	
21A6	C39F21		JMP LOOK	keep looking, you'll find it
21A9	23	FOUND	INX H	when you find it,
21AA	5E		MOV E, M	get address from table
21AB	23		INX H	
21AC	56		MOV D, M	
21AD	EB		XCHG	and put it in HL
21AE	CDC821		CALL SECOND	offer new alphabet
2181	CDBA21		CALL OOP	print the chosen letter
21B4	C39821		JMP TALE	and loop back to do it again
21B7	CD7E20	FIRST	CALL SMENU	offer alphabet
21BA	2A840C	OOP	LHLD CI	fetch chosen item's address
21BD	7E		MOV A, M	
21BE	CD240C		CALL WH1	and print it
21C1	47		MOV B, A	save it for look-up;later
21C2	3E20		MVI A, ','	
21C4	CD240C		CALL WH1	and print a space
21C7	C9		RET	
21C8	CD7E20	SECOND	CALL SMENU	offer alphabet
21CB	2A0E0C	SECONDS	LHLD POS	get ready to backspace
21CE	2B		DCX H	and
21CF	220E0C		SHLD POS	do it
21D2	C9		RET	

SUBROUTINE: PUNCT

The subroutine PUNCT handles offering the control codes (by calling another subroutine) and it offers the punctuation symbols. It uses one of the spelling subroutines to handle punctuation.

Addr	Hex	Label	Instruction	Comment
21D3	216B22	PUNCT	LXI H, CONTROLM-1	
21D6	CDFF20		CALL MESSAGE	offer CONTROL?
21D9	CAEB21		JZ CONTROL	
21DC	21AC22		LXI H, PSTART	starting address of punctuation
21DF	CDCB21		CALL SECOND	offer them
21E2	CDBA21		CALL OOP	print the chosen one
21E5	C34C20		JMP ELOP	go back to offer SPELLING?
21E8	000000		NOP NOP NOP	I took out something here

SUBROUTINE: CONTROL

CONTROL offers and executes the control commands.

Addr	Hex	Label	Instruction	Comment
21EB	217322	CONTROL	LXI H, BACKSPACE?-1	
21EE	CDFF20		CALL MESSAGE	offer backspace
21F1	C2FA21		JNZ TWO	
21F4	CDCB21		CALL SECONDS	backspace
21F7	C30820		JMP ENTERL	back to offer SPELLING?
21FA	217D22	TWO	LXI H, ERASE LAST WORD?-1	
21FD	CDFF20		CALL MESSAGE	
2200	C21222		JNZ THREE	
2203	2A0E0C		LHLD POS	
2206	2D		DCR L	back up
2207	2D	MORE	DCR L	back up
2208	3EA0		MVI A, ' '	
220A	BE		CMP M	have we reached a space?
220B	C24022		JNZ RUB	
220E	23		INX H	leave the space
220F	C30820		JMP ENTERL	and go offer SPELLING?
2212	218D22	THREE	LXI H, SPACE-1	
2215	CDFF20		CALL MESSAGE	
2218	C22322		JNZ FOUR	
221B	3E20		MVI A, ' '	
221D	CD240C	END	CALL WH1	
2220	C34C20		JMP ELOP	back to offer SPELLING?
2223	219422	FOUR	LXI H, NEXT LINE?-1	
2226	CDFF20		CALL MESSAGE	
2229	C23122		JNZ FIVE	
222C	3E0D		MVI A, 0DH	
222E	C31D22		JMP END	
2231	219E22	FIVE	LXI H, ERASE SCREEN?-1	
2234	CDFF20		CALL MESSAGE	
2237	CA0020		JZ START	start all over
223A	CD8221		CALL ESCAPE	
223D	C3EB21		JMP CONTROL	
2240	36A0	RUB	MVI M, A0H	put blank on screen
2242	C30722		JMP MORE	
2245	00		NOP	
2246	00		NOP	
2247	00		NOP	
2248	00		NOP	
2249	2181F8	PATCH	LXI H, F881H	initialize text address
224C	220E0C		SHLD POS	
224F	C9		RET	

In the listing below I haven't typed the hex equivalents for the ASCII (this listing was hand-assembled).

Addr		Label
2257		ESCAPE?
2260		SPELLING?
226C		PUNCTUATION?
2274		CONTROL?
227E		BACKSPACE?
228E		ERASE LAST WORD?
2295		SPACE?
229F		NEXT LINE?
22AC		ERASE SCREEN?
		'.?;:!0123456789#$"%&'()*+-
22C6	FB	DB FBH end flag

ALPHABET LOOK-UP TABLE

Here is the look-up table for the various alphabets, in non-standard form.

```
22C7   C13023      A 2330H
22CA   C24B23      B 234BH
22CD   C36323      C 2363H
22D0   C47723      D 2377H
22D3   C59123      E 2391H
22D6   C6AC23      F 23ACH
22D9   C7C223      G 23C2H
22DC   C8DA23      H 23DAH
22DF   C9F123      I 23F1H
22E2   CA0C24      J 240CH
22E5   CB1324      K 2413H
22E8   CC2B24      L 242BH
22EB   CD4624      M 2446H
22EE   CE6024      N 2460H
22F1   CF7B24      O 247BH
22F4   D09624      P 2496H
22F7   D1AD24      Q 24ADH
22FA   D2B124      R 24B1H
22FD   D3CC24      S 24CCH
2300   D4E624      T 24E6H
2303   D50025      U 2500H
2306   D61A25      V 251AH
2309   D72825      W 2528H
230C   D83F25      X 253FH
230F   D95725      Y 2557H
2312   DA7025      Z 2570H
```

THE ALPHABETS

And here are the alphabets, once again without the hex.

```
2315   ASTART   TAOSWIHCBFPMRELNDUGYJVQKZX
232F            FB                              end of alphabet flag
2330            NTSRLDCIGVMYPBKUFWOJXHZEQA
234A            FB
234B            EAOUYRISLJTVMBDWCGHNPFK
2362            FB
2363            OEHATKILURCYSONDZMW
2376            FB
2377            EIUARSOLMDGYNVJQWHEFTPKBZ
2390            FB
2391            RSNDALMCETVFPXIGYOWUHQKBJZ
23AB            FB
23AC            ORIFEAULTSYWBMGCHNJPD
23C1            FB
23C2            EHROAIGSLUTNYMFDBWZJKPC
23D9            FB
23DA            EIAOTURYLNWDSMBHQFPCGK
23F0            FB
23F1            NSTOCMLAREDVGPFBKXUZQIJLWY
```

```
240B   FB                   DB FBH
240C                        AEOUIJ
2412   FB                   DB FBH
2413                        EISANLYOGFWTURDPMKBJCHV
242A   FB                   DB FBH
242B                        EIALYODTSUFRMVWKPCBGNHJZXQ
2445   FB                   DB FBH
2446                        EAOIPMUYSBLFNTHCDRWGJKVCZ
245F   FB                   DB FBH
2460                        DTEGSCIAOYNLFVUKMJRQPHWXBZ
247A   FB                   DB FBH
247B                        NFRUMPLTOWSDCVIBEYAKHJGXZQ
2495   FB                   DB FBH
2496                        ROAELTSPIHMUYWFGKBNDCJ
24AC   FB                   DB FBH
24AD                        UIO
24B0   FB                   DB FBH
24B1                        EIOATSYDMNURCLVKGPWBFHXQJZ
24CB   FB                   DB FBH
24CC                        TEIOSHUCAPYKMWNLGQFBDRVJZ
24E5   FB                   DB FBH
24E6                        HEIOARSTUYLWCFMNBPDZGKVJQ
24FF   FB                   DB FBH
2500                        TSNRLCGPAEMDIFBOYZXUVKQJH
2519   FB                   DB FBH
251A                        EIAOYUSRVZKGM
2527   FB                   DB FBH
2528                        EAHIONRSLTDYKUPFBCMZWG
253E   FB                   DB FBH
253F                        EPTICAHUYOQLNWFSVGBKMRD
2556   FB                   DB FBH
2557                        EOSAITPMBLNWCRGDZHUFVXIK
256F   FB                   DB FBH
2570                        EAZOYIUKTVWHJB
257E   FB                   DB FBH          end of alphabet
257F   C1C1424C45C1424F   AAbleAbout        beginning of vocabulary storage
       5554
```

TEXT AND EDITOR

```
3F00   CD200C   TEXT      CALL WHO        keyboard input
3F03   FE7F               CPI 7FH         is it rubout?
3F05   CA263F             JZ RUB
3F08   FE5F               CPI 5FH         is it shift 0?
3F0A   CA383F             JZ EDITOR
3F0D   FE1C     CTL       CPI 1CH         is it a control character?
3F0F   DA213F             JC CONTROL
3F12   FE20               CPI 20H         is it a control code?
3F14   D2193F   PRINT     JNC PRINT       if not, print it
3F17   C6DF               ADI DFH
3F19   77                 MOV M, A        store it in memory
```

Addr	Hex	Label	Instruction	Comment
3F1A	CDE53F		CALL LPRINT	put it on TV
3F1D	23		INX H	next memory location
3F1E	C3003F		JMP TEXT	do it all over again
3F21	F6C0	CONTROL	ORI C0H	make eighth bit high
3F23	C3193F		JMP PRINT	for 'capital' letters
3F26	CDE53F	RUB	CALL LPRINT	rubout on TV
3F29	2B		DCX H	back up in memory
3F2A	C3003F		JMP TEXT	go do it over
3F2D	2A800C	RETEXT	LHLD 0C80H	fetch starting address
3F30	3E0C		MVI A,0CH	erase TV
3F32	CD240C		CALL WH1	
3F35	C3003F		JMP TEXT	
3F38	22800C	EDITOR	SHLD 0C80H	save start of current line
3F3B	3E0D		MVI A,0DH	start a new line
3F3D	CD240C		CALL WH1	
3F40	2A800C		LHLD 0C80H	fetch start of current line
3F43	EB		XCHG	
3F44	CDD103		CALL DEOUT	print address in hex
3F47	EB		XCHG	restore address
3F48	3E20		MVI A,''	print space
3F4A	CD240C	LOOP	CALL WH1	
3F4D	7E		MOV A,M	fetch character from memory
3F4E	CDE63F		CALL LPRINT	put it on TV
3F51	7E		MOV A,M	
3F52	23		INX H	
3F53	FEFB		CPI FBH	was it the end of a line?
3F55	DA4D3F		JC LOOP	if not, keep printing
3F58	CD200C	KEY	CALL WHO	wait until a key is pressed
3F5B	FE20		CPI ''	is it a space?
3F5D	C2673F		JNZ M1	if not, keep checking
3F60	2A800C		LHLD 0C80H	fetch starting address
3F63	23		INX H	space skips one character
3F64	C3383F		JMP EDITOR	and reprints line
3F67	FE7F	M1	CPI 7FH	is it rubout?
3F69	C2873F		JNZ M2	
3F6C	2A800C		LHLD 0C80H	fetch starting address
3F6F	E5		PUSH H	copy HL
3F70	D1		POP D	into DE
3F71	3EFF		MVI A,FFH	end of vocabulary flag
3F73	010000		LXI B,0	start counting at zero
3F76	2B	M3	DCX H	
3F77	23		INX H	
3F78	0B		DCX B	count one byte
3F79	BE		CMP M	check for end flag
3F7A	C2773F		JNZ M3	keep counting if not the end
3F7D	2A800C		LHLD 0C80H	fetch starting address
3F80	23		INX H	we are moving one space
3F81	CD0001		CALL MOVE	
3F84	C3383F		JMP EDITOR	display edited line
3F87	FE0D	M2	CPI 0DH	is it carriage return?
3F89	CA383F		JZ EDITOR	then display next line
3F8C	FE0A		CPI 0AH	is it line feed?
3F8E	C2A03F		JNZ M4	
3F91	2A800C		LHLD 0C80H	fetch starting address
3F94	2B		DCX H	back up
3F95	2B		DCX H	keep backing up
3F96	7E		MOV A, M	
3F97	FEFB		CPI FBH	look for control flag
3F99	DA953F	M5	JC M5	and keep backing up until found
3F9C	23		INX H	skip the flag
3F9D	C3383F		JMP EDITOR	and display previous line
3FA0	FE5F	M4	CPI 5FH	is it shift 0?
3FA2	CA2D3F		JZ RETEXT	if so, go to TEXT
3FA5	FE1C		CPI 1CH	is it a control character?
3FA7	DABE3F		JC M6	if so, it is upper case
3FAA	FE20		CPI 20H	could it be a control code?
3FAC	D2C03F		JNC INSERT	if not, insert it as is
3FAF	FE1E	M7	CPI 1EH	be sure it is not M or L
3FB1	D2B93F		JNC M7	if its not, then control code ok
3FB4	C6B0		ADI B0H	make into L or M
3FB6	C3C03F		JMP INSERT	and insert it
3FB9	C6DF	M6	ADI DFH	make into control code
3FBB	C3C03F		JMP INSERT	
3FBE	F6C0	INSERT	ORI C0H	make into capital letter
3FC0	2A800C		LHLD 0C80H	fetch starting address
3FC3	F5		PUSH PSW	save character on stack
3FC4	3EFF		MVI A, FFH	end of vocabulary flag
3FC6	010000		LXI B, 0	start counting at zero
3FC9	2B		DCX H	
3FCA	23	M8	INX H	
3FCB	0B		DCX B	
3FCC	BE		CMP M	
3FCD	C2CA3F		JNZ M8	
3FD0	54	M9	MOV D, H	move forward
3FD1	5D		MOV E, L	
3FD2	13		INX D	
3FD3	7E		MOV A, M	
3FD4	12		STAX D	
3FD5	1B		DCX D	
3FD6	2B		DCX H	
3FD7	0C		INR C	count one space
3FD8	C2D03F		JNZ M9	
3FDB	04		INR B	
3FDC	C2D03F		JNZ M9	
3FDF	F1		POPPSW	get back character
3FE0	12		STAX D	and insert it
3FE1	23		INX H	
3FE2	23		INX H	
3FE3	C3383F		JMP EDITOR	
3FE6	FE60	LPRINT	CPI 60H	is it upper case?
3FE8	D2240C		JNC WH1	print as is
3FEB	FE41		CPI 41H	is it lower case?
3FED	DA240C		JC WH1	if not, print as is
3FF0	C620		ADI 20	make it lower case
3FF2	C3240C		JMP WH1	and print it

SPEECH RECOGNITION AS AN AID TO THE HANDICAPPED

by

Horace Enea and John Reykjalin
Heuristics, Inc.
900 North San Antonio Road
Los Altos, CA 94022

ABSTRACT

Speech recognition permits control of devices as well as entry
of data to computers. Using the control aspect of speech
allows an otherwise immobile person to control a wheelchair or
turn lights on and off across the room even though restricted
to a bed or iron lung.

A pilot project in speech control is described which uses a
model car linked to the computer by radio. Computer programs
are included.

MICROPROCESSORS IN AIDS FOR THE BLIND

Robert S. Jaquiss, Jr.
P.O. Box 500
Beaverton, Oregon 97077
(503) 644-0161 ext. 5617

Introduction

There are increasing opportunities for the blind to function effectively in society because of the increased use of computers for data manipulation and information retrieval. This is possible because blind persons can receive the same information as their sighted co-workers.

The rapidly increasing number of small business computers enable blind persons to work as programmers and clerks. Also, as research assistants, the blind can use data bases such as NTIS (National Technical Information Service), IEEE and Lockheed to research articles on topics of interest.

The major difficulty for a blind person is determining the computer's response. This difficulty can be overcome by the use of equipment such as braille terminals, Optacon, closed-circuit television magnifiers, and speech output devices.

This paper describes some of the devices that enable the blind to read computer outputs and explores the future role of microprocessors in this field.

Braille Printing Terminals

Various devices have been designed to produce braille computer output. Although a number of equipment prototypes have been built and demonstrated, the only commercially available braille computer terminals are sold by Triformation Systems, Inc. of Stewart, Florida. Triformation makes two basic types of devices: one produces braille on a paper tape and the other on fanfold paper. (See figure A-1.)

An impact printer, such as a teletype or lineprinter, can be modified to produce braille. A pad (thin rubber or elastic) placed over the platen, or in front of the hammers on a line printer, allows impressions of the period to simulate braille. The braille produced in this manner is rather poor, because the dots are not spaced correctly and because the printing mechanisms are not strong enough to emboss heavy paper.

A modification of this type is available for the IBM 1403 printer. While rather expensive, this printer does produce good quality braille.

The Optacon

The Optacon allows the blind person to read ordinary print directly from the screen of a bright crt or from a hard copy (see figures A-2 and A-3). In operation, the Optacon picks up the image to be read with a camera and focuses it onto an array of photodiodes. The user feels the ends of vibrating wires, one for each photodiode, to feel the shape of the characters. Uppercase type is the easiest to read, followed by lowercase type. The plainer the type, the more easily it can be read. While the Optacon is not as fast to use as braille, it is more versatile.

Closed-Circuit Television Magnifiers

For use by partially-sighted individuals, the closed-circuit television magnifier is basically a camera attached to a high resolution black and white television monitor. The camera is equipped with a special 20X magnification lens. These machines also incorporate a reverse image capability. This provides white letters on a black background which, in some cases, is easier to read.

Unless the closed-circuit television magnifier is equipped with a viewing table, reading of hard copy may be a problem, and the copy is not portable. A plotter can be used to make a portable copy with large letters. However, this is a slow job, and the paper must be manually placed on the machine.

Speech Output Devices

Speech output is available in various forms. Votrax types require a string of phonemes for speech generation, while the Compu-Talker type requires more complicated software. In certain applications, such as obtaining listings of long programs, this type of output is not acceptable because it would require the user to memorize entire programs in order to make corrections.

TSI (Telesensory Systems, Inc.) makes a type of speech board which has a canned vocabulary on it. The canned words will be generated when the board is sent a number corresponding to the desired word.

New Research

There is some work being done with tactile displays, the blind person's version of a crt. To my knowledge, there are no commercially-available devices of this type. Some proto-types have been built that use air to drive pins up to form a braille line. Others use the input port on an Optacon so a computer can generate uniform characters on the Optacon display. This last method solves one of the major problems of the Optacon, which is the necessity of holding the camera to the screen with one hand, typing with the other, and reading from a braille coding sheet.

Microprocessors in Aids for the Blind

Microprocessors will be invaluable in aids for the blind because they can perform data formatting and searching at a reasonable cost. Braille books are very large. For example, the 1959 edition of the World Book Encyclopedia requires 43 feet of shelf space. The Art of Computer Programming, Volume 1, by Knuth if almost two feet long. (See figure A-4.) Because of the large size and high cost of braille books, attempts are being made to store books onto data cassettes which can then be read by a computer and displayed on some sort of refresh device. With this approach, it will be possible for the blind to have books available, at a reasonable price, which can be rapidly read and/or scanned through using an editor.

Microprocessors can also be used to interface display devices to instruments. This will be much easier in the future, because of the GPIB interface bus that is being used more and more in instrument design.

Summary

Equipment now on the market enables blind persons to determine a computer's response and thereby receive the same information as their sighted co-workers.

The cost and size of braille books can be greatly reduced by storing them on data cassettes. Microprocessors will play an important role in the devices that enable the blind to read the data stored on the cassettes.
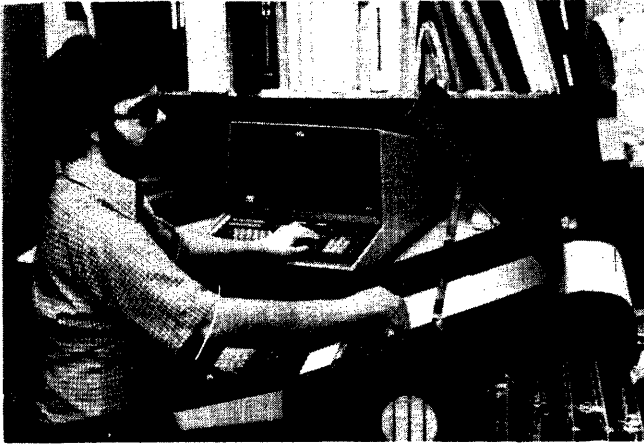
Figure A-1

The author is reading the output of a braille printing terminal with his right hand.

Figure A-2

An Optacon is being used to read from the screen of a Tektronix display terminal.
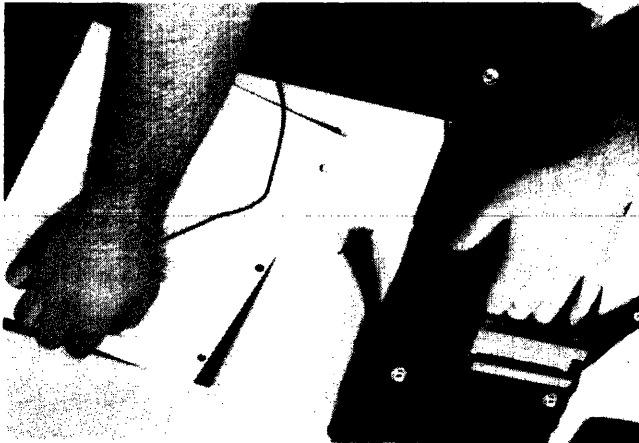
Figure A-3

An Optacon is being used to read a hardcopy printout.

Figure A-4

The braille edition of *The Art of Computer Programming*, Vol. 1 by Knuth.

BLIND MOBILITY STUDIES WITH A MICROCOMPUTER

Carter C. Collins, William R. O'Connor
and Albert B. Alden

Smith-Kettlewell Institute of Visual Sciences
and Department of Visual Sciences
University of the Pacific, 2232 Webster Street
San Francisco, California

ABSTRACT

In the evaluation of a new sensory aid for blind mobility comprising a wearable tactile vision substitution system we have made a number of behavioral measures. The initial batch of data was collected and reduced by hand which pointed up the necessity for automating the procedure in order to process the volume of data anticipated in a large scale evaluation program. For this we have designed and built an 8080 microcomputer based blind mobility evaluation system, utilizing an ultrasonic triangulation ranging method, which today contintinuously tracks and plots the real time course followed by a blind person in our 20 x 30 foot mobility laboratory space. The detailed path of the blind walker is recorded in graphical form on a monitor screen overlayed by the plan of arrangement of obstacles in one of many arbitrarily chosen obstacle courses.

BASIC language software is being developed to compute, store and display some of the more important mobility parameters, including collision avoidance, average walking speed, location and duration of stops, total travel time, travel efficiency and safety. As an example of the kind of output this system will be required to produce we relate the findings of our initial study.

We have examined the effects of some 40 consecutive practice trials on the safety and efficiency of indoor mobility performance of each of two blind subjects in a laboratory travel environment. After two hours mobility experience with only the tactile imaging device, blind subjects walked freely at about one foot per second on a 65 foot mobility course through a room cluttered with furniture, detecting and avoiding over 95% of the obstacles (100% in half the trials). Subjects decreased stop and search time from 61 to 14 seconds with a 120 second mean total travel time. Travel efficiency (percent time spent walking) increased from 63% to 86% during these trials.

These tests demonstrate the feasibility of the concept that optical information alone presented on the skin can contain sufficient information to permit the blind to avoid obstacles and steer a clear path for successful indoor mobility.

Introduction

Vision is probably our most important mobility sense, permitting us to walk rapidly, accurately and confidently wherever we wish to go. Since this source of mobility information is not available to the blind, there has long been the need for an effective sensory aid permitting safe and efficient travel by the blind pedestrian. The latest electronic guidance devices (1,2) have not yet been generally accepted by the blind community (3), and today the long cane remains their best available mobility aid (4).

It has been the specific aim of this present preliminary investigation to determine the feasibility of utilizing wide field optical information impressed onto the skin as a mobility aid for the blind. In this study we have set up a synthetic mobility environment and have made a number of objective measurements of the mobility performance of blind subjects using a newly developed wide field of view sensory aid as their only guidance device.

This preliminary study alone required over 150 man hours of data manipulation. In order to expedite processing of the voluminous data expected in a full scale evaluation program it has become necessary to devise an automated data collection and reduction system. This system was designed to follow a person picking his way through our laboratory mobility testing area. We required sufficient range to cover the 20 foot square mobility course in our 20 x 30 foot laboratory. Range resolution of less than 2 inches was required to detect collisions with obstacles, and this can clearly delineate individual footsteps. The sampling rate must be fast enough to preserve the continuity of the information, even with rapid motion, without in any way encumbering the motion of the pedestrian being tracked. In addition, the system had to store the X-Y coordinates of each sample of the pedestrian's track for statistical analysis of a trip through the room, and provide sufficient extra machine capability for storing the outlines and positions of furniture and other obstacles in the mobility course.

For training purposes, we wanted to provide the possibility of performance feedback to the pedestrian, in something approaching real time, by the operator of the system; i.e., a meaningful way to permit the pedestrian to compare one passage taken through the room with another, based on information collected by the system, and made available shortly after completing a passage.

The system which we have designed comprises:
- An ultrasonic locator system consisting of an RF-ultrasonic transponder to mark the position of the blind pedestrian by triangulation.
- A triangulation (R_1, R_2) coordinate to Cartesian (X,Y) coordinate conversion program with an output graphics display map of the pedestrian's path.
- A mobility performance evaluation program resulting in archival mass storage of path $(X,Y$ and time) coordinate and performance evaluation data.

Ultrasonic Locator System. The blind pedestrian locator and tracker consists of a "wireless tether" similar to the apparatus used by Strelow, Brabyn and Clark (11) to follow the progress of blind pedestrians in their mobility laboratory at Canterbury University in New Zealand. However, our system eliminates the three long strings which they attached to the subject's head in order to compute his location in the laboratory. Such strings would become entangled in the overhangs, columns and other tall obstacles encountered in our mobility laboratory. We have replaced the strings with two invisible RF-ultrasonic links, leaving the pedestrian completely free, with no strings attached. This ultrasonic triangulation technique is diagrammed in Figure 1.

The components of this system are:
1) a transponder (transceiver) worn by the blind pedestrian made up of an RF receiver which pulses an ultrasonic transmitter (small 40kHz loudspeaker). This RF receiver picks up the signal from
2) an RF transmitter triggered by
3) control electronics which are controlled by
4) the microcomputer and
5) two microphones which pick up the ultrasound pulses generated by the ultrasonic transmitter.

These microphones feed their amplified signals back through the control electronics to the microcomputer which generates the X-Y coordinates of the pedestrian from the ultrasonic time delays as described later. The components of this triangulation-tracking system are shown in Figure 2A, and the system is being worn (by O'Connor) in Figure 2B.

The operation of the ultrasonic system is as follows. The microcomputer (Figure 1) sends a start pulse to the control electronics. This sets two flip-flops, and applies a 4kHz

audio modulation signal for about 15 ms. to a low power 28MHz R.F. transmitter. The outputs of the two flip-flops start two counters counting the output of a 10kHz clock signal generated by the microprocessor. The 4kHz modulation of the broadcast signal is detected by a tuned filter in the R.F. receiver carried by the subject. This detected signal is used to turn on the ultrasonic (40kHz) transmitter worn by the subject. The ultrasonic receivers each receive the transmitted 40kHz ultrasound delayed by a time proportional to the distance from the subject to the receiver. The first received 40kHz signal from each receiver exceeding a set threshold is used to reset the flip-flop associated with that receiver. This resetting stops the counter whose count is proportional to the measured distance. (For a 10kHz clock, one count represents a resolution of 1.32 inches.) The microprocessor reads the counters, resets them and initiates a new cycle with a start pulse.

X-Y Coordinate Computation. The ultrasonic triangulation data must be converted to X-Y coordinates for plotting and convenient data reduction. The triangulation data R_1 and R_2 are converted to X-Y Cartesian coordinates in the following manner.

To help speed the calculations some algebraic manipulation was done to allow one of the two equations to be solved without the use of square roots.

Referring to Figure 1:

$X = L - X'$ where L is the length of the testing area, in this case 20 feet.

(a) $\quad X^2 + Y^2 = R_1{}^2$

(b) $\quad Y^2 = R_1{}^2 - X^2$

(c) $\quad (D-Y)^2 + X^2 = R_2{}^2$ where D = the separation between the microphones.

expanding (c):

$$D^2 - 2DY + Y^2 + X^2 = R_2{}^2$$

substituting (a):

$$D^2 - 2DY + R_1{}^2 - X^2 + X^2 = R_2{}^2$$

$$-2DY = R_2{}^2 - D^2 - R_1{}^2$$

(1) $\quad Y = \dfrac{R_1{}^2 + D^2 - R_2{}^2}{2D}$

from (a)

(2) $\quad X = \sqrt{R_1{}^2 - Y^2}$

Even more simplification and speed-up of the software was possible due to the fact that the combination of resolution (an inch or two) and range (from 10 feet out to the 36 foot diagonal of the 20 by 30 foot room) allowed the use of a single computer word of precision in the input, provided that the data was normalized for this.

In operation the system runs at the afore-mentioned tenth second repetition rate and its resolution is fixed by the clock applied to the counters. This determines the minimal change in delay, between the strobe and the returned pulse detectable in either channel. In one of the clock periods chosen (C 100 ms) sound will travel 1.32", which in turn is the unit of measure used with the 256 x 256 graphics output for a useful length in either direction of 28.16 feet vs. the mobility course dimension of 20 feet on a side.

The X-Y coordinate computing application program is written in 8080 machine code and runs on an IMSAI (Figure 3) equipped with the following hardware:

- Extensys 64K dynamic RAM card with appropriate disables.
- IBEX 16K PROM board with the (half used) Processor Tech. ALS-8 firmware used in development.
- Chromemco BYTESAVER with the application and I/O programs in EPROM.
- IMSAI PIC-8 with the counter-mux. hardware built in.
- Processor Tech. CUTS board for "CUTER" based mass storage on cassette.
- Processor Tech. 3P&S board for I/O.
- Matrox ALT 256**2 graphics board for plotting the output.
- Matrox ALT 2480 board running the system output.

Everything is tied together as illustrated in Figure 3.

The X-Y coordinate computing program is written in 8080 machine code and is indepen-dent of any higher level language or operating system. During operation the program behaves as follows. An initialization routine sets up the stack pointer, a jump table, and the address of both the list and a buffer where sampled data is kept temporarily. It then programs the priority interrupt/programmable counter board to interrupt the system at tenth of a second intervals, and halts.

The interrupt service routine controls the program; when called it inputs data from the previous cycle, issues a reset to the hardware and outputs the R.F. strobe.

The time during which the hardware is ac-quiring new data, is used in this manner to process the data that is now loaded into RAM. In processing, the data is first checked for errors and then normalized. The "Y" coordinate is then found using eq. 1, that result is stored and used in the finding of the "X" coordinate with eq. 2. The pair of coordinates is then output on the map display, and stored

on a list which resides in RAM, filling upwards from Ø1ØØ H. Finally the lists' pointer is incremented, and things halt until the next interrupt.

Control of the ultrasonic system is imple-mented through one output and one input port on the Chromenco 3P+S (I/O) board. The output port uses three bits to operate a custom built circuit mounted on the Priority Interrupt/ Counter board (PIC-8) containing two eight-bit counters, a multiplexer to switch between them, and buffers. Bit zero starts the counters at strobe time (they are stopped from counting by the detected ultrasonic pulse). Bit one selects one of the two counters, and bit two resets them after they have been read. Input is via a single eight bit wide input port.

System I/O is through a Wyle Computer Prod-ucts CRT and keyboard terminal chosen for economy. Input from it required a custom soft-ware package and, an input port and a status bit on the 3P+S board.

Output to the CRT comes from a MATROX 24x80 memory mapped video board (which occupies 4K of space because it uses twelve address bits to access characters by row and column) chosen for compatibility with printer formats. It also needed a custom softward package.

Pending the arrival of our disk, development tools were Processor Tech's. ALS-8 firmware module (8K) for assembler, editor, and debugger as well as their tape based CUTER system and board for mass storage.

The applications package has its own special output (Figs. 2 and 3) on another CRT graphics monitor; this is driven by a MATROX 256*256 graphics board, which looks to the system like four I/O ports. They include; "X" address, "Y" address (both 0-255), intensity (on-off, but expandable with decoding and multiple boards for colors and grey scale) and screen erase. A sample output of a pedestrian's track on the graphics monitor is illustrated in Fig. 4B.

There appear to be provisions for synching the rasters of the two MATROX boards and sum-ming their video to get graphics and alpha-numerics in one combination display.

Taking its input from the X,Y coordinate computing program's output, the mobility performance evaluation program in BASIC lan-guage should prove very flexible. Any changes or additional relationships to be calculated can be simply added at any time. The mobility performance parameters which we have found most important to be evaluated at this time include:

- . X,Y coordinates of blind pedestrians' position
- . instantaneous velocity and acceleration of the subject
- . average walking velocity over entire course
- . location and duration of stops
- . length of individual paths between stops
- . total distance traveled by subject

- number and direction of turns taken by subject
- number of collisions with obstacles
- number of obstacles approached by subject
- collision avoidance, i.e., percent of encountered obstacles with which subject avoids colliding
- total travel time
- travel efficiency
- the productive walking index or percent of total time spent actually walking
- other factors, as are deemed important, can be programmed for computation in simple BASIC language.

Examples of such reduced data are taken from our original batch of manually processed information.

Sensory Aid. The wearable sensory aid utilized in these experiments (Figs. 5 and 6) has been developed over the past ten years (5,6,7). It utilizes a miniature, monolithic, wide angle television camera mounted on the frame of a pair of glasses which serves as the artificial eye of the sensory aid. Images from this camera (Fig.) are electronically impressed point-for-point onto the skin of the abdomen by means of a 10-inch square array of 1024 coaxial stimulus electrodes mounted on a flexible, elastic supporting garment worn directly against the skin of the abdomen (7). The complete system weighs five pounds including two pounds of rechargeable nickel cadmium batteries for eight hours of operation.

A small and versatile optical system was designed by one of us (CCC) for blind mobility use with the sensory aid. The optics provide an adjustable field of view up to 180°, an infinite depth of field which permits operation with no focusing adjustments required by the subject, and a large aperture (f:0.5 for operation with available room light.

Preliminary trials indicate that a 90-degree field of view appears to be about the best compromise between sufficient resolution with the 32-line system to detect obstacles and the very wide peripheral field of view necessary for mobility. The optical axis of the lens was directed 45° downward to include a field of view from just above the horizon down to the space directly in front of the subject's feet in order to permit him to detect low obstacles within a footstep of his path. This field of view and lens direction were used throughout these tests.

Mobility Course. The experiments performed in this study were carried out in a modular, quickly alterable and objectively definable synthetic indoor mobility environment contained in a 20 by 30 foot room. This mobility course was layed out in a Cartesian coordinate grid system with one foot square vinyl floor tiles (Fig. 4A). The obstacles consisted of real walls, a door frame, pieces of real furniture, high and low tables, chairs, a podium, wastebaskets, boxes and wall curtains, as well as simulated columns and overhanging beams con-

structed of corrugated cardboard for subject safety (Fig. 6).

Sixteen different courses were layed out on graph paper with obstacles located on numbered squares corresponding to those of the room. This facilitated quick relocation of obstacles and permitted a rapid and essentially random temporal sequence of different layouts to be presented to each subject. Illumination was 30 to 50 footcandles at floor level.

The mean total path length of a single mobility course was 65 feet (20 meters). Of the 30 total obstacles, it was expected that only 10 or 20 might be closely encountered by the blind subjects as they made slight variations in their travel paths. The mean number of closely approached obstacles requiring avoidance was 12 per course. The mean free travel path length between obstacles was 5 feet. There were 6 turns per route on average.

The mobility course and evaluation design philosophy has borrowed heavily from the precepts of Armstrong (8), Kay (1), and, in particular, Shingledecker (9), in Emerson Foulke's laboratory.

Subjects. Two blind male subjects, age 28 and 37, were utilized in these experiments. Neither subject possessed any degree of functional vision and each had been blind for over 25 years. Both subjects were excellent cane travelers with years of practice. They each had over 200 hours experience with other, fixed tactile imaging devices.

Procedure. Subjects were initially given about 15 minutes of pretrial mobility experience with the sensory aid. They were given a verbal description of the nature of the courses and were requested to walk at a normal pace avoiding collisions and that the course was designed to keep the "shoreline" left. The experiment consisted of the subjects walking completely through respectively 39 and 48 consecutive trial courses, each one selected from the 16 different courses, such that each course was different from the last. Successive trials were made about every five minutes. Experimental sessions lasted from about 15 minutes to one hr.

The entire series of tests were recorded on videotape and the experimental data were obtained by a number of replays of the tape. Two observers with stopwatches and a counter measured the distance and duration of each short path leg walked by the subject; number and duration of stops; number, location and type of collision, and number and size of head movements.

The detailed path of the subject was recorded in graphical form overlaying the plan of arrangement of obstacles for each course as in Figure 4A. A separate plot resulted for each of the 87 total runs. Data were correlated with these graphical records.

Results. The travel safety of the blind subjects was scored in terms of collision avoidance, that is, the percent of the encountered obstacles with which they avoided collid-

ing. The collision avoidance score showed a mean value of 91.77% for both subjects combined, with an initial value of 84.5% and final value of 95.04%; an increase of 12.5%. For subject B.G. the mean was 89.87%. The linear regression fit of the data for this subject indicates an initial score of 86.75%, increasing to a final score of 92.99% for a 7.19% increase over 39 trials. The mean collision avoidance score for subject L.S. was 93.31% with an initial score of 89.91% and a final score of 96.72%, an increase of 7.56% over 48 trials. This performance is shown in Figure 7 with the linear regression fit of the data.

As suggested by Shingledecker (9), the change in travel time with practice was analyzed in terms of three components of travel efficiency: walking speed, number of stops, and duration of stops along the travel path. The mean total time to negotiate the mobility course was 120.3 seconds for both subjects. Subject B.G. took more time at the outset, 183.4 seconds vs. 108.9 seconds for subject L.S. But with two hours training both subjects took the same final time, about 98 seconds mean to complete the course. During this experiment subject B.G. decreased his total travel time 46% with a mean time of 141.1 seconds (Fig. 8). The faster walker, subject L.S., decreased his total time by 10% with a mean of 103.3 seconds total travel time to complete the mobility course.

Walking speed indoors was fairly stable at 0.8 feet per second mean for both subjects. (We have observed indoor walking speed to be roughly half that of outdoor speed for both blind and sighted persons.) The mean walking speed for subject B.G. was .73fps, showing a 5.6% increase (Fig. 9). The mean walking speed for subject L.S. was .84fps over 48 trials. We could not measure a change in his walking speed.

The mean number of pauses or stops along the travel route to search for a new clear travel path was 2.95 for both subjects combined. They initially made a mean of 5.11 stops decreasing by 80.3% to a mean of 0.92 stops by the end of the experiment. Subject B.G., with a mean number of stops of 3.36, showed a 72.8% decrease from an initial value of 5.5 stops to a final value of 1.2. The mean number of stops for subject L.S. was 2.62, with an initial value of 4.8 and a final value of 0.7, for a decrease of 85.4% during the course of the experiment. The mean number of stops for both subjects combined decreased from 5.11 initially to 0.92 stops at the end of the experiment, an 80.3% decrease.

The mean total stop and search time for both subjects combined was 37.5 seconds, varying from an initial value of 61.3 seconds to a final value of 13.7 seconds; a 78% decrease over the duration of the experiment.

Subject B.G. with a mean total stopped time of 55.2 seconds decreased 88.3%; from 98.8 seconds initially to 11.6 seconds final value (Fig. 10). Subject L.S. exhibited a mean total stopped time of 23.2 seconds, decreasing 49.8% from 30.9 seconds initially to a final value 15.5 seconds.

The mean duration of each individual stop and search period was 12.93 seconds for both subjects combined; initially 12.0 seconds, it actually increased to 14.89 seconds final value (but subjects averaged only one stop at the end of the experiment).

The PWI, or Productive Walking Index, introduced by Armstrong (8), is a measure of the continuity of progress of the subject towards his goal. It is the percent total time spent actually walking. The mean PWI score for subject B.G. was 68.2% with an initial value of 48.7% and a final value of 87.7% for an increase of 80%, an impressive practice effect as shown in Figure 11. The mean PWI score for subject L.S. was 80.60% with an initial value of 74.12% and a final value of 87.12% for an increase of 17.48%. The mean PWI for both subjects was 75.04%. Initially 62.7%, with a final value of 87.3%; mean PWI for both subjects increased 39%.

Discussion. The experimental results indicate that the optical information provided by the new tactile television sensory aid has permitted blind subjects to safely avoid most obstacles and to steer a clear path for successful and increasingly efficient indoor mobility.

Apparently the device immediately provided them anticipatory information about the travel route as evidenced by their initial 85% collision avoidance score. We are encouraged to believe that with considerably more practice subjects could learn to avoid essentially all obstacles, as suggested by their 95% collision avoidance score after only two hours of practice, and 100% in 10 out of the last 12 trials for subject L.S. (Fig. 7).

The blind subjects quickly learned to increase their travel efficiency with the sensory aid as shown by the 39% increase in their Productive Walking Index and 31% decrease in travel time during the same two hours practice. The most significant component contributing to the decreased time to complete the mobility course was the 80% decrease in the number of stops made by the subjects, resulting in a 78% decrease in stop and search time. By the end of the experiment subjects were averaging about one 15 second stop for the entire 65 foot mobility course. The relatively long scanning time taken by subjects at each stop may well be due to their attempts to recognize details of obstacles and escape routes with the low resolution (3°) due to the 90° wide angle of the display (90°/32 lines = 3°).

To meet this problem we now have designed and built a 16 to 1 zoom lens with a field of view from 10° to 160°. This will permit the blind pedestrian to utilize a wide field of view for orientation and navigation, but when he encounters an unrecognized obstacle he will be able to focus his attention by zooming down onto the details of the object in order to better recognize it. Thus, we believe subjects will be able to reduce their stop and search time.

Because of the inordinate amount of time required to collect and process this type of information, a great body of valuable mobility performance and rate of learning data would go uncollected if an automated collection and reduction system were not available. We have designed a small, economical, dedicated micro-computer system to fill this need.

Acknowledgements

References

1. Kay, L:Conf on Eval of Sensory Aids,NAS,1972
2. Nye, P:Prelim Eval of C-4 Laser Cane, NAS, 1973
3. Graystone, P & McLenan: AFB Res Bul 17, 173, 1968
4. Hoover, R:in Blindness, P Zahl, Princeton Press, 1950
5. Collins:Proc Nat Sym on Info Display 8, 290, 1967
6. Collins:IEEE Trans Man-Machine Syst 11, 65, 1970
7. Collins & Madey: Proc San Diego Biomed Sym 13, 1974
8. Armstrong, J: Human Factors in Health Care, Pickett and Triggs (Eds) Lexington Books, Lexington, MA, 1975
9. Shingledecker, C: PhD Thesis, U Louisville KY, 1976
10. Gibson, J: The Senses Considered as Perceptual Systems, Haughton Mifflin, New York, NY, 1966
11. Strelow, ER, J.A.Brabyn and G.R.S. Clark: Behavioral Research Methods and Instrumentation, 1977
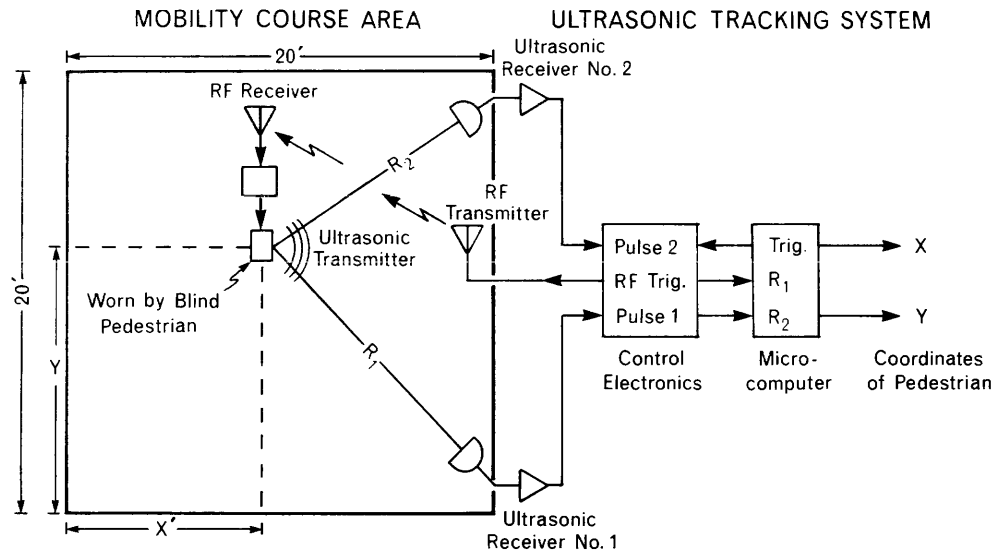
Fig. 1. Schematic layout of the microcomputer
controlled ultrasonic locator system for
generating the X-Y coordinates of a blind
pedestrian from ultrasonic triangulation data.
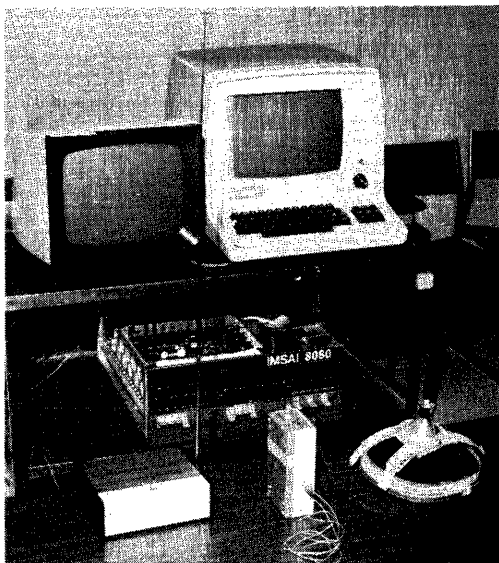
Fig. 2A. Components of the ultrasonic
locator system for blind mobility tracking.
The RF receiver is the small box in the center
foreground with its trailing antenna. This
receiver is generally worn on the belt of the
blind pedestrian. The ultrasonic transmitter
is shown in the right foreground as a small
omnidirectional ultrasonic loudspeaker mounted
on a lightweight headband worn by the pedes-
trian being tracked. The RF transmitter is
in the left foreground and the IMSAI 8080
microcomputer in the lower background. The
control terminal on the right and the output
display graphics monitor is on the left. The
blind pedestrian's path is traced out in detail
on this monitor. The sequential coordinates of
this path are stored in memory and subsequently
dumped onto a cassette recorder (not shown)
when the 64K memory is full.
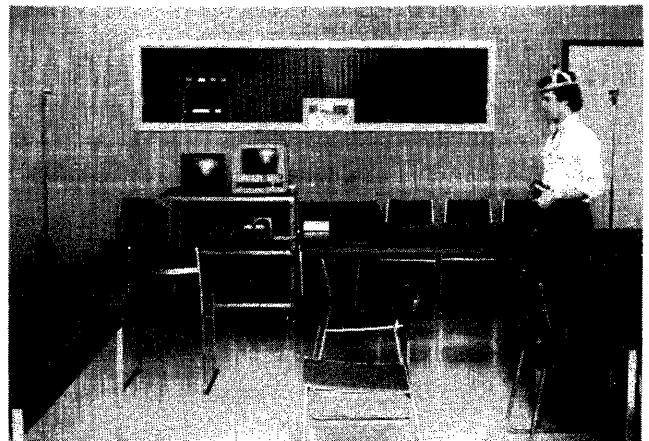
Fig. 2B. The microcomputer controlled ultra-
sonic locator system showing the components in
use. The subject is wearing the lightweight
ultrasonic transmitter (loudspeaker) on his
head and he carries the RF receiver. The two
microphones (on stands at the far edges of the
picture) pick up the ultrasonic pulses from the
subject and deliver them to the ᵤicrocomputer
which continuously computes the X-Y coordinates
of the subject and plots them on the graphics
monitor screen (left background).
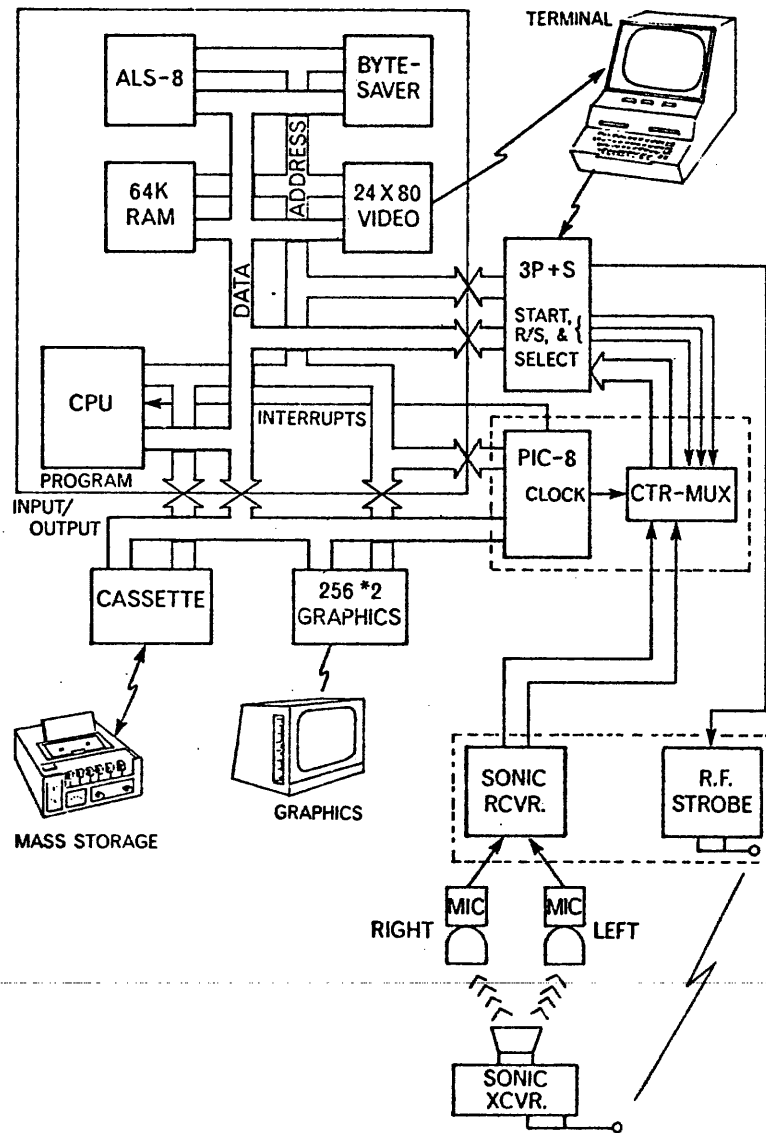
Fig. 3. The hardware complement of the micro-
computer controlled ultrasonic locator system
(described fully in the text in the X-Y
coordinate computation section).
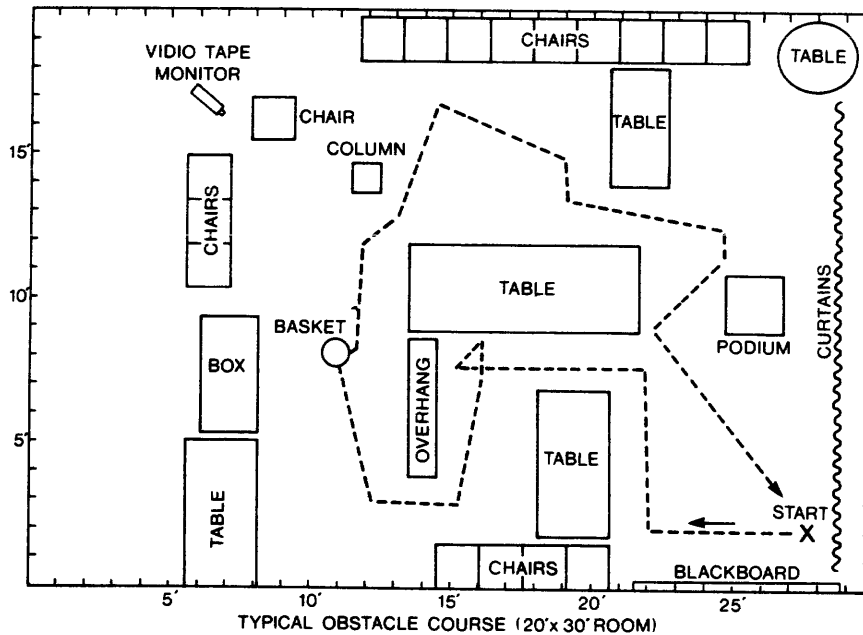
Fig. 4A. (Top) Plan of one of the 16 mobility courses with a typical path walked by a blind subject. Note he avoided all obstacles (except the small basket on the floor) and found a clear path through the maze of obstacles with only the tactile mobility aid.
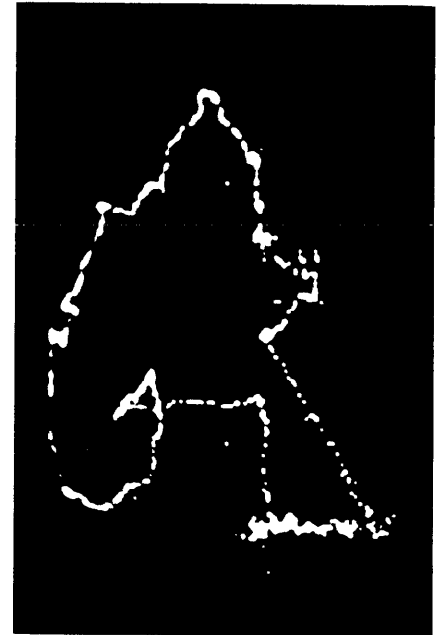
Fig. 4B. (Bottom) CRT tracing of a similar path of subject plotted by ultrasonic tracking device and objective microcomputer evaluation system.
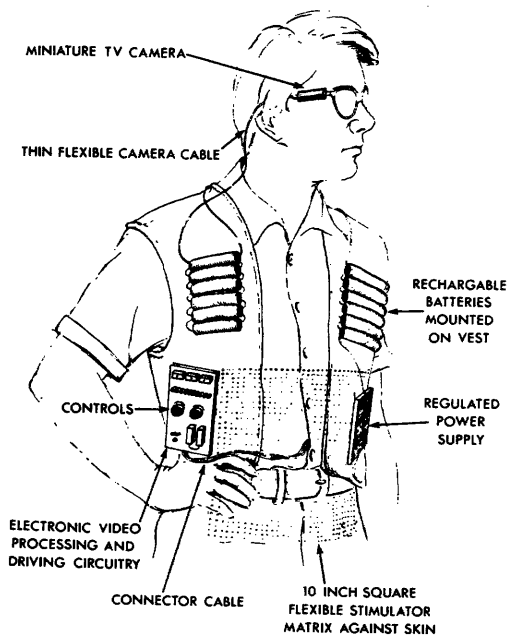
Fig. 5. Artist's conception of the 1024-point fully portable electrotactile mobility aid. The miniature TV camera mounted on a pair of glasses permits the object at which the wearer points his head to be imaged on the skin of his abdomen. The image is converted to a pattern of electronic pulses applied to the skin by an array of small electrodes in a flexible undergarment.

Fig. 6. One of the 16 furniture arrangements of the indoor mobility course used for blind mobility tests. Course required subject to avoid overhang (in background), and thread between chairs and tables. In the background is a cul-de-sac of tables he was required to negotiate.
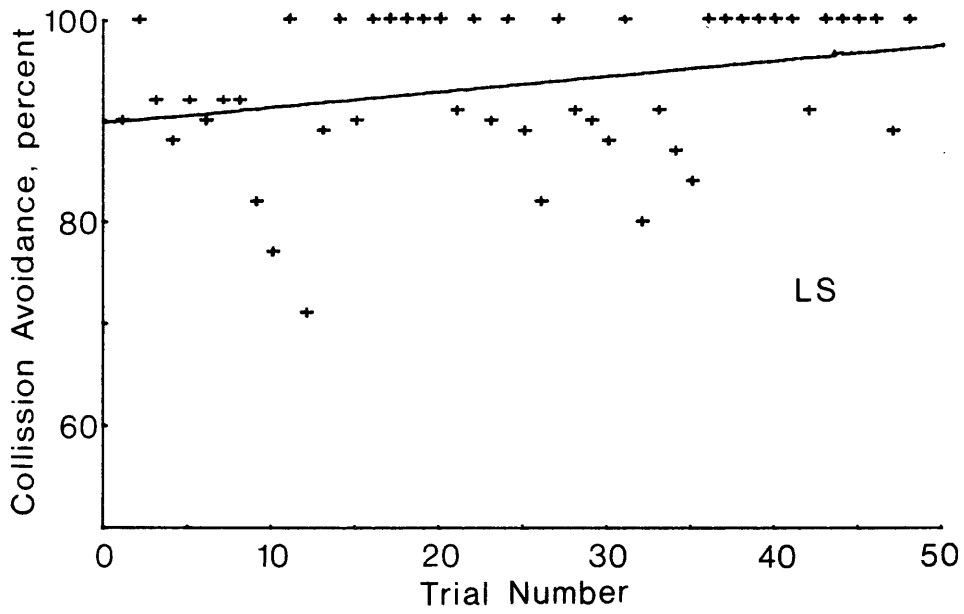
Fig. 7. Collision avoidance mobility data is one measure of blind pedestrian safety with a sensory aid. Here, performance improves over 48 trials.
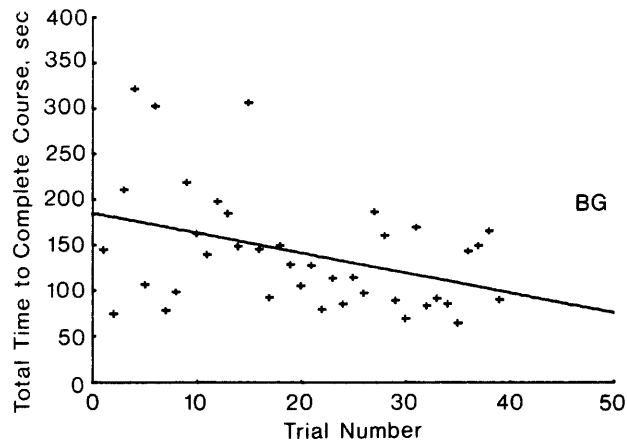
Fig. 8. Total time to complete each course is a measure of efficiency of blind mobility.

Fig. 10. Cumulative time subject stopped to search for new path during each trial.
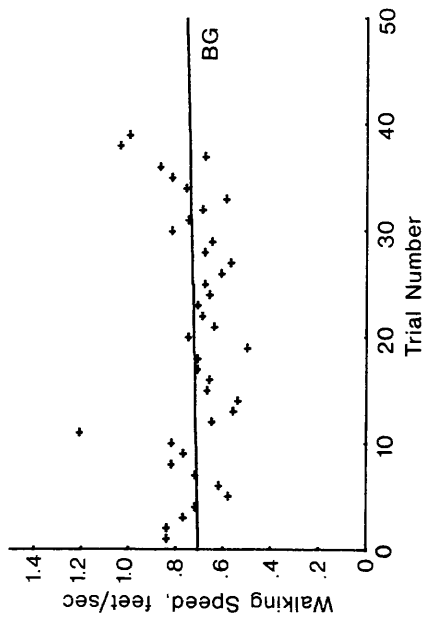
Fig. 9. Walking speed of one subject.
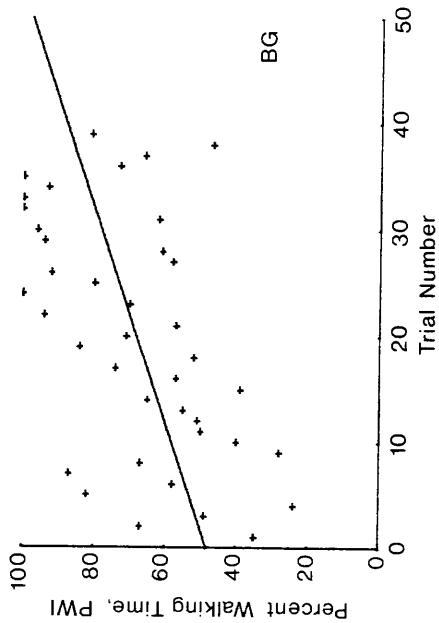
Fig. 11. Productive Walking Index (PWI) is the percent of the total time spent actually walking.

THE DESIGN OF A VOICE OUTPUT ADAPTER FOR COMPUTER

William F. Jolitz, 2124 Parker Street, Apt. 309, Berkeley, CA 94704

ABSTRACT

The design of a Voice Output Adapter for visually handicapped computer programmers is discussed. This device, based on a DEC LSI-11 microcomputer and a VOTRAX VS-6 synthesizer will generate speech from ordinary typed text. Phonetic translation is accomplished by a set of rules, instead of a dictionary. Although this device uses a VOTRAX synthesizer, and experimentation has been limited to English, the device will act independently of language or synthesizer type. All software will reside in the main memory; no peripheral memory will be used. This results in a compact device.

Introduction

Attempts have been made to use speech synthesis to aid visually handicapped people, but most have not encountered great success. Often, these attempts were too costly, too complex, too limited or otherwise impractical. An attempt [1] is currently being made to break this "practicality" barrier by designing an inexpensive device which couples a Hewlett-Packard HP 9825A desk-top calculator (see A-1) to a VOTRAX VS-6 voice synthesizer. This voice display will allow completely unhindered use of the calculator by unsighted operators, and will require no extra training to use. Since the unit is based on a microprocessor, it will be compact and relatively inexpensive -- two features which lend themselves to mass manufacturing. The designed system is not language or hardware dependent. It is possible to have a multilingual design and/or interface to other computers or calculators.

Why Voice? There are different methods that can be used to present computer output to the blind. A method that has been widely employed is using a Braille terminal to translate a line of text into Braille embossings, which is a direct method of approaching the problem, but one with some limitations. Some of the limitations of this Braille method are that it requires copious quantities of consumable paper, prints at a slow rate and requires Braille training.

An alternative method is to use voice output from a speech synthesis unit. This method has not been frequently used, primarily due to the high cost of the hardware. However, with the recent revolutions in the microelectronics area (which made personal computing a reality), this is no longer a problem.

Voice methods seem to complement Braille methods, in that they require no consumables (except, of course, electricity), communicate very rapidly, and require no training. Extending this thought further, Braille methods leave hardcopy, while voice methods are suited for interactive use. The differences are similar to those found when choosing either a CRT or a printer type terminal, and this simile is accurate enough to predict where voice or Braille (or both) can fit into an application.

In data entry applications, hardcopy is a hindrance, so CRT terminals are used. Conversely, CRT terminals are a hindrance when composing programs, since often you need hardcopy to refer to later. Hardcopy has nice properties that are easy to handle, easy to transport, and hardcopy is useful in discussing program/data text among a group. Many computer systems mix both types of devices to take advantage of both. Another advantage of the voice method is that it can be used by sighted users to double check data entry. Feedback of this kind has been found to be very efficient at detecting entry errors in a flight research experiment.[2] In addition, an easy-to-use speech synthesis system (as this basically is) allows for miscellaneous special purpose applications (paging systems, etc.) Voice is a good medium for getting a short message across from a computer to a person.

A Sample Session With The Voice Display System. In order to understand this voice system, let us look at how it will work in use. In the following example (see A-2), the operator will type a one-line program into the calculator which will display "hello" on the calculator's display panel when run. The "dsp" mnemonic is a calculator function to display text on the display panel. As can be seen in the example, the operator presses a key on the calculator, then the key's name is spoken by the synthesizer. This is called echo-feedback, and it's purpose is to allow the user to monitor text

input on a character by character basis. When the operator presses the store key, the calculator accepts the typed line of text as a program step. Upon observing the calculator accepting the program step, the voice system reads it out word by word. The operator can execute the program by pressing the run button, where the entered program then writes "hello" on the display (which the voice system reads out) and stops. To review the program entry, the operator can choose to "fetch" it, whereby the program step is read out as it is displayed.

Basically, the calculator's display is expressed with speech instead of print. It is as if you had a person reading the display for someone who was entering a program but could not see the display.

Prototype Philosophy. The intent of this project is to show that an inexpensive device to aid blind computer programmers can be made. No attempt is being made to engineer a product; all that we intend to do is demonstrate the ideas for such a product with a working model. All equipment is stock, with the lion's share borrowed from various parts of NASA Ames Research Center, Digital Equipment Corporation and the Sensory Aids Foundation. All software is generated in the high level language, C, [3] for ease of programming (in systems programming applications, C is the APL of computer languages). Any additional reduction in program size, through using assembly language, would mean a larger cost in programming time and frustration (anyway the C compiler used generates size optimized code).

Hardware

Practicality is a main concern in the design of this system. Hardware was chosen with availability in mind. Although no attempt was made to compact hardware, it was felt that overly large hardware would obscure the basic concept of practicality. Hardware was chosen whenever possible to reduce size. As a result, all of the hardware associated with this project (less printer) is about the size of a breadbox. If one were to custom design the hardware (less printer) with no major changes, except for removing redundant and unnecessary circuitry, one would probably have a well-stuffed 8" x 10" board. Given future (fourth quarter 1978) technology, such a board could probably be built in a fourth of the size and at half the cost.

The hardware that is used for this project consists of:
1. DEC LSI-11 microcomputer (see A-3)
2. VOTRAX VS-6 Voice Synthesizer
3. Intercept Interface
4. Triformations, Inc. BD-3 Braille strip printer

LSI-11. Microcomputer selection was based solely on processing speed, physical size, and high level language support. Processing speed is really a function of what kind of operations are most frequently performed. For the speech synthesis software, it was empirically discovered that 16 bit pointer arithmetic operations would be the most common (70% of the time the software is searching or indirecting through matrices, many larger than 256 bytes long). The LSI-11 was chosen because it can handle pointer arithmetic more rapidly than other available choices (Z-80,8085,6800). A high level language (C) was also available that generated efficient code.

VOTRAX Synthesizer. Speech synthesizers are constantly improving, as need for clearer speech is required. The pace of such change is so great that last year's products usually are surprisingly outmoded by current products. The synthesizer unit used in this project is an example of this; although it has moderately good performance (intelligibility), in a few months it will probably be superseded. However, this unit was chosen because it was available, and it generates reasonably clear speech. It has a 64 phoneme sound-capacity with four possible levels of inflection (one should remember that a phoneme is somewhat of an abstract concept of being a basic sound from which words are made up. Also, phonemes vary between languages and dialects. When a manufacturer advertises a device with a capacity of 64 phonemes, this means the device has 64 available sounds which mimic some basic sounds in a dialect of a language; usually not complete in coverage.)

Intercept Interface. At the start of this project, some limitations were placed on the interface hardware. One was that no modifications of any kind would be made of the calculator. In other words, access to the signals coming from the calculator's keyboard could not be made via "pick offs" on the circuit boards of the calculator, but instead must be made in a more civilized manner by attaching to some connector already available on the outside of the calculator. It was found that keyboard and display data could be obtained by subtle decoding of the calculator's I/O bus. The device which will accomplish such decoding is known as the Intercept Interface, which is presently being created by NASA Engineer, Donald Billings. By using a Hewlett-Packard built card assembly to buffer calculator data bus lines, no direct electrical connection will be made to the calculator from the microcomputer, so both units will be isolated.

Triformations Braille Printer. Rounding out the systems hardware completely, a Braille printer from Triformations, Inc. allows limited hardcopy use. Although the main emphasis in this project is to demonstrate voice methods,

some hardcopy capability is desirable (good engineering practice). The BD-3 printer used here, embosses Braille on a strip of paper (Braille ticker tape!) Something should be mentioned about Braille: Contrary to popular belief, not all visually handicapped people read Braille. Many use low vision aids, which allow them to read with what limited vision they have left, by hand scanning a portable TV camera over text which is viewed on a TV screen.

Software

The voice system has a large amount of software, most of it is concerned with translating text strings into phonetic strings. There are five major procedures:
1. Primative - I/O Monitor (PRIM)
2. Pronunciation by Rule (RULE)
3. Statement Symbol Translator (SYMBOL)
4. Display Parser (PARSE)
5. Braille Code Converter (CODE)
The interaction among these procedures is illustrated, (see A-4). Before discussing this figure, the individual procedures should be described.

PRIM. The Primative I/O Monitor functions as a buffer to the I/O devices for the main procedures. All device-dependent code is present here, so that communication between device and system software is via queues. Housekeeping functions, like device error detection/recovery, are also PRIM's responsibilities. By organizing the monitor this way, only one procedure must be modified to allow for different hardware.

RULE. To allow for a flexible vocabulary, pronunciation by rule was chosen as the method to convert text into phonetic text, which is more palatable to the voice synthesizer. Procedure RULE will accomplish this by checking input text for rules that might apply and performing simple transformations on the text when the given rule applies (done in real time). Some of these transformations are quite simple, like removing the silent "e" from the ends of words, while others search for sinister medial vowels (like the "e" in "houseboat"). A very readable article by Allen [4] describes this process well, including both successes and failures of rule sets (a translation example from this paper is reprinted, see A-5). Failures usually result in comprehensible but unusually pronounced words (not unlike the way a young child will pronounce a new word). A study of a rule system, similar to the one that will be used here, shows that it produces intelligible speech on 97% of running text [5]. Considering that it is almost impossible to maintain a dictionary of such scope (also considering accessing such data in real time), this is very reasonable performance. This capability is achieved by approximately 800 rules. The number of rules is limited by

computer speed and memory storage; it is conceivable that with faster computers and larger memory, performance could be increased.

In RULE no special processing is done that is particular to English language. Only generalized rules are used, allowing this system to be used with other languages. To change languages, all that is required is: 1) a new set of rules to express phonetic transcription of the language, 2) new symbol translation table, and 3) new voice synthesizer (only needed if new language has a different set of phonemes; in the case of Spanish, a trilled r is needed, and in German vowels like ü are also required). It would be possible to have a multilingual unit where rule sets could be selected (probably by means of a memory bank switch), provided the voice synthesizer has a large enough selection of phonemes.

SYMBOL. A particular problem with most computer languages is that they contain unpronounceable expressions, like * / + > = ¯ etc. which must be pronounced by use of a symbol dictionary. This is the function of SYMBOL, to pronounce programming language symbols. SYMBOL will be large, due in part to the large number of program symbols (approx. 200) the calculator has (this includes program mnemonics, like prt for print, gto for go to, cll for call, etc.) In addition to pronouncing symbols, keyboard echo feedback is accomplished by this module (to avoid unnecessary duplication of code).

PARSE. In order to separate program symbols from English text that is input to the system, a simple LR parser is used to make the distinction. PARSE does not blindly separate symbols and English, but instead attempts to determine if the given text should be considered as English text, program symbols, or raw program data, or a mixture of both. For example, we can have the program symbol "prt" (meaning print), or the English text "prt" (unpronounceable), or program data "prt" (pea are tee). Only syntax can decide (not always successfully though) which one of these ways the symbols should be treated.

CODE. A problem in using Braille printout for computer use is that standard Braille does not have all the special symbols required by most programming languages. The disparity between standard Braille and ASCII character sets is quite large. The differences are critical, as a line of program code that might appear (in say BASIC) as: P=X↑3 + 20 * X * Y + 3/Z would have the Braille form of P=X3 + 20XY + 3Z; which is totally different in meaning. There are a few methods to deal with this difficulty; the Braille character set can be expanded, or the unrepresented characters can be expressed as combinations of existing characters. Expanding the Braille character set has the obvious prob-

lems associated with changing a widely used standard (can you imagine all the trouble that might occur if ASCII was extended from 7 to 9 bits/character?) Playing with such standards shouldn't be done lightly, as it might have disastrous consequences for general purpose use (for example, making Braille much more difficult to learn or use).

To avoid these problems, one can use the standard code and use combinations of characters to represent special characters, like using "greater than" to represent ">". This has the advantage of incorporating all character sets that can be described (i.e. in APL, ⎕becomes "quad quote", but there is difficulty with an arbitrary symbol like ⍉!) A disadvantage is that it now takes 8 - 10 characters to represent one symbol, which wouldn't be bad if it wasn't for the fact that Braille takes up four to six times as much space to print as standard text. The compromise that has been chosen is to use compressed mnemonics, like "cln" for ":".

With the above adjustment in character sets, the Braille software allows listing of the calculator's program on the Braille printer. An ideal situation would be to list Braille and typed versions simultaneously on the same paper in adjacent columns. This would allow easy discussion of the program between sighted and unsighted programmers, as each could locate errors or discuss critical sections without shuffling around. Unfortunately, this takes special hardware which is not available.

How It All Fits Together. The software procedures described interact with each other along the lines in the Program Interaction Graph (see A-4). PRIM acts as a transparent buffer to the other four procedures by performing I/O functions to/from a device, from/to a queue. PARSE reads the current display line from a queue, and then separates the line according to item type to either SYMBOL or RULE. SYMBOL and RULE translate their text into phonemes, which are left in the voice synthesizer's queue. SYMBOL also examines individual keypresses from the keyboard and echoes their name with the synthesizer. Finally, CODE translates any printing requests into expanded Braille format and outputs reformatted text to the Braille printer.

Conclusion

A system has been described here which will allow the visually handicapped to use an HP 9825A desk-top calculator, by means of voice communication from a VOTRAX voice synthesizer. An LSI-11 microcomputer will translate program and written text into phonetic text for the synthesizer. This translation is accomplished by using a set of rules. Echo feedback of the calculator keyboard is also done by the microcomputer. In addition to the voice system, a Braille strip printer will be used to provide hardcopy on demand. The complete system will be compact and portable.

ACKNOWLEDGMENTS

REFERENCES

[1] National Aeronautics and Space Administration, Contract A4576B.

[2] Elson, Benjamin M., "Inexpensive Avionics Concepts Being Sought,"
Aviation & Space Week, August 1, 1977.

[3] Ritchie, D. M., C Reference Manual, Bell Laboratories,
Murray Hill, N. J. 07979

[4] Allen, Jonathan, "Speech Synthesis from Unrestricted Text,"
from a collection of papers in Speech Synthesis, edited by Rabiner.

[5] McIlroy, M. Douglas, "Synthetic English Speech by Rule",
Bell Telephone Laboratories, Murray Hill, N. J. 07979.

A-1 Text Accompanying Photo

"The HP 9825A Calculator is seen here in use.
The dark window near the top is a 32 character
LED-ASCII display. The VOTRAX VS-6 Voice
Synthesizer is the large box sitting on top of
the calculator. Voice pitch, speech rate and
volume are adjustable from the potentiometers
on the front of the case."

SAMPLE SESSION EXAMPLE

Operator (Keypresses)	Calculator Display	Voice Output
d	d	dee
s	ds	es
p	dsp	pea
"	dsp "	quote
h	dsp "h	hay-ch
e	dsp "he	ee
l	dsp "hel	el
l	dsp "hell	el
o	dsp "hello	oh
"	dsp "hello"	quote
store	0: dsp "hello"	store step zero display quote hello quote
run	hello	run hello
fetch	fetch	fetch
0	fetch 0	zero
execute	0: dsp "hello"	execute step zero display quote hello quote

Note: This HP9825 calculator uses STORE as an end-of-line button for program statements and uses EXECUTE for indicating end-of-command.
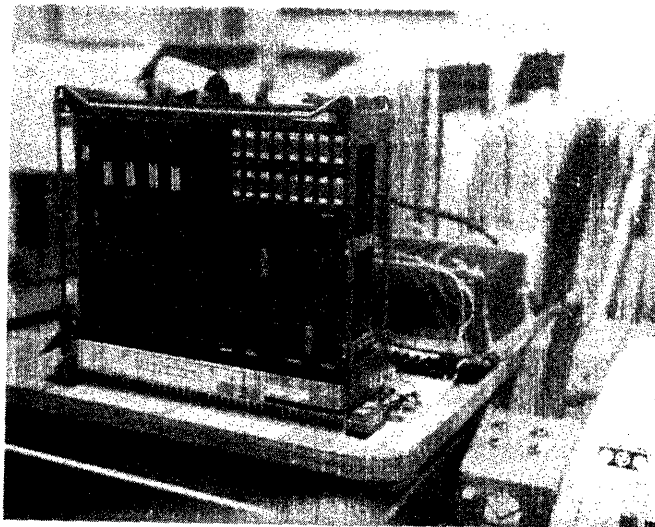
A-3 Text Accompanying Photo

"Here is a Digital Equipment Corp. LSI-11 microcomputer. The five large chips on the facing board contain the microprocessor and its control store. In the background can be seen supporting power supplies and console teletype."

A-4

I/O Devices　　　　　　　　　　　Queues　　　　　　　　　Main Procedures

VOTRAX VS-6
VOICE
SYNTHESIZER

INTERCEPT
INTERFACE

TRIFORMATIONS
MODEL BD-3
BRAILLE PRINTER

PRIM

PRIMITIVE
INPUT/OUTPUT
MONITOR

PHONEMEN

KEYBOARD

DISPLAY

PRINTER

RULE

SPEECH SYNTHESIS
BY RULE

SYMBOL

STATEMENT SYMBOL
TRANSLATOR

PARSE

LR PARSER

CODE
BRAILLE
CODE
CONVERTER

A-5

Reprinted from Allen [4].

\# CHROME \#
\# CHROME \#
\# KROMS \#
\# KROM \#

\# MYTHOLOGY \#
\# MYTH + OLOGY \#
\# MS^TSSSSSS \#
\# M^I^T +^ALY^JE \#

\# DISPASSIONATELY \#
\# DIS = PASSION + ATE + LY \#
\# SSSSPSS ...　　　S \#
\# D^IS= P"A^SYN + YT + LE \#

=weak vowel
sound
(like "uh")

"　=plosive

=,+=pause

LETTER-TO-SOUND CONVERSION

WEST COAST COMPUTER FAIRE　　　　64　　　　**BOX 1579, PALO ALTO CA 94302**

DEVELOPMENT OF PROTOTYPE EQUIPMENT TO ENABLE
THE BLIND TO BE TELEPHONE OPERATORS

Susan Halle Phillips
Vocational Coordinator
Sensory Aids Foundation
399 Sherman Avenue, Suite 4
Palo Alto, California 94306
(415) 329-0430

This report describes the development of prototype equipment to interface a blind telephone operator to a TSPS console. Sensory Aids Foundation contracted with Telesensory Systems Inc. to build an interface utilizing voice output to determine if a blind telephone operator could perform the job of TSPS operator within the performance requirements set by Pacific Telephone Company for sighted operators.

The system has been successfully tested and two persons have been placed as operators within the Pacific Telephone System. This project will now enable blind people to be competitive for job placement within the Bell Telephone System.

Sensory Aids Foundation, a non-profit corporation, and the California Department of Rehabilitation established a program to develop new employment opportunities for job ready persons who are blind. This jointly funded grant provides for the expansion of entry level occupations through the application of sensory aids and the development of new adaptive devices. During the first two years of the Innovation and Expansion Project, 63 blind and partially-sighted individuals were placed in a variety of employment settings.

One of the primary engineering projects initiated by Sensory Aids Foundation was the development of prototype interface equipment for blind TSPS (Traffic Service Position System) operators. The goal of this project was to open the TSPS operator position with Pacific Telephone Company to blind people and to enable them to perform the job competitively within the standards set by the Telephone Company for sighted employees.

The TSPS is the specific console used by Pacific Telephone Company long-distance telephone operators to perform their jobs. Generally, the TSPS computer console handles all calls requiring operator assistance, with the exception of "information" number requests. Sensory Aids Foundation contracted with Telesensory Systems Inc., Palo Alto, to build two prototypes of TSPS console overlays which monitored nixie tubes and 72 lighted indicator lamps and buttons with voice output systems, utilizing the Votrax Voice Synthesizer. The voice output prototypes would give to the blind operator audible information that a sighted operator sees. This information is necessary for the operator to recognize the state of the console and to service customer requests.

System Design

The various elements of the TSPS Interface System are illustrated in Figure A-1. At the top of the figure is the TSPS console itself. It consists of the nixie tube display and the operator control panel with lighted pushbuttons and indicators. The overlay is placed directly over the operator control panel. Each button on the overlay is made from clear acrylic and is provided with a photo-transister to sense if the button is illuminated. Because the buttons are transparent, the status of the console may be determined by a sighted operator. This is important both in training and when the supervisor needs to assist in handling a call.

The Optical Sensing System is placed into the recess formed by the housing of the nixie tube display, as illustrated in Figure A-2. This system is composed of 12 identical modules each sensing a single nixie tube and thus avoids the difficulties of mechanical scanning. The system is both removable and portable; an adjustment is provided to align the Optical Sensing System to a particular console.

The Interface/Control Electronics Module accepts commands from the Operator Control Box, processes data from the Optical Sensing System and console overlay, and generates suitable output to a Votrax Voice Synthesizer. The output from the Votrax is presented to the operator via a second earphone. With this control box, the operator

can manually interrogate various sections of the console or read the nixie tube display, as illustrated in Figure A-3. The entire system is carried on a moveable cart with self-contained power supply, as illustrated in Figure A-4. The output of a 12 volt DC battery is converted to 115 AC by an inverter and provides enough power for at least 8 hours of continuous operation. A battery charger is also provided so that the entire system may be recharged when not in use or at the end of the shift.

System Operation

As a call comes in to the TSPS console, appropriate lamps light up which signal the kind of call, class, and charge. For a blind person to service the incoming call, these visual cues must be transformed into spoken words.

A step by step outline of the process is as follows:

a) An incoming telephone call stimulates a specific pattern of lamps to light on the TSPS panel;

b) The computer recognizes which lamps are lit via photosensors on the overlay;

c) The computer determines which words should be spoken;

d) The computer signals the Votrax and speech begins.

Spoken words from the Votrax give the blind operator the cues which the sighted operator obtains visually. These cues are necessary to make the appropriate response to the customer.

Figure A-5 illustrates the TSPS console position for two operators. The right side has the interface prototype equipment for the blind operator.

Additional Equipment Modification

A second important part of the project was to modify the job station equipment, other than the TSPS console, so that it could be efficiently used by the blind operator.

Operators must continually refer to handwritten notes in order to remember names when making person-to-person or collect calls, rate-and-route information for overseas calls, and other information used in filling out manual billing tickets. A suitable device was needed for blind telephone operators. Equipment on site could not be noisy. A quiet, sturdy, small brailler was purchased

for this purpose from the Royal National Institute for the Blind, London, England. Information stored on plastic cards used by all TSPS operators, including area codes, operator codes, numbers of business offices, repair services, and emergency numbers were brailled in a format designed by the Sensory Aids Foundation Staff. A template (Manock Comprehensive Designs, Palo Alto) was designed to enable the blind operator to complete the Mark Sense computer ticket. The blind operator was then able to transfer information from the braille notes to the billing ticket by lifting the template over the ticket and using a Mark Sense pencil to mark appropriate digits.

Summary

As a result of cooperative efforts between the Sensory Aids Foundation, Telesensory Systems Inc. and Pacific Telephone Company, adaptive equipment and job station modification to enable the blind to function as TSPS console operators have been successfully developed. Using prototype console overlays, two totally blind individuals have been competitively placed as TSPS operators in the Mountain View, California Pacific Telephone System. With the success of this pilot project, it is anticipated that the TSPS interface equipment will permit the blind to be hired as telephone operators throughout the Bell System.

Acknowledgements

I wish to express appreciation to A.J. Sword, M. Linvill, J. Azevedo and D. Farr for providing technical assistance and photographs, and to C. Anderson for secretarial assistance.

Appendix

Figure A-1 Prototype TSPS Interface
            Equipment - Schematic
Figure A-2 Nixie Tube Display Reader
Figure A-3 TSPS Console with Overlay, Nixie
            Tube Reader and Control Box
Figure A-4 TSPS Interface Equipment
Figure A-5 TSPS Console Position
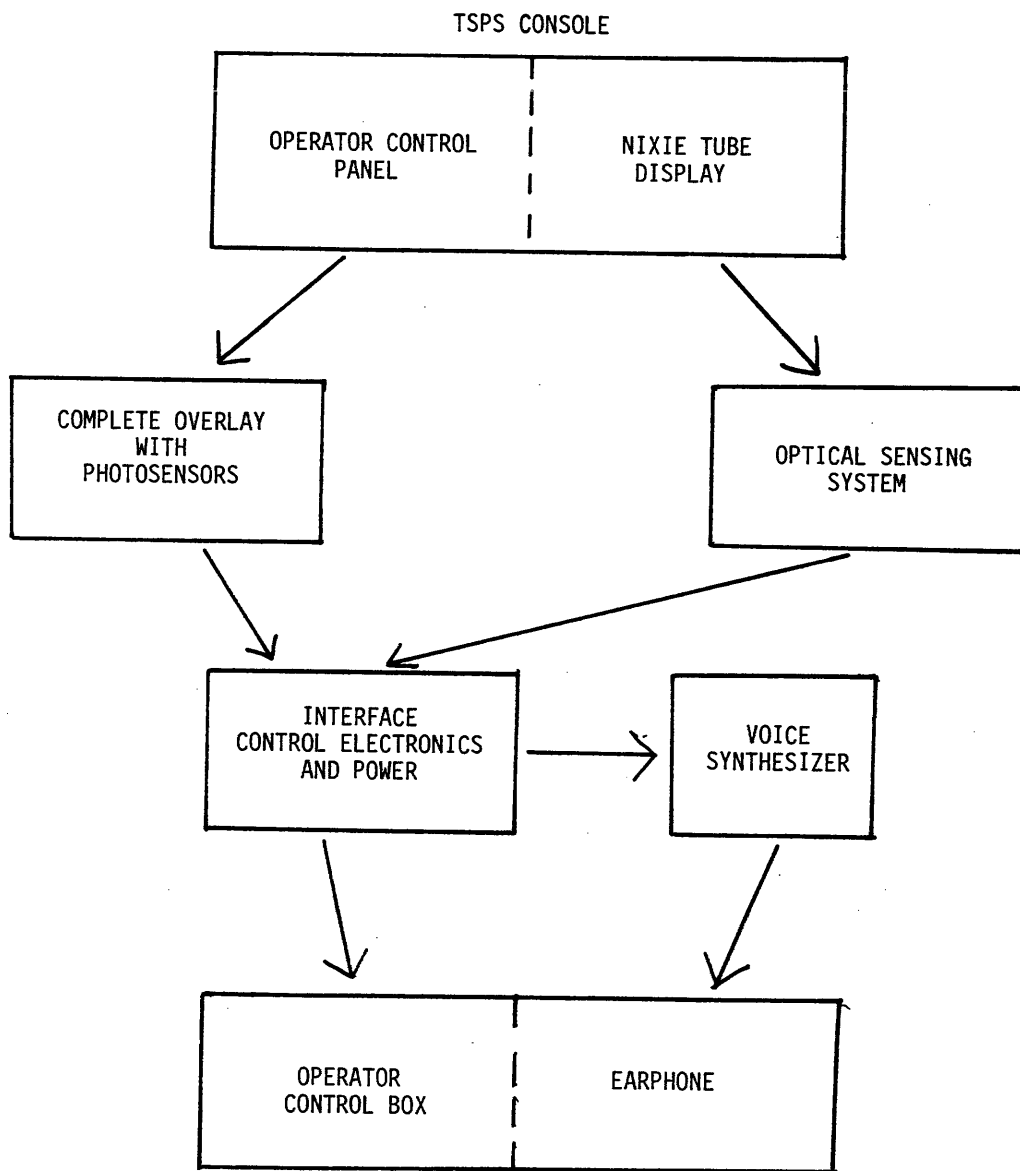
TSPS CONSOLE

Figure: A-1   Prototype TSPS Interface
Equipment - Schematic
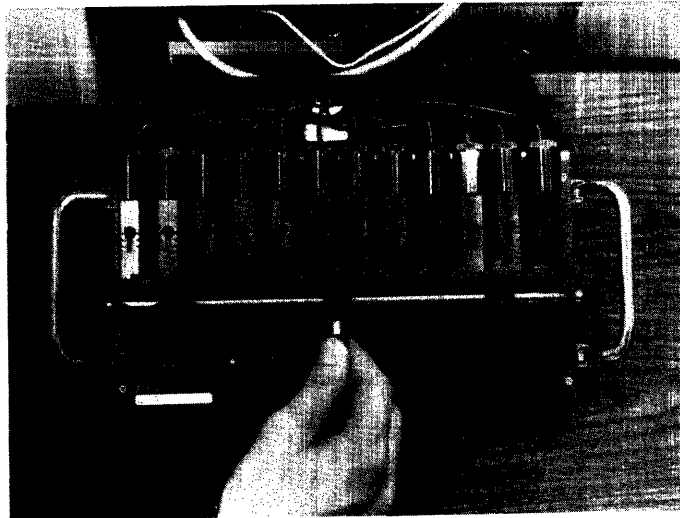
FIGURE A-2
Nixie Tube Display Reader
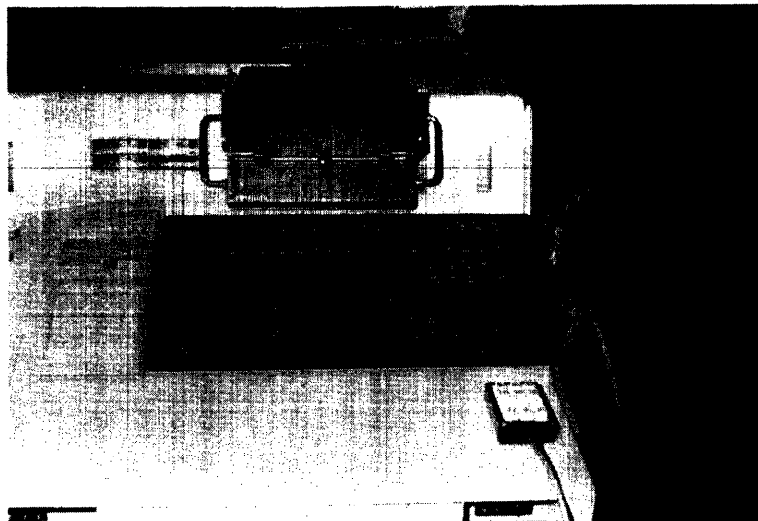
FIGURE A-3
TSPS console with overlay, nixie tube reader & control box
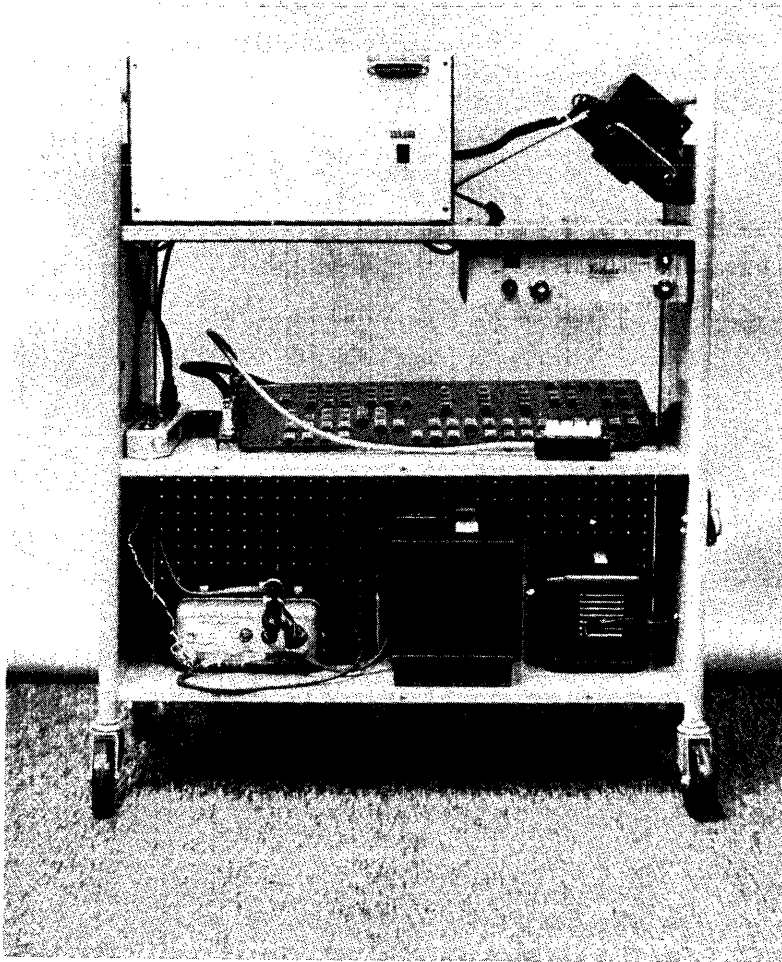
FIGURE A-4
TSPS Interface equipment
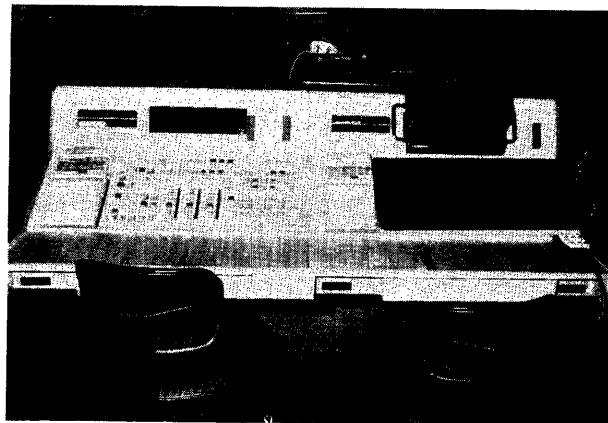
FIGURE A-5
TSPS Console position

MICROCOMPUTER-BASED SENSORY AIDS
FOR THE HANDICAPPED

J.S. Brugler, Ph.D.
Vice President Engineering
Telesensory Systems, Inc.
3408 Hillview Avenue
Palo Alto, CA

INTRODUCTION

Telesensory Systems, Inc. was formed in order to provide high technology aids for the handicapped, especially the blind. Nearly all of our recent projects have utilized micro-computer technology. The resultant programmable aids provide performance and flexibility impossible to achieve in the past. This paper gives details of five of these devices - the SPEECH+ talking calculator for the blind, the TSPS telephone console interface for a blind operator, The Games Center for the blind, the Crib-O-Gram, and the LSI Speech Synthesizer. Some of these devices will also be demonstrated.

SPEECH+

The SPEECH+ is a hand held, battery operated calculator developed expressly for the blind. Our research concluded that speech is the most effective calculator display modality. SPEECH+ therefore contains a built-in limited vocabulary speech synthesizer which provides the blind user with spoken speech verification of every keystroke and readout of the display upon command. The speech unit also indicates overflow and low battery conditions. In addition to English, units speaking German, French, and Arabic have been programmed.

Inside the unit, speech and control data are stored on a single 16K bit mask-programmed ROM. A custom LSI microcontroller chip accepts input commands and looks up the "recipe" for speaking the desired word. It then constructs the proper speech waveform from the stored speech data. To reduce storage costs, a number of unique encoding schemes are used. The keyboard scanning, calculating and speech code generation are done by a TMS-1000 single-chip microprocessor. The use of three large chips - the custom microcontroller, the TMS-1000, and the 16K ROM - plus a minimum of support circuitry enable a convenient portable unit to be made available at reasonable cost.

Since the speech data is stored in ROM, vocabularies are easily changed. In addition to foreign vocabularies, other speech vocabularies can be generated. To try other application besides the calculator, a "general purpose" and an ASCII vocabulary have been programmed. Potential applications being explored include talking elevators, talking meters, and talking computer terminals.

TSPS

The telephone operator's job, being auditory, has in the past been very well-suited for the blind. The introduction of computer-based "TSPS" telephone exchange systems having many visual cues for the operator has made the job impossible for a blind person. TSI undertook a demonstration project to develop interface equipment which would enable a blind TSPS operator to compete successfully with a sighted TSPS operator.

The TSPS (Traffic Service Position System) console used by the operator is a special purpose computer terminal containing over 80 pushbuttons and a 12-digit numeric display. The hardware we developed is based on a 6800 microcomputer. In operation, special optoelectronic circuits read the lighted pushbutton and numeric display status information into the computer. The computer interprets the input data and tells the blind operator, via synthesized voice, the information necessary to handle an incoming call. Once a call is serviced, the computer continues to monitor the console to verify that the call was completed properly. Since a bewildering variety of type of incoming calls are possible, the details of the system operation are quite complex, and have undergone considerable evolution. An important feature is the ability of the operator to interrogate several important console parameters.

Two demonstration systems were built, and are being used every day by two blind operators at Pacific Telephone. Quantitative performance measures are quite encouraging, and we are seeking funding to production engineer the system to enable widespread usage.

THE GAME CENTER

The availability of low-cost speech technology, keyboards, and microcomputer know-how prompted us to breadboard a set of electronic games for the blind. The breadboard unit met great success, so we will soon be producing a limited number of games units for use at agencies and centers for the blind.

The unit's electronics consist of an 8800 processor board, a speech board, and an analog board. Of the eight games, seven are played on the keyboard. Three games are modifications of well known pastimes (Blackjack, Craps, Tic-Tac-Toe), while four were specially invented ("Skeet-Shoot", "Number Run", "Tug-O-War", and the "Chain Game"). The eighth game, called "Paddleball" is a simulated video game involving hitting of a moving "ball" heard via stereo earphones. The ball is indicated spatially by a tone going up and down in pitch and back and forth in stereo separation. Scoring in "hits" is done with the microcomputer until one of two players wins.

CRIB-O-GRAM

Crib-O-Gram is the only project we have undertaken outside the field of aids for the blind. The Crib-O-Gram is a screening device for potential hearing loss in newborn babies. It is designed to automatically test infants while in the crib, and to flag those with a possible hearing problem. Babies that fail the test are then thoroughly rescreened at six months of age.

The system is controlled by an 8080 microcomputer. A sensitive motion transducer is placed under the mattress, and a small loudspeaker mounted nearby. The computer monitors the state of the baby's activity, and when conditions are appropriate, turns on a 92db 2-4 kHz white noise stimulus. The computer then determines through various algorithms whether the baby reacted to the sound. In order to insure statistical validity, a number of tests are given over a 24-hour period,

some of which are silent control tests. The computer keeps score and, at the completion of the test sequence, gives a pass or fail indication.

The Crib-O-Gram System is presently undergoing evaluation at the Stanford Hospital. The hardware is designed, and various software improvements are continually being added. Workers in the field agree that early detection of hearing loss is vital. The Crib-O-Gram, by using microcomputer technology, will make available an automatic, low cost hearing screening technique so that remedial steps can be initiated at an early age.

LSI SPEECH SYNTHESIS

The human vocal tract can be electrically simulated by means of series and parallel resonators driven by periodic and noise excitations. Their parameters (center frequency, band width, and gain) are varied as a function of time to create the various speech sounds. This technique is called "Formant Synthesis". In contrast to the techniques described in section I, an unlimited vocabulary of utterances can be generated. Nearly all experimental and commercial formant synthesizers have utilized analog circuitry. This circuitry is prone to tolerance and drift problems and is relatively costly and inflexible.

To circumvent the inherent problems of an analog synthesizer, we have simulated and breadboarded an all-digital formant synthesizer suitable for ultimate fabrication in LSI form. This unit is programmable, so can implement a variety of high performance synthesizer structures. If production volumes can be high enough, this synthesizer promises to be sufficiently inexpensive to find wide application in aids for the blind.

CONCLUSION

Through the use of silicon technology, Telesensory Systems, Inc. has developed a number of devices for the handicapped. Applications range from serious, such as vocational aid and hearing screening, to recreational games. Such devices can help erase the somewhat artificial distinction between the handicapped and the "normal". Our ideas for further applications always exceed the available resources and time.

ACKNOWLEDGEMENTS

The TSPS project was sponsored by the
Sensory Aids Foundation, Palo Alto, Califor-
nia. The cooperation and help of Pacific
Telephone was vital.

The Crib-O-Gram was conceived by Dr. F.
Blair Simmons and is licensed from Stanford
University.

The LSI Speech Synthesis work was
partially funded by The Seeing Eye, Morris-
town, NJ.

AMBITIOUS GAMES FOR SMALL COMPUTERS

Larry Tesler, Xerox Palo Alto Research Center
3333 Coyote Hill Rd, Palo Alto, Ca 94304

Abstract

Some of the more challenging games that can be played
on a computer require more memory than is available on
today's personal systems. The paper presents various
simple encoding techniques that were used by the author
and a friend to implement an interesting subset of a
complex game on an 8 kilobyte microcomputer.

Beyond PONG

Given the choice of a PONG machine and a pool table, I
will always choose the latter. Hand-eye coordination is
somehow more challenging when the whole body is
involved. Moreover, the feel of the cue and the sounds
of balls colliding and dropping into pockets appeal to my
senses more than little plastic levers and electronic beeps.

On the other hand, when I am home and seeking a bit of
solitary escape, I no longer turn to Solitaire, crossword
puzzles, TV, or even (unfortunately) reading. I switch on
the computer and play a game like Roger Chaffee's
version of *Adventure*.

*YOU'RE IN THE ASHRAM. THE SMELL OF
INCENSE IS HEAVY HERE, AND ALL
DIRECTIONS SEEM THE SAME.*

WHICH WAY? S

*YOU'RE IN THE LAND OF XANADU. BELOW
YOU ALF THE SACRED RIVER RUNS
THROUGH MEASURELESS CAVERNS DOWN
TO A SUNLESS SEA.*

WHICH WAY? D

YOU CAN'T GO IN THAT DIRECTION.

WHICH WAY? W

*IN A DARK TUNNEL, YOU STUMBLE OVER A
HARD OBJECT. IT FEELS LIKE A METALLIC
CHEST, AND THERE SEEMS TO BE A LATCH.
THE LATCH IS RUSTY, BUT THERE IS NO
LOCK IN EVIDENCE.*

DO YOU WANT TO OPEN THE CHEST?

An hour or two later, after a hundred or more moves, I
have explored all the twenty-five rooms of the labyrinth,
found the treasure, lost it, found it again, and escaped
safely to the surface with my skin intact.

Hungry for greater challenge, I wish my $800
Commodore PET were an $800,000 megaputer, so I could
play Will Crowther's original *Adventure*, MIT's *Dungeons*,
and other complex games based on the *Dungeons and
Dragons* theme. With those versions, I could create
labyrinths for others to explore, with a hundred rooms,
multiple treasures, assorted demons, magical objects, and
elements yet to be conceived. The adventurer could give
commands in stylistic English phrases ("take jewels",
"throw rock") instead of by answering multiple-choice
questions.

So I study ROGER'S little BASIC program, marvelling at
how he fit even thirty flowery descriptions of rooms and
predicaments into the tiny computer, along with their
interconnection toplogy, not to mention a selection of
tasty algorithms for deception, subterfuge, and final
reward.

The choice: buy more memory? a floppy disk? or... learn
sardine canning...

A clear case of Aladdin economics. Stuff the genie into
a bottle.

more bits per byte

More bits per byte! No, that won't work.

SQUEEZINGOUTSPACE

The ALTAIR BASIC manual has bunches of techniques
for reducing program space. These work in other
BASICs, too, including the PET's. Eliminate REMarks
(ugh), eliminate blank spaces (retch), use the same
variable for several purposes (horrors).

Such travesties against good programming technique can
only be justified in life-and-death situations. But what
could be a greater emergency? The goblins must be
overcome!

Roger has already squeezed the program quite a lot. I can
obtain another couple of hundred bytes by making the
program completely illegible. (Please look the other way,
this is not for children to see.)

Encoding the Graph

Hmmm. The topology of the labyrinth is specified in DATA statements. Roger has already saved a lot of space by encoding some predicaments ("you can't go in that direction", "the giant is here, you'll have to get out") as if they were rooms. They are marked specially to indicate that after the description is printed the adventurer should be forced into a real room, either the one from whence he came or some other chosen partly at random or partly based on possession of the treasure.

Each room (and predicament) lists its own number and six connection numbers, one for each of North, East, Up, Down, West, and South. The connection number is either 0 to mean no exit in that direction, or the identification of another room. A typical example:

```
9020 DATA 2,4,0,23,29,0,0
```

There are more than sixty distinct characters available on the machine (actually, 128), so why not assign a character to each room? Then the connectivity specification is simply:

```
9020 DATA BD#X]##
```

The D signifies a connection to room D, the # means no exit. In general, A-Z represent 26 of the rooms, and then on beyond Zebra for the rest.

This seems to cut 15 characters of DATA to 7. But actually, it is better than that, because these DATA statements have to be READ. When a number is read into an integer array, it is converted to binary and stored as a sixteen bit quantity in two bytes. So the array storage for six numbers takes 12 bytes of storage for each room, in addition to the 15 in the DATA statement (for a total of 27). But when a string is read into an array in this BASIC, it takes only 3 additional bytes of storage (for a total of 10), because the characters of the string are not copied: two of the three bytes are an address into the DATA statement itself and the third byte is the string length.

It takes about the same size program to deal with either representation, so we have made a net gain of 17 bytes per room, or 510 bytes for 30 rooms. Enough for some dwarves, elves, and magic rings no doubt.

YIOU9$?

The connectivity specification is not the bulk of the data base describing the labyrinth. Most of it consists of those flowery text descriptions:

```
9020 DATA 2,4,0,23,29,0,0,"YOU'RE IN A
NARROW EAST WEST TUNNEL, WITH AN OPENING
ON THE NORTH SIDE TO A WIDE PASSAGE.
WATCH OUT FOR GOBLINS!"
```

Though flowery, the vocabulary is rather repetitive. The word 'gnome' is used in two different rooms, 'there' is mentioned in 7 places, 'to' in 9, 'in' in 13, and the word 'the' occurs no less than 33 times. Altogether, there are 181 different words, punctuation marks, and word-endings (no, I didn't count them, the computer did). The sixty "words" used more than once account for 60% of the text. Let's assign the following one-character abbreviations to the most common English words and endings employed in modern dungeons:

A=a, B=better, C=can, D=direction, E=-ed, F=for, G=go, H=here, I=I, J=giant, K=cavern, L=little, M=message, N=not, O=of, P=pit, R=rock, S=-s, T=the, V=room, W=wide, X=Bilbo, Y=you, a=above, b=but, c=climb, d=-d, e=-es, g=get, h=tight, i=in, j=guillotine, l=ledge, n=on, o=opening, r=are, s=is, t=to, v=chamber, w=was, x=gnome, y=you're, &=and, \=through, ↑=there, ←=back

We can also make some of the PET graphic characters stand for words: ⊥ for 'north'; a high-up line for 'top' and a low-down line for 'down'; something that looks thick and vertical for 'wall'; and so forth. Over 120 words can be represented this way in PET BASIC. Although some BASICs only provide 60 to 100 distinct characters, one can always encode about 250 words with an 8-bit code processed in machine language. Anyway, sixty words does seem to be a sufficient vocabulary of repeated words. The above room description boils down to this:

```
9020DATA"BD#X]##yiA6NARROW├──┤6TUNNEL,4W
ITH2ANonT⊥4SIDEtAW7PASSAGE.5WATCH3OUTF7
GOBLINS!
```

Don't tell me -- that's the kind of stuff your teletype always comes out with! Good; you already know how to read it. After DATA" are the seven characters that we discussed earlier for the connectivity description; might as well combine them with the string used for the text to save a little more space. Next comes the text: y=you're, i=in, A=a, and then 6NARROW -- which indicates a six character word not in the vocabulary -- then ├=east, and so forth.

Various conventions too detailed for this paper are used in both the vocabulary and the descriptions to control the insertion of spaces between words and the formatting of lines of print. The printing subroutine runs a lot slower than it used to, but it is still faster than a person can read.

Compressing the room descriptions saves a lot of space; but the printing subroutine gets longer and the vocabulary has to be stored. The net gain is about 400 bytes. Compressing other messages printed by the program should save another hundred bytes.

The Bottom Line

A combination of the above techniques frees over 1000 bytes of storage. A new room whose description shares a lot of the existing vocabulary can be added at a cost of only 40 or 50 bytes, so we could add another 20 or more rooms.

I would rather add new twists that are available in bigger implementations, such as objects that are found in rooms and that can be taken along with the adventurer. No more than two objects could be carried at a time. Some objects would be valueless, others would be valuable to take out of the labyrinth at the end of the game. Still other objects would be useful during exploration, each to surmount a specific obstacle that can be encountered. To determine the properties of the objects, the adventurer would have to experiment and take risks.

Educational Applications

Although *Adventure* is cast as a one-player game, it should be possible to have several people collaborate.

One way to collaborate is to sit at the terminal together planning moves and developing a model of the labyrinth. Another way is to explore at different times, all agreeing to end up in a particular room after the next session. Those who make it back there may then compare notes and try to benefit from the experiences of others.

With more memory, a multi-discipline educational game with a similar structure could be concocted by running several simulations in parallel. For example, the adventurer is an international trader. His or her business partner (a simulated being) is off on a voyage but never remembers to write home. The partner must be found within a year or the business will be taken over by a sinister cartel, or the IRS, or something like that.

The adventurer travels through foreign lands seeking clues. He must buy and sell commodities in order to finance the trip. The supply and demand of commodities is affected by location, time of year, and random factors such as weather and luck. As time runs out, the opposition erects barriers to progress, and so forth.

Such a game would be an earth-bound blend of *Adventure* and the game *Star Trader* (see the book **What to do After You Hit Return**). It would teach some things about geography and economics, as well as problem-solving, strategy, and planning.

But all this must wait for cheaper data storage. In the meantime, if you'll excuse me, a minotaur is waiting for me in the family room.

EPIC COMPUTER GAMES: SOME SPECULATIONS

Dennis R. Allison, Consultant, Menlo Park, CA 94025
Lee Hoevel, Stanford University, Stanford, Ca 94305

INTRODUCTION

Few themes are more common in litera-
ture than that of the epic adventure.
In these tales the hero, through the
exercise of his wit and brawn, over-
comes all to achieve his objective be
it treasure, romance, power, or what-
ever. Even the pulp novel, now an al-
most extinct beast, owes much of its
structure and character to the tradi-
tional epic.

The traditional epic and its modern
imitators provide rich paradigms for
computer games. Such games are far
more interesting and compelling than
the usual "guess the number" or "zap
the spaceship" fare. Some proto-epic
games already exist, but the most ful-
ly ellaborated games present implemen-
tation difficulties of considerable
substance. In this paper, and in an
earlier paper published in the
March/April issue of People's
Computers we explore some of the pos-
sibilities and problems of this class
of fantasy games.

GAMES AS RECREATION

The futurists and pundits of the per-
sonal computing world all have an-
nounced that the primary use of com-
puters in the not too distant future
will be recreational. On the other
hand, I find nearly all computer and
video games boring. I can't get my-
self interested for long periods of
time.

The first game which got my attention
long enough to be become entranced was
Adventure. It was written by W.
Crowther (now at Xercx PARC) and modi-
fied by Don Woods at Stanford AI. It
is written in FORTRAN and has migrated
to many machines. A much extended
game based upon the same general
ideas, but with significant improve-
ments, has been written by Tim Ander-
son, Marc Blank, Bruce Daniels, and
Dave Lebling at MIT; it is called
Dungeons and is written in a dialect
of LISP out of CONNIVER and PLANNER
called MDL (Muddle). Both programs
are rather large: Dungeons requires
about 120,000 words (36 bits) on a

PDP-10; Adventure is somewhat smaller.
Neither is really microcomputer or
personal computer fare at this point.

The siren-like charm of Adventure is
not limited to computer folk. The oth-
er evening I was dining in a local
Chinese restaurant and chanced to
overhear a table of college women dis-
cussing just how one got around the
green snake, one of the Adventure haz-
zards. And then there is the story of
one of my collegues who introduced his
psychology graduate student neighbor to
the game one evening; the neighbor was
so engaged that he went out, bought a
terminal, and spent the next two weeks
playing Adventure.

Why is Adventure fun to play. Because
it is an adventure! The essence of
the game is exploration of an unknown
and clever fantasy world with hazzards
and treasure. Both Adventure and
Dungeon use a cave as a universe; a
cave in which treasure and adventure
can be found. Both games owe much to
Dungeons and Dragons.

GAME DESIGN

What makes a game fun? interesting?
compelling? The answers are most cer-
tainly burried in the psyche and the
folk-tradition of the player. However
there are certain elements which might
be considered characteristic. First,
passive games are not really interest-
ing. The player must be actively in-
volved. Second, there must be some
fantasy fulfillment; the game must
fulfill some basic psychological need,
however obliquely. Third, there must
be some kind of contest and resolution
of the associated conflict. Fourth,
there must be some reasonable mix of
discovery and invention. Games and
puzzles with one solution are of in-
terest only once. Chess, with its
complexity, is a game of continual
discovery and invention. This leads
to a fifth criterion, complexity. The
game must be adequately complex that
it cannot be known, yet not so complex
as to appear to be random. There must
be some higher rules at work, but not
all motivations and manifestations of
the rules should be clear. Lastly,
there must be variety; boredom feeds
on repitition. That is not to say that

one should exclude ritual (in the sense of folk conventions). These are the fundamental fabric of knowledge by which the game maker and the game player communicates.

A game which follows the epic traditions draws upon a long proven structure. While no real computer game will be textbook perfect, one can expect that some elements of the traditional form will be preserved. The central figure, the hero the player is identified with, is of national, international, or galactic importance. The setting of the game matches the importance of the hero. The hero must perform some difficult deeds in his quest. Gods, daemons, or other supernatural beings may take an active part. The game starts in media res with the player discovering what is happening as the game progresses.

Everyone can make a catalog of fantasy universes which are potential environments for such games. One can extract the universe of dangerous characters and the Oriental Express from the classic spy thriller, the starships and strange beings of the star explorer, the world and honor code of the knights of the round table, and on and on.

SCRIPTING A GAME

Scripting rather than programming is important. The creation of an epic is an enormously complex task. And there is no information about to indicate that programmers are particularly skilled at doing it. The creation of a new game should be supported with special purpose tools so that it is available to non-programmers.

The game author must make a number of choices. He must decide upon the game universe, establish all its natural laws, and create all the persona (players, human or otherwise) who populate the universe. He must also create those inanimate objects of special significance to the game and distribute them throughout the universe. It is rather like playing god.

The real problem here is the non-deterministic nature of the game. All persona might not appear in any given game; the ending is dependent upon how the player responds. It is a bit like a TV script for which all middles and endings are worked out.

Persona are the most difficult problem. Good games need interesting interesting characters to populate their world, both good and evil. Not only this, persona must be able to communicate with the player in reasonable natural language, and the response must be tempered by the character of the persona. While many characters are rather shallow, others must have some psychological depth.

A most important persona is the alter ego of the player himself. It performs his commands and observes the game universe as his eyes, ears, and nose. The alter ego may also have the role of conscience, arguing with the player when he tries to do something out of character.

CONFLICTS AND RESOLUTION

Swordplay at a terminal is not really practial. Yet even the non-violent games need to have some way of resolving contests. One way is to replace the actual contest by an idealized one. When the black knight encounters the white knight, he need not actually joust; a quick game of tic-tac-toe might be equally as fulfilling. Other possibilities come to mind: a simple trivia quiz, an factual quiz, a riddle, or a simple number game.

TOWARDS AN ELECTRIC NOVEL

Perhaps the most exciting possibility is what one might call the Electric Novel. It is like today's escapist literature, except it is a participatory experience. You, the player, are the hero. You don't experience the hero's decisions vicariously; you make them. We have not really yet solved the problems of sex and violence in such literature, but it does make interesting speculation.

With the advent of inexpensive voice recognition and synthesis units, good color graphics, and very very large mass storage devices, the possibilities are even better.

Special "Laboratory" Session on Computer Games:

CREATE YOUR OWN (COMPUTER) GAME
An Experience in Synectic Synergistic Serendipity

Ted M. Kahn

Department of Psychology, University of California, Berkeley
and
XEROX *Palo Alto Research Center*

This session will be held back-to-back with the session on *Extraordinary Computer Games.* Its purpose is to allow small groups of participants to generate and elaborate new ideas for various types of computer games in an "experience it" environment, without worrying about problems and details of specific computer implementations. The session will be semi-structured through the use of META-GAME*, a game which I have devised for generating new game ideas through cross-fertilization between different fields. In addition, various types of typical game pieces and boards will provide stimulus material for game structures. The entire creation process will proceed as an exercise in group problem-solving, one which will allow people to meet and cooperate with each other in an enjoyable atmosphere while working on a common task.

The session will emphasize three important aspects in the creation of original games, computer-based or otherwise. The process is:

- *synectic* - Many interesting ideas come as a result of making connections between fields which may seem to be unrelated (e.g., "Think of a game which involves principles of physics and fantasy.");

- *synergistic* - "A game is more than just the sum of its component parts";

- *serendipitous* - One often discovers unexpected results or treasures while originally looking for something entirely different. This is probably the most exciting and unpredictable part of creative activity.

This session will be of special interest to teachers and parents who are interested in helping children of all ages to develop their own ideas, and to game programmers and non-programmers alike.

---

* META-GAME c Copyright 1977 by Ted M. Kahn and Moshe D. Caspi

PSYCHOLOGICAL TESTS WITH VIDEO GAMES

Sam Hersh and Al Ahumada, Aero/Astro Dept., Stanford U., Stanford, CA 94305

Abstract

Because of the similarity between video game logic and psychological testing logic, a microcomputer designed to facilitate game programming can also be used to present stimuli, acquire responses, and perform the preliminary data reduction required for the testing of perceptual and cognitive abilities. The capabilities of a game system--the RCA 1802 COSMAC Video Interface Processor--will be discussed and demonstrated with three tests. The game-playing microcomputer is an effective tester when the display format requires flexible graphics and little alphanumeric information. Low cost, portability, and ease of programming are its principal advantages. Demonstration tests include an auditory discrimination test, a Sternberg memory test, and a water jar problem-solving test.

The Testing System

A minimal system for presenting psychological test items must have graphics and auditory display capability, two or more response keys, the ability to measure keypress latencies to the nearest hundredth of a second, and it must be programmable. From among the many microprocessors shown at the first Computer Faire, we chose the RCA COSMAC VIP because it satisfied these requirements, was cheap at $275, and was locally available off the shelf. We were also attracted to the COSMAC because an interpretive language was was provided for game programming, RAM was expandable to 4 K bytes on board, the I/O port provided a convenient audio channel, and its low power consumption and small size made it appropriately portable.

CHIP-8, the interpreter, turned out to be surprisingly good for programming tests, because it was optimized for games. It takes up a little over 512 bytes of RAM and provides a timer and a random number generator as well as display control and keypad response collection. The two byte instructions look like machine code, but are easy to learn and use. For example, 8XY4 sets variable X to the sum of varibles X and Y. The instruction DXYN displays the N byte pattern at coordinates specified by variables X and Y.

CHIP-8 and machine language subroutine calls are provided for.

We have had to write three machine language procedures to have psychological testing capability: MOVE, which permits the computation of one display while actually displaying another; DUMP, which outputs a waveform in RAM to the D/A converter at a sampling rate of 10 KHz; and REACT, which measures reaction time with 2 msec precision during visual display.

Psycho-Logic

Psychological tests record a sample of behavior in response to a standard test item. These items might take the form of questions, perceptual displays, or objects to be manipulated. The behavior is usually reduced to a number indicating a response category and/or the time elapsed between item presentation and response. At the end of the test, summary scores are computed. Video games like PONG have the same logical structure. The test item is a moving dot on the playing field, the response is the turn of the paddle knob, and the score is one against you if you fail to deflect the dot.

Demonstrations

Psychoacoustic discrimination. Two auditory waveforms are stored in RAM and are presented in random order. The subject's task is to indicate which is the target sound (or the louder or higher in pitch, etc.). Performance is scored by the number of correct responses.

Memory scanning. A set of from one to six digits is displayed for a few seconds. The subject decides whether or not a subsequent probe digit is a member of the set and responds as rapidly as possible. In this situation, the reaction time is a linear function of the number of items in the remembered set. The slope of this function is a measure of the access time of short-term memory.

Water jar problem. The capacity and current contents of one full jar and two empty jars are displayed. The subject's task is to pour half the contents of the first jar into the second. The total number of pours measures problem solving ability.

Acknowledgment

The water jar program was written by Dr. Richard Marken, Augsburg College, Minneapolis, MN 55404.

COMPUTER ART AND ART RELATED APPLICATIONS IN COMPUTER GRAPHICS:
A HISTORICAL PERSPECTIVE AND PROJECTED POSSIBILITIES

Beverly J. Jones, Ph. D.
Assistant Professor, School of Architecture and Allied Arts
University of Oregon

This slide lecture presents a historical review of computer
art and art related applications in computer graphics from
1945 to the present. Most of the images shown were generated
by large·computer systems and involved extensive programming.
Those aspects which seem most promising for development by
individuals or community centers with small computers are in-
dicated. The research, educational, recreational and economic
possibilities inherent in using computer systems for art-
related tasks are briefly discussed relative to the images
shown. Because the conference presentation depends so heavily
on slides, the paper presented here represents only a summary
of ideas. It also includes a resource list of bibliographic
material.

Introduction

Electronic technology, particularly the inform-
ation processing devices known as computers
have the potential of affecting many areas
of our lives. One of these areas is the arts.
Because the computer can generate and man-
ipulate visual images, I believe it offers
promise for use by artists, art educators,
art historians, aestheticians, and museolo-
gists.

The pursuit of this line of thought led to a
collection of slides depicting the images and
objects which have been generated with the aid
of computers from 1945 to the present. A re-
view, analysis, and projections based on this
collection comprise the main body of this pre-
sentation.

Computer science is a unique discipline in
that it has the potential for application in
many fields in which the computer scientist
is not necessarily knowlegable. Conversely
the individual knowlegable in a particular
subject matter area is not necessarily aware
of potential computer applications. I
believe these two statements are especially
true in relation to the arts as a subject
matter area. Part of the reason for this
may be the anti-technological stance which
has been fashionable in the arts in recent
years. As individuals in the arts become aware
that human values and considered choices
based on these values can direct computers,
that computers may be used to individualize
images, objects and events; and that com-
puters need not be used in the mode of mechan-

ical technology, perhaps a greater willing-
ness to utilize the potential of computers for
art applications may develop.(20, 21)

Computer Graphics: A Historical Review

In the 1940's analogue computers were used
to generate the earliest computer graphics
which were displayed on cathode ray oscil-
loscopes. Ben F. Lapofsky and Herbert W.
Franke were among the pioneers in creating
these images.(4) An early version of a
plotting device was the Henry Drawing Compu-
ter, a modified analogue computer designed
by D.P. Henry. It produced drawings by
combinations of pen movements and table move-
ments.(15)

Later digital computers were used to generate
computer graphics using line printers, plot-
ters and cathode ray tubes as the most common
output devices. Systems combining analogue
and digital components were also used to
produce graphics. Most of these images were
produced by engineers and technicians for
practical purposes. For example, William
Fettner's program created the image of a man
with 7 movable components using data repre-
senting the 50th percentile pilot of the U.
S. Airforce.(4) However, some digital
images were produced for purely aesthetic
purposes, such as "Stained Glass Windows",
a graphic designed by the Army Ballistics
Research Lab.(15)

Some of the most effective of the graphics,

with purely aesthetic intent, were created by the Computer Technique Group of Japan. In any report of computer graphics the work of this group is certain to be included. Their transformations of photographs of President Kennedy and their interpolations such as "Running Cola Becomes Africa" may be considered classic examples of computer art from this period. All of the individuals comprising this group were engineers and programmers. It contained no professional artists. At that time very few people with extensive art training were working to create computer generated images. One of these was Charles Csuri and another was Robert Mallary. Well known examples of their work include Csuri's film "Hummingbirds", his drawing "Sine Curve Man", as well as Mallery's machine tooled sculptures created with the program TRAN2.(4, 9, 11, 15)

Some of the techniques used to create computer art introduced a characteristic look to this medium. For example, geometric graphics generated using equations to describe the form have been used extensively. Some of these closely emulate the Op Art which was popular during the 1960's. Many of these graphics make good use of the computer's ability to do exact and repetitious tasks more easily than humans. Another technique resulting in a different type of form, was the use of stochasticism or randomization in a portion of the program. Similarly, some environmental variable such as movement or sound was recorded electronically and included in the image determining data within a program. These two techniques illustrate the computer capability to generate many forms using one program which includes variables which may be altered at random or with a preconceived pattern in mind.(2, 4, 14, 15)

Another type of form was introduced with the technology which permitted digitizing of the scanned image of a photograph or object in terms of a value scale. This technique alone and combined with interpolation led to the production of a variety of images characteristic of computer art.(4, 8) The use of interpolation between drawn images was also common. Usually these drawn images were introduced to the computer using a light pen as the input device.(4, 15)

In 1968 the first major international exhibition of computer art was held in London. It was called Cybernetic Serendipity. Following this exhibit, more artists began to take an interest in the computer as a creative medium.(15) Currently computer art has taken on an international flavor with work going on simultaneously in many countries of the world. Artists such as Barbadillo,

Sykora, Giorgioni, Bonacic, Leavitt, Bangert and others are now using the computer as a designing or executing device in their work.(1, 2, 9, 16) However, some recent technical developments have not been widely incorporated in the work of artists and remain evident mainly in the province of graphics created by technicians for experimental and practical purposes. Examples of these are three dimensional shaded color graphics and computer generated holograms.(3, 13)

Art Applications and Projections

The attitudes and working approaches of contemporary artists using the computer to assist them in designing and/or executing their work vary considerably. An examination of recent issues of Leonardo or of Ruth Leavitt's recent book, Artist and Computer, will reveal the variety of conceptual modes and technical methodologies which computer artists are using in their work.

Their projected applications are even more revealing. For example, Tony Longson proposes to use the computer to help him create forms to better understand visual perception and the creative process. Edward Ihnatowicz wishes to create responsive Kinetic sculptures in an exploration of the field of artificial intelligence. He proposes to use these to understand cognition through studying the behavior of these artificial systems which would be capable of simulating natural behavior. Charles Csuri suggests that artists could manipulate visual displays of statistical data to express artistic view of reality relating to social problems. In a more conventional vein Patsy Scala talks of creating visual poetry with computer generated video images and Herbert W. Frank expresses interest in creating graphic music. Still other artists continue in an even more conventional mode. They retain the traditional mode of creating art works while using the computer to assist them in design.(9)

Some aestheticians are using the computer to generate images for use in testing the aesthetic response. Some are using computers to analyze statistical data gathered from subject's responses to conventional art works. Computer simulations of the style of non-computer art such as that of Mondrian, Klee, and Hartung suggest experiments to determine what factors are most relevant in determining certain types of responses to works of art.(4, 5)

Museologists and art historians began to tap the potential of the computer for information retrieval and analysis to aid them in studying classifying and caring for museum collections. The programs created by the Museum Information Network have been used by museums for these

purposes.(7, 19)

A few art educators are interested in using computers for instructional and research purposes. Guy Hubbard has attempted to use computers in programmed instruction. Thomas Linehan has used computers to help students understand their own preference styles. I have suggested research applications for three computer capabilities: graphics, statistics and information retrieval.(6, 10)

While a few individuals within the international art community are beginning to sense the potential the computer has for transforming the conventional views about art, I do not believe anyone has projected some of the possible effects this could have on society. For example, what new choices are available to people for use in education, recreation or economic use because of art-related computer applications. Whose responsibility is it to cultivate an awareness of these choices and share it with others? Exploring one illustration may illuminate the nature of these choices.

Moles in his essay "Art, Cybernetics and the Supermarket" noted the potential of introducing a variable into the computer program which results in the magnetic tape which runs machine tools for industry. By these means every item to come from the assembly line could vary slightly, thus giving the customer more choice. The variability would probably be cosmetic in nature, not essentially altering the product purpose or functional form. Thus the choice would be regarded as an example of marginal differentiation.(14)

However systems such as that used by Mallary for producing sculptures, Lourie for producing weavings, or the Synthavision System discussed by Elin could be used by individuals wishing to design and create artifacts such as furniture, fabrics, and prints which would be unique and suitable to their special requirements. If canned programs with many optional branches were used to assist individuals in utilizing this type of system very little computing knowledge would be necessary. Because machine tools operating from magnetic tape output could produce the artifacts, little knowledge of craft processes would be necessary.(7, 9, 18)

As computers are presently used in automation they serve mechanical technology which demands exact repetition for mass production and is best served by heavy centralization of industry. Small systems for individually designed and produced

artifacts would not be bound by the same considerations. With the dropping cost of small computers and their growing versatility it appears that many homes will have several single purpose microprocessors for games or for the control of appliances. This approach to computer application is in the style of mass production and allows the consumer little control over the product except by veto of non-purchase. A small computer which allowed the customer to program many essential aspects of the design of his environment would be more in keeping with the idea that human choice is important in directing the use of computers.

As Duane Palyka notes, "...the versatility of this medium is its ability to handle quite varying devices for input or output. All that is required is that each device have a wire or two containing electrical current which varies within a certain prescribed range. The rest is within the imagination of the individual designing the device."(9) To date drawings, paintings, prints, weavings, and sculptures, have been created using specialized output devices. Responsive kinetic sculptures and responsive environments have also been created existing as output devices. That the public at large could manipulate canned programs and create their own programs to operate such devices does not seem to me to be an unreasonable possibility. In this way computers could allow them to regain control of the artifacts and environments which surround them daily. This seems to me an exciting prospect which has implications for education, recreation, and our economic system.

REFERENCES

1. Bangert, Collette S. & Bangert, Charles J., "Experiences in Making Drawings by Computer and by Hand" Leonardo, vol. 7, p. 289-296. 1974.

2. Bonacic, Vladimeer, "Kinetic Art: Application of Abstract Algebra to Objects with Computer-Controlled Flashing Lights and Sound Combinations" Leonardo, vol. 7, p. 193-200. 1974.

3. Csuri, Charles, "Computer Graphics and Art" Proceedings of the IEEE, vol. 62, no. 4, p. 503-515. 1974.

4. Franke, H.W., Computer Graphics, Computer Art. Phaidon, New York. 1971.

5. Gips, James and Stiny, George, "An Investigation of Algorithmic Aesthetics" Leonardo, vol. 8, p. 213-220. 1975.

6. Jones, Beverly, Computer Applications in Art Education Research unpublished dissertation. 1976.

7. Kranz, Stewart, Science and Technology in the Arts Van Nostrand Reinhold Co., New York. 1974.

8. Knowlton, K. & Harmon, L., "Computer-produced Grey Scales" Computer Graphics and Image Processing, vol. 1, p. 1-20. 1972.

9. Leavitt, Ruth (ed.) Artist and Computer, Creative Computing Press, Morristown, New Jersey. 1976.

10. Linehan, T., "A Computer Graphics System for Visual Preference Detection and Analysis" p. 90-121, unpublished document.

11. Mallary, R., "Computer Sculpture" Art Forum, p. 29. 1974.

12. Molnar, Vera, "Toward Aesthetic Guidelines for Painting with the Aid of a Computer," Leonardo, vol. 8, p. 185-189. 1975.

13. Phillips, J.W., Ransom, P.L., Singleton, R.M., "On the Construction of Holograms and Halftone Pictures with an Ink Flotter," Computer Graphics and Image Processing, vol. 4, p. 200-208. 1975.

14. Reichardt, Jasia (ed.), Cybernetics, Art, and Ideas, New York Graphics Society, New York. 1971.

15. Reichardt, Jasia (ed.), Cybernetic Serendipity, New York-Washington. 1969.

16. Sykora, Zdenek and Blasek, Jaroslav, "Computer-Aided Multi Element Geometrical Abstract Paintings," Leonardo, vol. 13, p. 409-413. 1970.

17. Thompson, Michael, "Computer Art: A Visual Model for the Modular Pictures of Manuel Barbadillo" Leonardo, vol. 5, p. 219-226. 1972.

18. Tuchman, Maurice, Art and Technology, Viking Press. 1971.

19. Vance, David, "Organization Profile: Museum Computer Network, Inc.," Information, p. 157-159, May/June 1975.

20. Weizenbaum, Joseph, "On the Impact of the Computer on Society, Science, p. 609-614, May 12, 1972.

21. Weizenbaum, Joseph, Computer Power and Human Reason, W. H. Freeman & Company, San Francisco. 1976.

MICROPROCESSOR CONTROLLED SYNTHESIZER

Ceasar Castro
295 Surrey Place
Bonita, CA 92002

Allen Heaberlin
5737 Avenida Sanchez
San Diego, CA 92124

Abstract

It does not necessary follow that high quality music synthesizes requires complex hardware. This paper discusses the hardware and software design of a synthesizer which utilizes a standard microprocessor and a relatively simple synthesizer card. Basically the synthesizer hardware is used for the high speed data processing and the software is used for the slower data manipulations. The hardware allows the microprocessor to control the frequency and amplitude of up to 32 tonal channels. Amplitude control provides the means of producing attack and decay envelopes and frequency control provides the means of producing frequency modulation of the output waveform. The synthesizer card has storage space for 16 unique tonal waveforms. These waveforms can be used to emulate different sounds. Their selection is controlled by the processor. The underlining design philosophy of the synthesizer was to tax the software as much as possible and also to give the software as much control as possible. This simplifies the hardware and gives the greatest degree of flexibility.

Background

There are several different approaches which could be used to design a synthesizer. Each has its advantages and disadvantages. As to which is the best depends on the predefined design goals. Before discussing the synthesizer design several different synthesizer methods will be discussed. These alternatives include both classical analog designs and several, new digital techniques.

Analog. This design, usually consisting of oscillators, filters, an other special waveform circuits, has traditionally been used in synthesizer and electronic organ design. These circuits tend to be simple. Usually each circuit performs only one function and no wide data paths are needed as in some digital circuits. A big disadvantage in this approach is that analog circuits, such as oscillators or filters, can not be time shared. Thus to implement a multitone synthesizer many similar circuits must be fabricated. In addition it is difficult to implement programmable

analog circuits. For instance, making an analog oscillator programmable adds significant complexity. If the circuit is not programmable flexibility and adaptability is lost. It would be difficult to incorporate new waveforms. Furthermore precise control of frequency and waveforms is difficult. As the precision increases the design becomes demanding. Oscillator and filter design becomes critical and components must be carefully chosen. In addition analog circuits have dynamic range limitations which are difficult to improve. Digital design can, at least in theory, improve the dynamic range by increasing the word length. To summarize analog circuits tend to be simple but have serious deficiencies when sophisticated performance is required.

Common Divider.[1] This approach is commonly used in electronic organs and uses a common master oscillator, usually around 2 MHz, and generates tonal frequencies by digital dividers. Usually these frequencies are then passed through different analog filters generating various waveforms. There are several companies which produce the divider chips, a significant advantage of this method. Since only frequencies which are divisible into the master oscillator frequency can be produced, resolution is restricted. Thus slightly different frequencies can not be produced. This approach is easy to implement since divider chips are available but the approach is somewhat restrictive.

Digital Harmonic Synthesis.[2] In this design each harmonic is separately generated with the amplitude and frequency specified by a piecewise linear function. The implementation has good potential in duplicating waveforms and should be able to generate almost any desired waveform. This approach requires a amplitude controllable sinewave generator for each tonal harmonic. This is a significant hardware requirement as there easily may be 10 to 15 required harmonics. In addition since the amplitude and frequency for each harmonic must be provided a significant amount of control must

also be provided, probably beyond the capability of common 8 bit microprocessors (6800, 8080, Z-80). Thus this approach offers high performance but unfortunately high hardware and control requirements.

FM Generation.[3] In the FM generator approach a sinewave (or possible other function) is used to modulate the phase of an oscillator. The phase is then converted to a sinewave, usually by table look-up. If the ratio between the modulating frequency and the carrier frequency is an integer a harmonic spectra is generated. By changing the ratio between the frequencies and the "modulation index" different spectrums can be generated. This approach is fundamentally simple as few calculations are required and harmonic rich waveforms can easily be generated. One objection to this method is a somewhat subjective one. Since the spectrum is not controlled directly but rather through a Bessel function it is difficult to relate the arguments to the generated spectrums. Thus it may be difficult to implement a specific tonal waveform if the parameters, modulation index, etc., haven't been obtained. In addition if the waveform produced is not bandlimited to the sampling rate aliasing will occur. This will introduce distortion as spurious frequencies will be generated. The only practical way to correct this problem may be to reduce the spectrum generated by insuring the spectrum does meet the Nyquist criteria. Another, even less attractive option, is to increase the sampling rate. Thus the approach does have potential but does have some pitfalls.

Direct Digital Synthesis (Phase Accumulation). This approach has been discussed in other papers[4] and has been used by the authors.[5] This approach is similar to John Snell's[6] approach except the memory stores the tonal waveforms rather than just a sinewave. In this design (see figure 1) a tonal frequency phase is generated recursively by using an accumulator. A digital word, corresponding to the phase shift between cycles is continuously added in the accumulator obtaining successive tonal phases. The phase is then converted into a waveform using a memory as a lookup table. The phase is the input address of the memory and the memory word is the waveform value. At the memory output the waveform can be scaled. This is done by multiplying the output by a scaling value. By controlling this value, decay and envelopes can be implemented and also tremlo effects. The circuitry can produce many channels by making the accumulator a multiword accumulator memory and adder. Of course control becomes more complicated since the circuitry is shared among different channels. In this case all the

waveforms are summed at the output and this sum is converted to an analog voltage in a digital-to-analog converter.

This approach is entirely digital with much of the tonal generation process easily controlled. The frequency is specified by the word in the frequency control RAM; the waveform produced is the waveform in the RAM and the amplitude envelope specified by another RAM. Thus all of the above can be controlled from a microprocessor merely by the processor writing into the RAM's.

This approach is computationally simple. An addition is required to generate the phase, a memory access to find the waveform value and a multiplication to implement envelope functions. In addition a summation must be performed over all tones. The approach is flexible in that all important parameters can easily be processor controlled. A logical microprocessor function is envelope generation since envelopes tend to change slowly. The full resources of the microprocessor would then be available for this function.
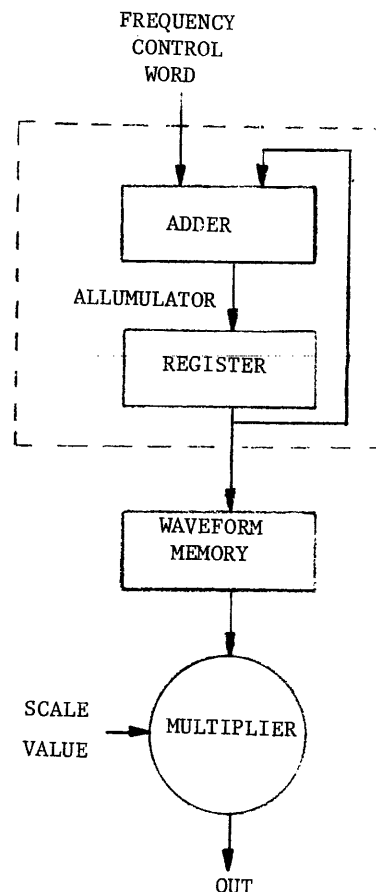
FREQUENCY
CONTROL
WORD

Figure 1. Direct digital synthesis.

Synthesizer Design Goals

The design goal was to allow sophisticated performance yet be of simple design. This was attempted by making the hardware simple and putting much of the complexity in the software. This affords maximum flexibility since software can be updated and changed easily. It also results in simpler hardware. The goals of sophisticated performance and simple design are somewhat exclusive and some compromises had to be made. The compromise was in the number of channels. In the approach taken the complexity is somewhat proportional to the number of channels. Thus to simplify the design a small number of channels was implemented since this will allow lower speed circuitry and some data multiplexing can be used. Furthermore in taking this approach a modular design can be implemented. To gain increased performance more of the identical synthesizer modules can be used to increase the number of channels. Once the design is completed it is easy to duplicate the circuitry.

An important use of the synthesizer was in constructing an electronic organ which sounds like a pipe organ. Basically electronic organs, especially affordable ones, don't sound like pipe organs. Aside from the accoustical environment where the organ is usually located there are several reasons for this. First of all there is usually no chorus effect. This is the simultaneous sounding of several tones of the same or octavely related pitches, each tone sounding at a slightly different frequency.[7] Most organs use a small number of oscillators, often 12, corresponding to the 12 notes in an octave, while pipe organs have the equivalent of thousands. Furthermore, separate manuals often use the same oscillators further aggravating the problem. Another significant problem is that electronic organs are limited in timbre or waveform generation. Usually the waveforms are constructed by passing a signal, such as a sawtooth, through a filter representing the desired sound. Also the filter usually covers 5 octaves. Coupling these two restrictions limits the waveforms that can be generated. Third, the transient effects of pipe organs are rarely incorporated. The pipe organ has definite decay and attack characteristics. If these effects are included they are usually only rudimentarily implemented.

Since the proposed synthesizer was to eliminate as many of the above deficiencies as possible the design must implement the effect of many oscillators giving a "chorus" effect; it must have very flexible waveform generation; and it must provide for attack and decay envelopes.

Initially several constraints were imposed upon the design. Fundamental to the design were performance constraints. First, the synthesizer must be capable of generating at least 25 different tonal frequencies. This figure was considered a lower bound since it is desirable to produce many more. Since there are ten fingers and two feet the maximum number of notes that one can play is 12. Having a minimum of 25 possible tonal frequencies insure that at least two tonal frequencies can be generated for each note. Secondly there should be a minimum of 4 waveforms or tonal sounds generated at one time. Third, there must be control of the envelope or amplitude variations such as tremlo. This control must also be programmable. Fourth, the frequency must be specified to a high degree of accuracy. This will allow close frequencies to be used for the same note giving a chorus effect. This frequency control must also be programmable. Finally the word length must be at least 12 bits giving a SNR (Signal-to-Noise Ratio) of 72 dB. In addition the number of samples specifying the waveform should be at least 1000. This allows waveforms to be accurately specified.

In addition there were important hardware goals. First, no non-standard technology should be used. Emitter coupled logic was not to be used, nor were special purpose chips which are difficult to find. Basically the design should use standard off the shelf IC's such as TTL and standard MOS. Second, the design, especially the controller, should be simple. Since the design was to be done in our spare time difficult trouble shooting problems were to be avoided. In addition a simple design insures simple maintenance. Next, the design should use a reasonable amount of power - about 10 watts maximum. Finally, a medium number of IC's, less than 100, should be used to fabricate the synthesizer.

A critical part of the design is the software. The software was to be designed with certain guidelines. First, to try to keep the synthesizer hardware to a minimum, software was to be used to its fullest extent. This would not only allow for simpler hardware design but potential performance increase. In the future microprocessors will have better instruction sets and will be faster. Since a large part of the processing will be done by the processor there is potential increased performance. In addition the microprocessor should completely control the synthesizer. This would allow for maximum flexibility in modifying and changing the synthesized sounds produced by the system. In essence all the high speed data processing will be done by the synthesizer card and all the data storage, general bookkeeping and control will be done by the microprocessor.

Finally, since the purpose of the synthesizer is to generate musical sounds, some input device to indicate which sound or notes

to produce is necessary. The obvious selection is a piano or organ keyboard. This appears to be the ideal input as the synthesizer is considered an instrument and should be played like one. In addition the interface must be under interrupt control. Sof*. ware polling is not practical since little software time would be devoted to other synthesizer software functions.

The synthesizer design method chosen was the direct digital method described earlier. This offers much in potential performance and also is computational efficient. It further lends itself for integration with a standard 8 bit processor.

Synthesizer Hardware Design

The tone generate part of the synthesizer consists of five fundamental parts: frequency generator, waveform, weighting, accumulation and output, and control (see figure 2). The frequency generator determines the phase at each sample point. This phase, of course depends directly upon the frequency control word (FCW) provided by the processor. The frequency generator recursively generates the newest phase by adding the FCW to the prevoius phase. The phase is then passed to the waveform section. Here the phase is "mapped" into the waveform: from the phase, $\emptyset_{n,k}$, the waveform, $F_k(\emptyset_{n,k},)$, is determined. (n is the sampling number and k is the waveform number.) Then the waveform is passed to the weighting section, where a processor controlled value, W_k, is used to scale the waveform obtaining $W_k F_k(\emptyset_{n,k})$. Finally these values are summed in the summer section obtaining the summed output of each of the 32 tone generators. The output of course is a digital number which is converted into an analog voltage.

As has been described earlier an important goal of the design is complete processor control over tone generation. In the design the processor does have complete control over the synthesizer. The processor can write into all controlling RAM locations. First the processor controls the frequency of each tonal frequency oscillator by writing the frequency control word (FCW) into the frequency generator RAM. Second, the processor is able to select the waveform, f_k, by programming the waveform number in the waveform select ROM. Finally, the processor can select the scaling values, W_k, for tone generation. Since all of the above are processor controlled they can change with time. However if a standard 8 bit processor, such as the 8080, is used updating all tone generators may be limited to approximately once every 2 milliseconds. Another important processor input is the waveform; ($F_k(\emptyset_{n,k})$. By controlling the above parameters and waveforms the processor can

completely specify the tonal waveform.

The interface between the synthesizer and processor is a memory map interface such as is used in the 6800 and the PDP-11. Approximately 128 bytes are required to specify the FCW, waveform number, and weight. Obviously if separate I/O addresses are devoted for each, control I/O space will be several reduced. In addition the waveform memories are not treated as memory space. Each waveform will require a 1K by 12 bit memory. This could require 2K bytes addressing space. With multiple waveforms the waveform memory space can easily become very large. Instead all waveforms are entered through a common port. As the data is entered an internal counter is incremented providing the address for the waveform RAM. Another port (memory address) is used to zero the counter and specify which waveform memory is to be loaded. Thus all waveform data is transferred to the same location. The interface circuitry allows the processor to write into the various RAM's (FCW, waveform number, weight, and waveform) controlling the synthesizer.

The controller is a simple circuit consisting of a ROM and a register. As the ROM is sequenced the control waveforms are generated. All the timing for the synthesizer is derived from this circuitry.
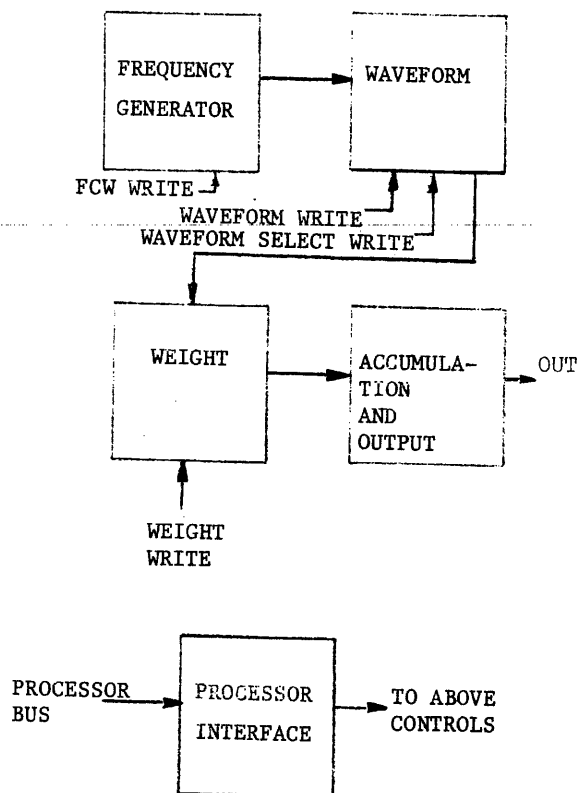
Figure 2. Synthesizer block diagram.

Frequency Generator

The frequency generator (see figure 3) generates the phase for each of the tonal generators. The generator is composed of a frequency control word (FCW) RAM, a phase RAM and an accumulator. All calculations are performed using two 8 bit words. Before tone generation the processor loads the FCW RAM with the two bytes specifying the FCW for each tone generator. This value is added to the previous tone generator's phase in two 8 bit additions obtaining the new sample period phase. The RAM's used have a cycle time of about 250 nsec. Since there must be two memory reads and two memory writes it takes about 1 microsecond to generate the phase. Implementing 32 tonal generators requires about 32 microseconds resulting in an output sampling frequency of about 30 kHz. The most significant 10 bits of the phase is sent to the waveform section.
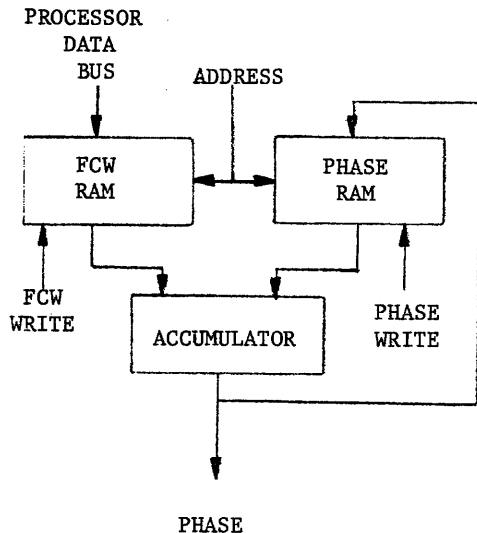
Figure 3. Frequency generator.

Waveform

The waveform section (see figure 4) consists of either a 4K or a 16K word by 12 bit memory. There are three different types of operations using this memory. The primary cycle is the waveform generation cycle. Here the phase from the frequency generator is used as the memory address and the waveform value is read out. The next type of memory cycle is a processor write cycle. In this case the address comes from the internal word counter and the written data comes from the processor data bus. In this cycle the processor loads the waveform memory. The third type of cycle is the refresh cycle. Here a memory read is performed with the address coming from the refresh counter. At the conclusion of this

cycle the referesh counter is updated. This refresh operation occurs every 32 microseconds for the 16K memory.

The memory uses 16 pin 4K or 16K dynamic memory chips. These chips require the address to be multiplexed in conjunction with two separate clocks: row address and column address strobe. These clocks are used in multiplexing the address to the memory chips. Since a memory fetch is required by the synthesizer once every microsecond and the memory has a 0.5 microsecond cycle time a spare cycle is available every microsecond. The spare cycle can be used for either refresh or a processor write cycle. The refresh is given precedence. If the processor attempts to write data and the synthesizer is not ready a memory wait will be asserted until the synthesizer is available and the cycle is completed. It is anticipated the memory maybe loaded under DMA control. This will allow the waveform to be loaded as the note is initiated.
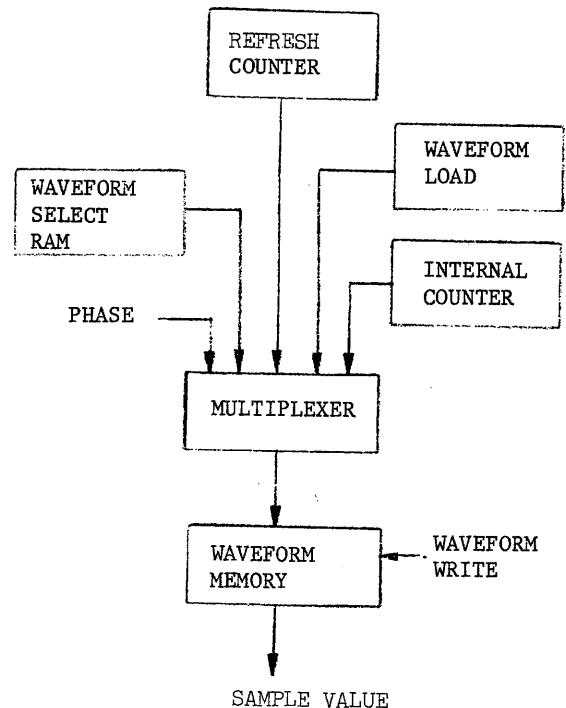
Figure 4. Waveform section.

Weighting

The memory output is sent to the weighting section (see figure 2). First the data is converted from parallel to a 12 bit serial word. Then multiplied in the multiplier (25LS14) and finally summed in a serial adder (25LS15). The serial multiplier and adder have moderate speed and low power. Use of the multiplier results in a 1.0 microsecond multiply time and minimal chip count.

Processor Interface

As far as the processor is concerned the synthesizer is a write only memory. This simplifies the design since no read bus drivers or multiplexers are required. The cycle is started by the processor setting a write request flip-flop asserting the memory wait line. When the synthesizer has an available time slot the write operation is performed. At the conclusion the request flip-flop is cleared and the cycle is completed. There are two types of write cycles. The first is into the parameter RAM's (2101's). These RAM's specify either frequency, waveform number or weighting and have an approximate 250 nsec cycle time. The other RAM cycle is for the dynamic waveform memory. This requires a cycle time of approximately 0.5 microseconds.

Controller

The controller consists of a read only memory (ROM) and a buffer register. This is a very simple controller with the successive ROM address determined from the ROM itself. As the ROM is cycled it generates the various controller signals. Since there is no inputs to the controller, other than the 20 MHz clock, the operation is straightforward. A simplified timing diagram is shown in figure 5. As can be seen each cycle consists of two major timing parts. In generating the phase, first the lower byte is calculated in the first half of the cycle. During the second half the upper byte of the phase is determined. The waveform memory uses this phase during the first half cycle to determine the waveform value. The last half of the cycle is available for memory refresh or for writing new waveforms. The weighting section loads the multiplier when the waveform data becomes available - in the middle of the cycle.

Switch Interface

The switch interface (see figure 6) allows the keyboard to communicate with the processor. Rather than putting switch de-bounce circuitry at each switch a common digital debounce circuit was constructed. This results in much of the complexity of the interface. The counter continuously counts generating addresses corresponding to different switches. At each address a RAM is used to remember the status of a switch - essentially how long it has been on or off. To understand the operation first assume the switch is off. At this time the RAM will have zero stored for that switch. As the switch is turned on the status count will increase each time the counter reads the switch. Finally a switch "on" threshold is reached and that switch is considered on. The operation is similar when the switch becomes off. Hysteresis has been added, by having the threshold when the switch is off greater than when the switch is on, to eliminate contact bounce. If
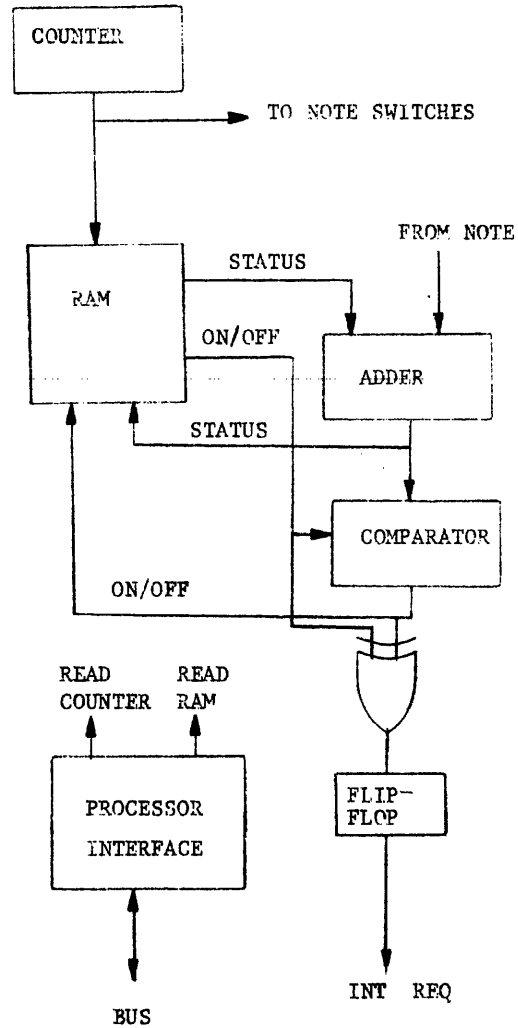
Figure 6. Switch interface.

SYNTHESIZER CYCLE				
PHASE (LOWER BYTE)		PHASE (UPPER BYTE)		PHASE
PHASE MEMORY READ	PHASE MEMORY **WRITE**	PHASE MEMORY READ	PHASE MEMORY WRITE	GENERATION
WAVEFORM READ CYCLE		REFRESH OR PROCESSOR WRITE		WAVEFORM GENERATION

Figure 5. Synthesizer timing.

a switch change is detected, that is, the switch becomes on or off, an interrupt request flip-flop is set. This also stops the counter. The processor responds to the interrupt by first reading the counter to determine which key (switch) is changing. Then the processor uses the address to read the RAM and determine the switch status. This information is used by the software to control the synthesizer and will be discussed further in the software section.

This may seem excessively complicated but it is much simpler than having a hundred or so debounce circuits. In addition the same switch interface can be used to interface the waveform selection switches - stop selectors on an organ.

Software

Introduction. The software to control the synthesizer and achieve the synthesis of a note can be broken down into four parts. The first parts deals with initializing the system such as setting up tables, pointers and counters. The remaining three parts deal with the start of a note, the synthesis of the note and termination of the note. The software does a great deal of data processing. The following paragraphs discusses the software in more detail in the hopes that the reader will get a better feel for the extent of the software.

Initialization. The software has been written to accommodate 128 different notes (forty more than a piano keyboard and six more than two organ keyboards). Information on each note must be stored, so that when the note is played the software can fetch the information from a note table. Table 1 shows the information that is stored for each note. The first entry, the timbre number, tells the software which output waveform the software should use to generate the given note. At present there are four output waveforms stored in the synthesizer card. The attack rate is used to determine how long the attack period will last. An eight bit register, the attack time, is summed with the attack rate every clock period. The clock period occurs every 2 milliseconds. When the software detects a carry from the above addition, the software will change the state of the note from the attack state to the steady state. For example, if the attack rate is set at 4, the attack period will last 64 clock periods or 128 milliseconds. Each note has its own rate so the attack period can be different for every note.

The third entry in the note array is the attack envelope number. At present there are four attack envelope tables stored in software.

The attack envelope number indicates which envelope the software should use for the given note.

TABLE 1

Note Table

Each Entry (128 Notes) Includes the Following Elements:

    Timbre number
    Attack rate
    Attack envelope number
    Decay rate
    Decay envelope number
    Steady state rate
    Steady state envelope number
    Frequency modulation rate
    Frequency modulation envelope number

The note table also contains the information for the decay and steady state period. This information is used in the same manner as the attack information. However, there is one difference with the steady state rate. The software never tests the carry. This means that once the note is in the steady state it will remain in that state, except for one condition; when the software has detected that a key is no longer being played, the software will change the state of that note to the decay state and thus the note will be terminated at the end of the decay period.

The final two entries in the note table are the frequency modulation information. This information would be used to modulate the output waveform. However, at present the software has not been written to implement frequency modulation.

Besides the note table there is a status active table. The synthesizer can generate up to 32 notes simultaneously. The status active table contains information on only those notes that are being generated. The number of entries in the status active table depends on the number of notes that are simultaneously being played. If no notes are being played, then of course there would be no entries in the status active table. The information stored in the table for each active note is shown on table 2. There are fourteen parameters listed for each note. The note number indicates which note from the note table is being generated. The timbre indicates which output waveform should be used to generate the note. The attack rate, attack envelope, decay rate, decay envelope, stready state rate, steady state envelope, frequency modulation rate and frequency modulation envelope is the same information that is stored in the note table. This information is transferred to the status active table when a note becomes active. The Fre-

quency Control Word (FCW) is directly related to the note number. The FCW is the actual information feed to the synthesizer to indicate tonal frequency produced. Following the FCW in the table is the FCW synthesizer address. This address is the location the FCW is stored in the synthesizer. The next entry, amplitude synthesizer address, is the location the envelope amplitude data is stored in the synthesizer. Finally, the timbre number address, this address is the location the timbre number is stored in the synthesizer. The initilization software sets up the pointers to the status active table and clears the status active counter to indicate that all 32 note generators are available.

TABLE 2

Status Active Table

This table contains status information on all active notes. Up to 32 entries are possible with each entry composed of the following:

    Note number
    Timbre
    Attack rate
    Attack envelope
    Decay rate
    Decay envelope
    Steady state rate
    Steady state envelope
    Frequency modulation rate
    Frequency modulation envelope
    Frequency Control Word (FCW)
    FCW synthesizer address
    Amplitude synthesizer address
    Timbre number synthesizer address

The initialization software also has the duty of setting up the timbre waveform in the synthesizer. The synthesizer card can hold four timbre waveforms. Each waveform consists of 1K 12 bit words which breaks down to 1.5K bytes. The waveforms are stored on cassette. The software transfers the data from the cassette to the synthesizer card. There is no restriction on the waveforms. They may be a simple sine wave or a complex waveform. This freedom provides the means of emulating instrument sounds, since the instrument's waveform can be stored in the synthesizer.

As mentioned, a note can be in one of three states: attack, steady state or decay. A state table is used to indicate which state a note is in. There is a maximum of 32 entries in the state table. Table 3 shows the information contained for each entry.

TABLE 3

State Table

This is the operating array for each active note. Each entry has the following elements.

    State
    Time
    Rate
    Envelope base address
    Amplitude synthesizer address
    Note number

The first entry is the state. This entry indicates which state the note is in. The second entry is the time. This entry indicates how long the note has been in the given state. Time is also a pointer to the location in the envelope table the note is presently using. The location contains the amplitude datum for the output waveform. Every clock period the rate is summed with time to give a new time and a new address for amplitude datum. If a carry is detected the software will change the state of the note to the next state. However, there is one problem, you do not want to leave the steady state and start the decay until the note has been released. To prevent this from occurring the software does not test the carry from the steady state time and rate summation. Thus the note will not go into the decay state until the software detects that the note has been released and chages the state of the note.

The next entry in the table is the envelope base address. This is the address of the first entry of the envelope table for the given state. By adding time with the base address, the location of amplitude datum is determined. This datum is loaded into the synthesizer amplitude address.

The final storage table is the note frequency table. This table contains the frequency control word for each note. Each FCW consists of two bytes and since there are 128 notes, the table occupies 256 bytes. The initilization software loads the proper FCW for each note.

Besides the above tables, the initialization software sets up two stacks. One stack contains the addresses of the available generators. The software loads the 32 addresses of the status active table. The second stack contains the addresses of the generators that are active. Since there are no active generators to start off, the software clears this stack and and clears the counter which indicates how many notes are active.

The final requirement of the initialization software is to set up pointers to the starting addresses of the above tables and to set up the numerous counters that are used.

Note Detection. As mentioned earlier the software excluding initialization can be broken down into three sections: note detection, clock detection and note deletion. The note detection software is used when a note is first detected. When a note is first played, the keyboard hardware detects the note and causes an interrupt. The interrupt vectors the processor to the note detection software. The first thing the software does is obtain the note number from the keyboard hardware. Using the note number the software determines the Frequency Control Word and loads it into the status active array. Next the note parameters are moved from the note table to the status active array. The synthesizer addresses are moved to the status active table. The state table is set up with the attack parameters. The Frequency Control Word and timbre number are loaded into the synthesizer. The output amplitude is set to zero. The generator available pointer is incremented to point at the next available generator. If no generators are available, the generator full flag is set.

The note detect software does not start the synthesis of a note but merely sets up the data so that it may begin. It is during the clock interrupt software that a output from the synthesizer is generated.

Clock Detect. Every 2 milliseconds the software is interrupted and vectored to the clock detect software. The software will then cycle through the active notes. For each note the time datum and rate are summed to give the address of the amplitude data. If a carry is detected the software will change the state of the note. The amplitude data is transferred to the synthesizer card. If the software detects the end of the decay state then the software will delete the note from the status active table and the state table. It will decrement the generate available count to indicate that a generator has been freed.

Note Delete. When the keyboard hardware detects the end of a note, an interrupt is sent to the processor. This interrupt cause the processor to vector to the note delete software. The software will obtain the note number from the hardware. It will go to the state array and change the state of the note to decay state. The clock detect software detects the end of the decay state and does the necessary bookkeeping as mentioned above.

Conclusions

The synthesizer described in this paper is intended to provide high quality musical synthesis. Up to 32 simultaneous notes can be synthesized. The attack, decay and steady state amplitude of each note can be individual controlled. The flexibility of the design is do to the fact that much of the processing and control is in software.

References

1. R. B. Cotton, "Tempered Scale Generation From a Single Frequency Source," Journal of the Audio Engineering Society, Vol. 20, pp. 376-382, June 1972.

2. J. A. Moorer, "Signal Processing Aspects of Computer Music: A Survey," Proceeding of the IEEE, vol. 65, no. 4, pp. 1108-1137, Aug 1977.

3. J. M Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," Journal of the Audio Engineering Society, Vol. 21, no. 7, September 1973.

4. J. Tierney, C. M. Rader and B. Gold, "A Digital Frequency Synthesizer," IEEE Trans. Audio Electroaccoust., vol. AU-19, pp. 48-56, March 1971.

5. A. Heaberlin, U.S. Patent No. 4,003,003, "Multichannel Digital Synthesizer and Modulator," January 11, 1977.

6. J. Snell, "Design of a Digital Oscillator Which Will Generate up to 256 Low Distortion Sine Waves in Real Time," Computer Music Journal, vol. 1, no. 2, pp. 4-25,1977

7. Richard H. Dorf, "Electronic Musical Instruments," Radiofile, New York, 1968.

8. A. Popoulis, "The Fourier Integral and Its Applications," McGraw-Hill, New York, 1962.

APPENDIX

MATHEMATICAL ANALYSIS OF TONE GENERATION

To give those who are mathematically inclined a better feel of the mechanics of the synthesizer, the following is a brief mathematical treatment of the synthesizer tone generation. The section consists of two parts. The first describes a mathematical representation of the tone generation process. The other section describes the errors introduced into the tone generation and shows how they can be determined.

Assume we are interested in generating a periodic waveform, $f_p(t)$, which has period, T_p. In addition let us generate a function $f_s(\emptyset)$ corresponding to a single cycle of the waveform but with period, 1:

$$f_s (t/T_p) = f_p(t) \text{ for } 0 \le t < T_p.$$

In generating this waveform we will generate a sampled representation of the signal with sample period, T_s. Thus we are only interested in $f_p(t)$ at discrete points, specifically:

$$t = nT_s \text{ for } -\infty < n < \infty .$$

Since the waveform is periodic we are only interested in the "phase," not the integer number of cycles. This "phase," normalized between 0 and 1, has zero representing the start of the cycle and one representing the end of the cycle.

We can define a remainder function, REM, which gives us the normalized phase. Mathematical we first separate nT_s into the integer number of cycles, K, and the phase at the nth sample period ϕ_n.

$$nT_s = KT_p + \phi_n T_p$$

This is done by finding K, the integer part of (nT_s/T_p). Then

$$\phi_n = (nT_s - KT_p)/T_p$$

$$\text{or}$$

$$\phi_n = \text{REM} (nT_s/T_p).$$

Thus

$$f_s (\phi_n) = f_p(nT_s)$$

We want a recursive method of generating $f_p(nT_s)$ or equivalent $f_s(\phi_n)$. We can generate $f_s(\phi_n)$ by generating ϕ_n recursively. We can generate ϕ_{n+1} from ϕ_n:

$$\begin{aligned}
\phi_{n+1} &= \text{REM} (nT_s/T_p + T_s/T_p) \\
&= \text{REM} (k + \phi_n + T_s/T_p) \\
&= \text{REM} (\phi_n + T_s/T_p) \qquad (1)
\end{aligned}$$

Thus we can recursively generate $f_s(\phi_n)$ or equivalently $f_p(nT_s)$ by recursively generating ϕ_n from (1) and then determine $f_s(\phi_n)$ from ϕ_n.

Equation (1) is easily implemented by using a digital register representing ϕ_n, and an adder. The remainder function is easily implemented by truncating the register output. This register and adder are often called the phase accumulation register. At each sample period T_s/T_p is added to the previous phase detaining the new phase. This phase, ϕ_n, is then converted into $f_s(\phi_n)$ by table lookup. A RAM or ROM is used to represent $t_s(\phi_n)$.

Error Analysis

If $f_s(\phi_n) = f_p(nT_s)$ with $\phi_n = \text{REM}(nT_s/T_p)$ for all nT_s then there will be no distortion provided, of course, the Nyquist criteria is meet: the highest frequency of $f_p(t)$ must be less than one half the sampling frequency, $1/T_s$.

However $f_p(nT_s)$ will not equal $f_s(\phi_n)$ for two reasons. First the waveform $f_s(\phi_n)$ will have finite entries and will appear as in figure 7. This is because $f_s(\phi_n)$ is constant between entries: in practice the phase is truncated and the next lowest phase, say $\phi_{\bar{n}}$, is is used to approximate $f_s(\phi_n)$. Thus $f_s(\phi_{\bar{n}})$ is generated - not $f_p(nT_s)$ and we can analyze the spectral content of this waveform to determine the distortion. The above introduces error when the phase has more resolution than the table used to represent $f_s(\phi_n)$. Of course if the phase, ϕ_n, is not truncated than there are no errors introduced. But typically the phase is generated with more resolution and will be truncated. The other source of error is quantization error. Since the word length used to specify the sample value is finite, errors are introduced. This results in white noise (constant noise power density as a function of frequency) and can be controlled by making the word length long enough. 12 bits gives a signal-to-noise ratio of 72 dB and is probably adequate. First the spectrum of $f_s(\phi_n)$ approximating the waveform $f_p(t)$ will be determined. A single cycle of the periodic waveform is sampled for storage in the table or memory (see figure 7). The sampling interval is the period divided by N, the number of entries in the waveform table. The waveform is sampled at each point and then this value is held constant until the next point (flat-top reconstruction). Thus the waveform in the memory, f_m, has the following representation.

$$f_m(t) = \sum_{n=0}^{N-1} h(t-n/N) \, f_s (n/N)$$

where

$$h(t) = \begin{cases} 1 & \text{for } 0 \le t < 1/N \\ 0 & \text{otherwise} \end{cases}$$

The tone generation process essentially determines a phase and then uses this to lookup the waveform value in the memory. This waveform determining process can be separated into two equivalent processes. First the waveform will be produced from the ROM at the correct frequency with infinite precision. Next this signal will be sampled at the sampling rate, $1/T_s$. This is identical to using the phase directly to determine the waveform value.
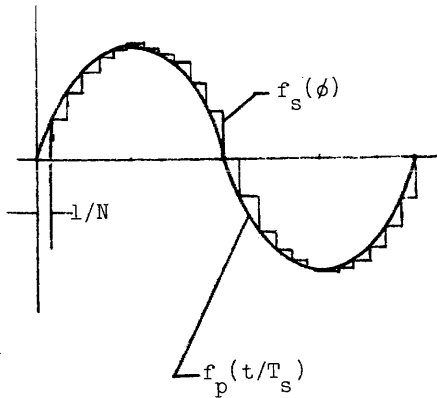
Figure 7. Memory waveform, $f_s(\phi)$.

Figure 8. Spectrum of memory waveform (sinewave)

First the waveform from the ROM at the correct frequency will be a periodic signal identifical to a single cycle of the memory waveform with frequency $1/(T_S/T_p)$. This waveform has the following representation:

$$f(t) = \sum_{h=-\infty}^{\infty} h((t-n/N)T_p) \, f_w(nT_s)$$

Here f_w is a periodic signal with $f(nT_s) = f_s(REM(nT_s/T_p))$. This has the following spectrum (see figure 8):

$$F(\omega) = \sum_{n=-\infty}^{\infty} F_w(\omega - n\omega_s) \, H\left(\frac{\omega}{T_p}\right)$$

Obviously the waveform is not bandlimited. There are abrupt transistions between entry points. This waveform will then be sampled at the sampling rate, T_s. Let us further assume the signal we are generating is a sinewave as has been represented in figures 7-8. This isn't necessary for error analysis but it does allow us to separate harmonic error from other errors. Essentially all spurious frequencies above ½ the sampling rate are translated into the region $-T_S/2$ to $T_S/2$ (see figure 9). Thus we can separate the error into two parts: harmonic and non harmonic. All harmonic error less than Nyquist bandwidth will remain undistorted. Since this error is harmonic it will not be as important as the non-harmonic error and will be disregarded here. However frequencies above the Nqyuist bandwidth will appear as spurs. The waveform to be generated can be separated into harmonics or partials. The fundamental will have one cycle stored in the memory thus N will be 1024 because all memory locations are used to specify the cycle. However for the other partials more cycles will be present in the memory and the effective N will different. For instance, for the first harmonic two cycles will be represented in the memory. Therefore the effective N is 512. A procedure for de-

Figure 9. Spectrum of generated signal.

termining the non-harmonic errors is as follows: For each partial, determine the effective N. This will be 1024 divided by the harmonic number plus 1. Next determine the first frequency above the Nyquist bandwidth as above. This will be the strongest spur and all frequencies above this will be spurious. Repeat this for each partial in the generated signal. In addition the spurs are proportional to the strength of the corresponding partial. Two possible approaches to determine allowable error could be considered here. One might be to consider just the maximum spur. But probably a better error function is to determine the total spurious error. This will be a power summation over all spurs and over all partials. Using this for a particular waveform and frequency the signal-to-noise ratio can be determined.

"DESIGNING YOUR OWN REAL-TIME TOOLS
A MICROPROCESSOR-BASED STEREO AUDIO SPECTRUM ANALYZER FOR
RECORDING STUDIOS, ELECTRONIC MUSIC, AND SPEECH RECOGNITION."

Byron D. Wagner
1701 Viewmont Dr.
Los Angeles, Ca. 90069
213-982-6200

Abstract

This paper describes the relative ease with which an inexpensive personal computer can be configured into a highly sophisticated, personalized tool or test instrument with performance equaling or exceeding that of costly, commercially available, less flexible equipment. The author details the design, construction, software development, smoke-testing and calibration of such a device - a real-time, 1/3 octave audio spectrum analyzer using a color television as a graphic display screen. Applications for this particular device are discussed and guidelines are offered for customizing systems to the users needs, either as a standalone instrument or as an automated controller to be integrated into an existing unit. Examples suggested include: mixing consoles, theatrical lighting boards, and a scanning electron microscope.

Tools

According to some scientists, the ability to create artificial extensions of fingers, specialized for different purposes, is what originally separated cavemen from the animals. History is split into periods representative of the materials used to form the tools, ie. the Stone Age, the Bronze Age. As man grew more sophisticated, so did his tools. Bits of sharpened rock and bone for cutting and grinding became knives and spear points, later arrows. With the invention of the wheel, man was no longer totally dependent on beasts of burden, and transportation became more efficient.

By the Middle Ages, technology and craftsmanship had raised the original utilitarian cutting tool to the level of being a powerful symbol. Ornately jeweled swords and scepters conveyed authority, items of ritual never put to practical use. For the most part, how-

ever, the advance of technology was not an end in itself, but a means of gaining leverage over materials used in making the physical world a better and safer place to live. The fact that people could derive pleasure from the esthetic challenges of organization is displayed in the creative patterns commonly found in early pottery and weaving, with even the most common household items crossing the line between strict functionality and the communication that is art.

When I was young, my father took me to visit a craftsman who made his living by repairing musical instruments, expecially brass horns. As I was admiring the amazing assortment of metal-working implements, I asked where someone might buy such unusual, specialized equipment. "I wouldn't know", he replied, "Imake my own myself", and he smiled with pride.

People have always been envious of such highly personalized and customized tools. From the gunfighter's pearl-handled revolver (with the hammer spur filed off to facilitate quick draws), to the pool hustler's weighted and balanced pool cue, or the professional musician's instrument, voiced just to his liking. In each case, the modifications are justifiably practical necessities for the pro, but luxuries to the layman (novice).

Personal Computers - Getting Beyond "Other People's Programs"

The personal computer provides the opportunity to create devices whose complexity and degree of personalization is limited solely by the imagination and motivation of the user. It is an accessible alternative for the solution of problems. In other words, one can trade an abundance of time for a nonabundance of money. With the additional advantage that, once a written logic framework or program is established, it can be easily adapted for later uses. Organizing information utilizes the same techniques whether the data manipulated

represents a stamp collection, phono-
graph records, recipes, ham radio con-
tacts, or an amateur astronomer's
sighting coordinates. Even at this
point on the mass memory price curve,
large system data base storage and
retrieval techniques can be scaled
down and applied to home systems.
The employment of reference pointers
such as directories or indices is al-
most as useful as storing the refer-
ence works themselves and offers a
substantial savings in equipment
cost.

Even though the mass marketplace
for personal computing products is
still relatively small in relation
to other hobbies, the array of
existing building blocks is pheno-
menal. Eight-and sixteen-bit CPU's
cheap RAM and ROM memory, inexpensive
tape and disc storage, terminals,
quality hard-copy printers, medium
resolution black-and-white or color
displays, speech generation and
recognition devices, communication
modems and touch-tone transceivers,
remote control of household appli-
ances using AC lines, music syn-
thesizers, analog to digital and
digital to analog converters, digital
logic analyzers, digital frequency
counter and digital multimeter sub-
systems, realtime clocks, and even
medium resolution video digitizers
exist, most compatible on a "plugin
and go" basis. Yet perhaps a greater
benefit is the willingness on the
part of most manufacturers to pro-
vide not only prewritten software,
but also technical support. This
includes schematic diagrams and
"theory of operation" information
on a nuts and bolts level (as
opposed to the traditional "once
you buy it we don't want to hear
from you" supplier-consumer re-
lationship.)

Real Time

Part of the precision with which
we operate our hands and fingers
stems from the fact that they are
able to send feedback signals to the
brain that controls their movements.
This is a classic example of a
closed-loop servo system (as oppo-
sed to blindly applied brute force.)
The system can be disturbed, though,
if a delay is introduced either
between the sensing apparatus and
the controller, or the controller
and the actuators. With children

or adults, the more quickly rein-
forcement follows action, the
faster the learning process is
accomplished. This builds the pattern
of challenge, motivation, and reward
into a constructive "vicious circle."
A significant benefit of this behavior
is that more good ideas will be follow-
ed through to completion, instead of
being abandoned because of petty but
time-consuming obstacles. Studies in
human ergonometric design confirm
practical experience relative to the
speed and accuracy with which a person
can assimilate raw data. Computer hex
codes are harder to understand than
conversational English, which in turn
is more difficult to read than small
strings of numbers. The simplest way
to communicate rapidly comprehensible
information reflecting changing signal
parameters is through the use of analog
position indicator movements referenc-
ed to a fixed linear or logarithmic
scale. Convention in the past dictated
that separate functions required sep-
arate displays, whether visually
oriented (meters and gauges) or audio
(buzzers, bells, and chimes). But
experience has shown that human beings
are capable of deciphering numerous,
although subtle, details from complex
integrated or holistic sets of stimuli.
An experienced mechanic listening to a
car's engine can diagnose many specific
different ailments relating to carbu-
ration, valve, timing, and mechanical
balance problems in much the same way
as a physician examines a patient.

An example of this type of design
applied to electronic test instruments
may be found in many recording studios
and even some home hi-fi systems.
When a standard oscilloscope is fed
stereo program material, with the left
channel going to vertical drive cir-
cuits and the right channel feeding
horizontal dirve circuits, a complex
"scrambled egg" pattern appears.
This pattern simultaneously shows peak
amplitude, stereo separation, relative
phase, frequency, and dynamic envelope
characteristics for both channels.

It has become feasible to design
instantly reconfigurable output inter-
faces with ultimte flexibility without
giving up the capability of providing
the ultimate in specialization and
detail. Such devices can be person-
alized to the idiosyncracies of an
individual (whether the user is left-
handed, speaks a foreigh language,
is male, female, or even a child or

animal), with different pre-programmed
levels of complexity. Obviously, this
is totally opposite to traditional
instrument design practices and does
offer disadvantages. These include
the lack of standardization of oper-
ation and maintenance procedures.
However, standards will probably emerge.
Hopefully, they will be generated on
the basis of need and logic, instead
of manufacturing expediency and
economy.

Obviously, a powerful systems
analysis tool such as we have been
describing, when used as the feedback
element for a process-control device,
is capable of creating a layered
structure suitable for complete auto-
mation or independend action. Yet
manual assistance, supervision, or
even complete override by a human is
possible while still retaining the
housekeeping functions and automatic
trim logic to allow the operator as
much or as little responsibility as
desired. A good example of this
would be an autopilot on a plane that
could coordinate turns, even though
the direction heading and other gui-
dance functions had been disabled by
the pilot.

A Real-World Example

In the field of professional audio,
historically, control of signals has
been limited primarily to the mani-
pulation of the amplitude of informat-
ion derived from real-world sources.
Typically, these are musical instru-
ments, voices, or in the case of motion
pictures, and television, sound effects
or background sounds. In recent years
this has been expanded to include equal-
ization (still a function of amplitude),
artifical reverberation and echo, multi-
track recording, phasing and digital
delay lines (time), and most recently,
pitch or frequency (made possible by
the use of analog to digital and digital
to analog conversions). Attempts at
either the direct synthesis or modifi-
cations of complex waveforms were limi-
ted for the most part to electro-mech-
anical devices (Hammond organ tone
wheels and Leslie rotating loudspeakers)
or prohibitavely time-or money- consum-
ing procedures using large-scale com-
puters (Music V). Additionally, the
computer programs did not operate
interactively in real-time. This state
of affairs explains why commonly used
indicators throughout the industry in
broadcast stations, recording studios,

and communication networks were
generally VU or Peak Program meters,
to be used in conjunction with loud-
speakers and sometimes an occasional
oscilloscope.

Things, however, are changing fast.
The changes result from such factors
as the consumerization and mass-
marketing of voltage-controlled
synthesizers like the Moog and ARP,
the skyrocketing technology of large-
scale integration, and the continu-
ing demand for more control and real-
ism in audio recording and sound
reinforcement. These changes also
brought about the need for more
sophisticated measurement tools at
affordable prices. The ability for
a human to tell the difference between
musical instruments, for example, a
flute or a clarinet, is due to the
difference in harmonic structures
and dynamic envelope changes that
occur during the duration of a note.
These are determined by the mechanics
of vibration peculiar to the gener-
ation of the sound emitted by that
particular instrument. In the case
of the flute, a vibrating column of
air; in a clarinet, vibrating reeds

In response to this demand, re-
searchers developed tools like the
spectrograph, a device which splits
audio signals occurring in the
spectrum between twenty and twenty
thousand vibrations per second into
bands, much as a prism devides white
light into its component parts. This
was accomplished by passing complex
signals through a turnable filter,
plotting the results, and repeating
in an overlay fashion after retun-
ing the filter. This yields a result
similar to the graphs popularly
known as "voice-prints". Early
attempts at real-time analysis were
cumbersome due to the number of
filters and meters necessary for
reasonably discrete identification
of separate harmonics. If the spec-
trum is divided into octave bands,
the required number of filters is ten.
If half-octave, nineteen; third-
octave, twnety-seven. That meant
dozens of tubes and lots of heat.
In early 1971, Hewlett-Packard in con-
junction with Altec-Lansing, intro-
duced an audio spectrum analyzer in
a relatively small, rack-mount case
with a CRT display, selling at the
breakthrough price of approximately
$3,000. This was followed a few
years later by a unit from Amber

Electro Designs of Montreal. It features an LED matrix display, with dimensions of ten by ten, and retails for about $2,000. (See Figure #1) The Ivie Corporation, based in Utah, is now marketing a palm-sized, hand-held, audio spectrum analyzer with built-in LED matrix display and calibrated microphone. This sells for less than half a thousand dollars.

As the availability of such devices grew, so did their uses. eg.: the study of acoustics and musical instrument tone generation, bandpass characteristics of amplifiers, tape machines, and so on, and the practice of tuning or "voicing" music reproduction or sound reinforcement systems using graphic equalizers to compensate for room characteristics, and displays of power bandwidth in disc mastering chains.

The Hardware

In the course of designing a professional recording facility with built-in video tie-lines connecting the performing, control-room and playback areas, the idea of including such an analyzer with the ability to distribute a display to the video monitor in the control room (saving valuable space) coupled with the possibility of providing the musicians with immediate feedback corresponding to their dynamics, proved irresistable. Unfortunately, none of the commercially available units offered such a capability without the need for some kind of scan conversion, for compatibility with NTSC video signal standards. A little investigation and guesstimating led to a potentially feasible design for accomplising not only these goals, but a host of others: color display, programmable overload threshold, with a provision for matching existing disk cutterhead curves, the ability to freeze a frame of display, or display frozen frames sequentially for a slow-motion effect (in forward and reverse) to facilitate the analysis of signals for music and voice synthesis and recognition, and the ability to display a large number of fullband signals from multiple-channel tape machines or mixing consoles, to eliminate the head swiveling (and resulting neckache) from trying to read thirty-two VU meters at the same time. One of

the most attractive points was that all the hardware, with the exception of filter banks and input signal conditioning circuitry, was available off the shelf. Final choices included : an IMSAI mainframe with front panel and CPU, Seals and IMS 8K memory cards, a Sony color monitor, a Bytesaver card, three D+7A's (analog to digital converters) and a Dazzler color video display interface, all from Cromemco. (Fig. #2)

The display format chosen was that of vertical, multi-colored bars, whose heights rise and fall with the output of the corresponding filter/peak detector combination. The basic algorithm for translating analog input amplitude changes to corresponding changes in bar-graph height was developed, and a test program written in basic, along with a routine that erased the screen at the onset of operation. Simply stated, a scanning pattern is established and, for a particular bar, a value is input and compared with the present position on the screen. If the screen value is greater than the input value, a block of color is written at that position. If not, the background color, in this case black, is written into the position. (See Fig. #3) With the test program running under Altair, 3.1 Basic at standard processor speed, it took approximately twenty seconds just to write one screen featuring 8 bars. Since the minimum update speed necessary for smooth and natural stepless transitions between frames is about a thirtieth of a second, it was both necessary and desirable to implement the program in machine language. An early incarnation of the assembly listing for such a machine language program (still displaying only 8 bars) is shown in Fig. #4.

Study will reveal the relatively crude nature of the counting loops and the fact that the entire bar is repainted each scan. The quadrant positioning and jumping arithmetic was made necessary by the architecture of the display used - a Cromemco Dazzler (high resolution color mode). Any memory mapped video display would work equally well. The software was expanded and further refined to allow interaction with simple switches so that a terminal would not be necess-

ary in the case of dedicated operation. Hardware bugs were chased down and exterminated and PROMS containing the final version of software were burned. The device was calibrated using the time-honored tradition of applying signals of known frequency and amplitude while making the corresponding necessary adjustments for proper indication.

The resulting display is a tremendously useful device - although the beauty of its fascinating patterns set to music may lead the more cyncial to cast aspersions on the purity of pragmatic intent of its creator. By plugging in other types of sensors, eg.: thermocouples, optical position encoders, anemometers, medical electrodes; and changing the reference scale and label overlay data, virtually any type of measurement parameters can be accomodated. The dynamic range, resolution, response time and ballistics, averaging criteria, linearizarion or log scaling and weighting, can all be defined by software. With the addition of a modem, remote polling and data acquisition become possible; with a mass storage device (disc, tape, bubble memory) the opportunity for extensive signal analysis and hard copy plots. Yet with all this flexibility, the system can satisfy the most subtle demand for highly specialized applications. If time permits, the author will discuss interfaces with control elements in theatrical lighting systems and the automation of recording studio mixing consoles and electronic music synthesizers as well as a scanning electron microscope function control and stage positioner.

Conclusions

In view of the current mindboggling leaps in technology and the complexity of available hardware, it is comforting to consider that, far from being closed, the available options for designing and molding a personalized, custom-fitted set of tools with which to create and communicate are as rewarding and open as in the past, if not more so.

Acknowledgements

The author would like to thank all those who contributed to the success of the project with suggestions or construction assistance. Notably: Mr. Rick George, Mr. Mike Ronstadt, Mr. Gale Lester, as well as Ms Sally Grimm and Ms Annie Moss whose typing tootsies made this paper possible, and a special thanks to Mr. Allen Immerman for his fearless and tireless help in toggling the beast into submission.

About The Author

Byron D. Wagner, 26, is originally from Omaha, Nebraska. He is currently employed in the Los Angeles area as an independent record producer and recording engineer - in addition to functioning as a consulting engineer for recording studio design, construction, and installation. Past and current clients include: Carole King, Linda Ronstadt, Peter Asher, Music Recorders, Inc., Motown Records, Ike & Tina Turner, Steve McQueen and Ali McGraw. His professional affiliations include: The National Academy of Recording Arts and Sciences, The Audio Engineering Society, the SMPTE, the Magic Castle, and the American Federation of Television and Radio Artists. His still photography has been featured on several album covers and he is the host of the PBS television program "Singer/ Songwriter". His education was assembled through such diverse facilities as the Omaha public school system, Ohio University, Brigham Young University, and the Eastman School of Music. (And he wants to be a movie star when he grows up.)
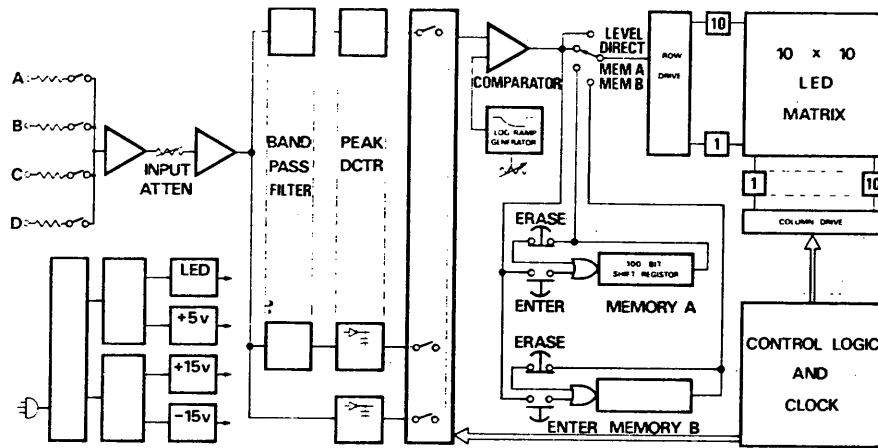
BLOCK DIAGRAM

FIGURE 1
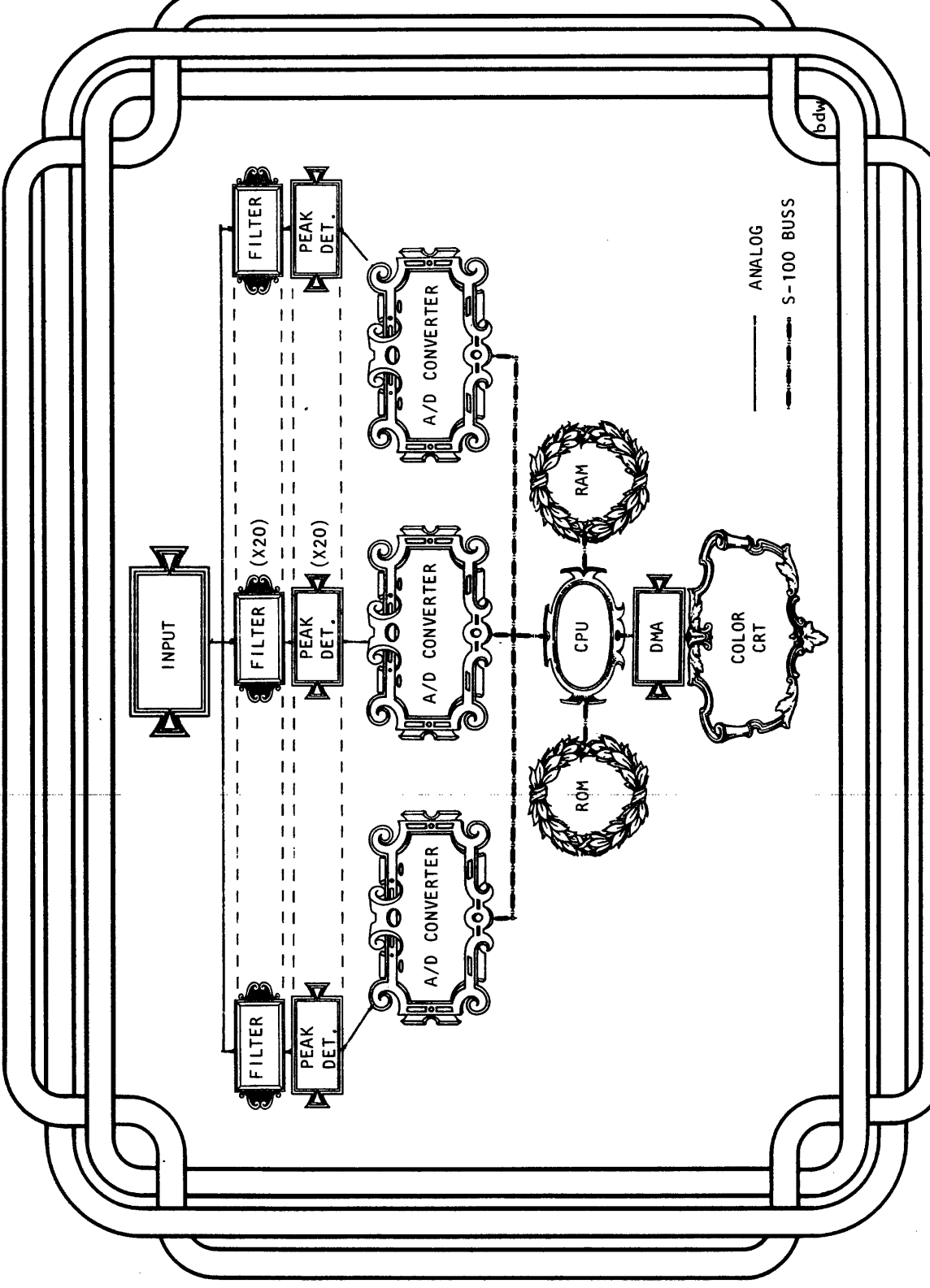
FIGURE 2

```
10 OUT 14,144
20 OUT 15,16
30 B=0
31 Z=8192
32 POKE Z,B
33 Z=Z+1
34 IF Z 8705 GO TO 32
40 C=51
50 P=8192
60 I=25
70 GO SUB 500
80 C=85
90 P=8194
100 I=26
110 GO SUB 500
120 C=119
130 P=8196
140 I=27
150 GO SUB 500
160 C=153
170 P=8198
180 I=20
190 GO SUB 500
200 C=187
210 P=8200
220 I=29
230 GO SUB 500
240 C=221
250 P=8202
260 I=30
270 GO SUB 500
280 C=255
290 P=8204
300 I=31
310 GO SUB 500
320 C=51
330 P=8206
340 I=33
350 GO SUB 500
360 GO TO 40
500 D=INP(I)
510 L=(8704-P)
520 IF L 0 THEN RETURN
530 IF (D*2) =L THEN POKE P,C: TO GO 550
540 POKE P,B
550 LET P=P+16
560 GO TO 510
```

FIGURE 3

```
ASSM 0700

0700 31 A5 07    U    0010 BEGIN LXI SP,TOPS+20  (SET STACK POINTER)
0703 3E 88            0020       MVI A,88H  (SET DAZZLER
0705 D3 0E            0030       OUT 14        FORMAT &
0707 3E 10            0040       MVI A,16         PICTURE
0709 D3 0F            0050       OUT 15              LOCATION)
070B 01 00 02         0060       LXI B,512  (SET ERASE LOOP COUNTER)
070E 21 00 10         0070       LXI H,1000H  (SET ERASE LOCATION)
0711 36 00            0080 ERASE MVI M,00     (DRAW BLACK @ LOCATION)
0713 0B               0090       DCX B      (DECREMENT E.L. CNTR.)
0714 23               0100       INX H  (ADVANCE TO NEXT LOCATION)
0715 78               0110       MOV A,B (LOAD COUNTR. TO ACCUMULATOR)
0716 FE 00            0120       CPI 00 (CHECK IF CNTR. IS DEPLETED)
0718 C2 11 07         0130       JNZ ERASE (IF NOT, REPEAT LOOP)
071B 79               0140       MOV A,C (REPEAT CHECK
071C FE 00            0150       CPI 00          ON OTHER BYTE
071E C2 11 07         0160       JNZ ERASE           OF LOOP COUNTER)
0721 06 33            0170 BARA  MVI B,51 (LOAD COLOR OF FIRST BAR)
0723 DB 19            0180       IN 25    (LOAD A/D DATA FROM PORT 25)
0725 21 00 10         0190 APLAC LXI H,1000H  (SET START ADDR. OF BAR)
0728 CD 74 07         0200       CALL PAINT    (GO TO DISPLAY SUBROUTINE)
072B 06 55            0210 BARB  MVI B,85
072D DB 1A            0220       IN 26
072F 21 02 10         0230 BPLAC LXI H,1002H       (REPEAT FOR NEXT
0732 CD 74 07         0240       CALL PAINT               BAR)
0735 06 77       .    0250 BARC MVI B,119
0737 DB 1B            0260       IN 27
0739 21 04 10         0270 CPLAC LXI H,1004H         (" ")
073C CD 74 07         0280       CALL PAINT
073F 06 A3            0290 BARD MVI B,163
0741 DB 1C   .        0300       IN 28
0743 21 06 10         0310 DPLAC LXI H,1006H         (" ")
0746 CD 74 07         0320       CALL PAINT
0749 06 BB            0330 BARE MVI B,187D
074B DB 1D            0340       IN 29
074D 21 08 10         0350 EPLAC LXI H,1008H         (" ")
0750 CD 74 07         0360       CALL PAINT
0753 06 DD            0370 BARF MVI B,221
0755 DB 1E            0380       IN 30
0757 21 0A 10         0390 FPLAC LXI H,100AH         (" ")
075A CD 74 07         0400       CALL PAINT
075D 06 FF            0410 BARG  MVI B,255
075F DB 1F            0420       IN 31
0761 21 0C 10         0430 GPLAC LXI H,100CH         (" ")
0764 CD 74 07         0440       CALL PAINT
0767 06 DA            0450 BARH  MVI B,0DAH
0769 DB 21            0460       IN 33
076B 21 0E 10         0470 HPLAC LXI H,100EH         (" ")
076E CD 74 07         0480       CALL PAINT
0771 C3 21 07         0490       JMP BARA    (REDRAW SCREEN)
0774 0E 7F            0500 PAINT MVI C,7FH (LOAD LOOP&POSTN. CNTR.)
0776 11 10 00         0510       LXI D,16    (LOAD MEMORY JUMP INCRMT.)
0779 00               0520 SIDOR NOP
077A 00               0521       NOP
077B 00               0522       NOP
077C 00               0523       NOP
077D B9               0524       CMP C   (COMPARE INPUT VALUE&SCREEN POSTN.)
077E 00               0525       NOP
077F 00               0526       NOP
0780 00               0527       NOP
0781 00               0528       NOP
0782 DA 8F 07         0530       JC BLACK   (IF POSITION GREATER,G.T.B)
0785 70               0540       MOV M,B (IF NOT,DRAW COLOR) BLACK
0786 19               0550 INCRM DAD D         (ADVANCE POSTN.)
0787 0D               0572       DCR C
0788 0D               0573       DCR C   (DECREMENT LOOP&POSTN.CNTR.)
0789 0D               0574       DCR C
078A 0D               0575       DCR C
078B F8               0580       RM     (RETURN FROM SUBRTN.IF DONE)
078C C3 79 07         0590       JMP SIDOR (CONTINUE BAR)
078F 36 00            0600 BLACK MVI M,00H (DRAW BLACK)
0791 C3 86 07         0610 TOPS  JMP INCRM  (CONTINUE LOOP)
EXEC 0700
FILE /FREAQ/
FREAQ 0100   061E
```

FIGURE 4

PERSONAL COMPUTING AND THE PATENT SYSTEM

David B. Harrison, Esq.
Owen, Wickersham & Erickson
433 California St., San Francisco, CA 94104
(415) 781-6361

Abstract

A cursory view of the patent system with focus upon its application to personal computing. The difference between inventions and patents, and requirements for patentability including a general discussion and update on the patentability of software; the relationship of patents to copyrights and trade secrets; and, obtaining licensing and enforcing patents -- why bother?

Introduction

When you stop to think about it, we live in a truly exciting time. And this fact is proven, at least, by our common interests in the present and future of personal computing which bring us together at this Second West Coast Computer Faire -- note please that the word "faire" is spelled F-A-I-R-E. In Marin County where I live, in late summer each year there is a Renaissance Pleasure Faire, spelled F-A-I-R-E, which celebrates the Renaissance, that glorious period of growth in the arts and sciences of four hundred years ago. Today, we celebrate a new Renaissance, a golden age of computer power and promise, an advanced technology which we have inherited from all of the innovations of the past, and yet which is but a crude and primitive glimpse and promise of what we can expect for the future.

Over the last three hundred years it has come to be recognized by enlightened people and their governments that human progress is promoted, particularly in the useful arts, by having laws which protect creative works of authorship and invention. What I'm talking about are the copyright and patent laws which protect authors and inventors.[1] Now, by laws which protect authors and inventors, I mean laws which shield them from infringements by others. I do not mean laws which grant inventions to inventors or works of authorship to authors -- they own their original works from the moment of creation.

What our intellectual property laws do is to recognize that unless a reasonable legal shield is provided, authors and inventors simply will not be motivated to disclose their works and discoveries -- they will keep them secret, and the rest of us will not learn about them and have a basis upon which to make improvements. Leonardo Da Vinci's secret discoveries or speculations in the Fifteenth Century about the parachute and the helicopter[2] were uncovered and appreciated only in recent years, and provide a good example of the point I am trying to make. Fifteenth Century laws did not promote the useful arts and offered no incentive to men and women like Leonardo to make their discoveries public.

Now, I have a little more background to give you before I get into my subject entitled "Personal Computing and the Patent System," a subject I happily approach from my perspective not only as a practicing patent attorney but also as an owner and user of a personal computer.

I am only going to mention copyright peripherally because another attorney interested in personal computing, Ken Widelitz from Los Angeles, is giving an excellent presentation on that subject, and each of us has promised not to steal the other's thunder, so to speak. So, for information on copyright I refer you directly to Ken's presentation.

It is my thesis that the encouragement which our Patent System has given countless thousands of inventors has made personal computing possible. A little bit more history, and then we will define patents, explain how they

are obtained and used and touch upon the dilemma confronting us as to patent protection of software.

The Telephone Grandparent

More than one hundred years ago in 1876, Alexander Graham Bell invented the telephone and secured a patent on it.[3] Since that time, the telephone has grown to be an absolute necessity of modern life and it is not surprising to learn that today the telephone company, AT&T. has more U.S. Patents currently in force than any other single entity, approximately ten thousand patents,[4] covering everything from plastic materials used in construction of home instruments to fiber optics, to patents describing highly automated, complex switching equipment that makes direct distance dialing a reality. And, it is this latter technology that is in my opinion one of the two grandparents of personal computing. You see, forty years ago, the telephone company had an acute need for a new theoretical approach to the design of automatic long distance dialing equipment. It was a brilliant Bell Telephone engineer, C.E. Shannon, who, in 1937, and fortunately for all of us interested in computers, recognized the potential of the work of the Nineteenth Century English philosopher George Boole, and developed from Boole's works what we now call boolean algebra -- an analytical tool for electrical circuits.[5] (You know: boolean algebra is the algebra of the binary number system, the base two, on-off number system, a system understood and utilized by our personal computers.)

Curiously, or perhaps not so, the telephone company sired the other grandparent of personal computing. While some of the Bell Telephone laboratory scientists were hard at work applying boolean algebra to long distance switching problems, three other billiant scientists at Bell Labs, Bardeen, Brittain and Schockley, were inventing the transistor in 1946.[6] In a book published in the early 1950's entitled Player Piano, author Kurt Vonnegut, Jr. futurized about a centralized national computer he called EPICAC of epic capability, installed in Carlesbad Caverns, New Mexico.[7] You see, to Vonnegut's prescient mind, it was a vacuum tube computer. Had he then known about the future of the transistor, Vonnegut's story might have been an even closer projection of what we have today. As written, Player Piano was an incredible insight into the future, including social dangers which might flow from the misuse of computer power.

Because of the transistor, your personal computer in your home or office is potentially more powerful than Vonnegut's Carlsbad computer or any vacuum tube computers that could ever have been built. The transistor and all of its progeny are the subject of literally thousands of U.S. letters patent, granted to thousands of inventors who have so rapidly advanced our semiconductor technology, from point contact transistors, to junction transistors, to epitaxial transistors, to field effect devices, to low scale integration, TTL, medium scale, and now to what we probably inaccurately refer to as large scale integration, inaccurate in that we are now pushing packing densities upwards with such patented technologies as V-groove[8] and high resolution masking equipment. It probably will not be too long before we will see, for example, a single chip 64K bit random access memory device. Think about it -- the possibilities that lie ahead. And, it has been the disclosures in patents that have provided inventors with the stepping stones of essential technical information which have marked the path that has led us to where we are today -- that brings us here together at this Computer Faire.

Now, hopefully, I have constructed enough of a background or operating environment to make the subject of patents, if not interesting, which is my belief and goal, at least palatable. And with that, let's consider some definitions.

Inventions and Patents Distinguished

Let's begin by comparing and contrasting the terms invention and patent. For every patent there must be an invention, but it does not follow that there will be a patent granted for every invention. In a broad sense each of us in an inventor creating an invention whenever, through exercise of our mentality and skills, we devise something new. The dictionary defines

invention as "a device, contrivance, or process originated after study and experiment." Another definition worthy of note accompanies the word "invent": "to produce something useful for the first time through the use of the imagination or of ingenious thinking and experiment, such as a new machine." Even in the popular definition of invention we find the element of novelty, that is, an invention must be something new or novel.

Before a patent may be granted under our U.S. patent laws, another essential ingredient must be present: "unobviousness," a word of art which, while easily explanable, is somewhat difficult to apply in practice. It is said that a patent will not be granted unless the invention is one which is not obvious to a person having ordinary skill in the particular technology to which the invention pertains. Let's apply this test by an example.

Suppose the invention is a new design for a microcomputer utilizing a bidirectional data bus for communication between central processing and memory. Let's further assume that the design is truly novel, that is, that no one has ever made a microcomputer that looks just like this one. Let's also assume that the person of ordinary skill is a graduate electrical engineer of average skill and five years experience in the design of electronic digital computer systems. Please note that my assumption as to the level of skill is purely arbitrary and not to be relied upon outside this illustration.

Now, let's put this hypothetical engineer of ordinary skill in a laboratory. On the walls of the lab, let's tack up the closest prior art references we can find which describe similar computer systems. Now comes the test: if our mythical engineer is able to synthesize our new invention from a combination of these references with his ordinary engineering skill, then the invention is not patentable. It is said to be "obvious" or too obvious to merit a patent. On the other hand, if the engineer cannot synthesize the invention from the references, then a patent should be obtainable. So we see that the subject matter of a patent is a <u>patentable</u> invention.

Patents are issued to inventors. Since patents are intangible personal property, they may be transferred, licensed to and/or owned by a party other than the inventor. This is often the case where engineers are hired by companies to make developments and innovations, and some turn out to be patentable -- in this situation the employer usually owns the invention, particularly if there is a written employment agreement which says so. Where inventions are made outside the scope of employment, usually the inventor owns the invention, although an employer may be able to claim a limited "shop right" in the invention if its time and/or resources were used by the inventor in making his or her invention. This is a tricky area, and it is not safe to suppose that there are clear cut rules or results. There are a lot of lawsuits filed in this area, particularly in the case of so-called "spin-offs" where a group of employees leave together and set up their own competing company.

The duration of a patent is 17 years from the date of issuance, non-renewable. The average time today for the examination of patents is approximately 19 months from the date of filing to final disposition,9 either issuance of a patent, abandonment of the application, or appeal. Currently there are approximately 1265 patent examiners at the Patent Office in Washington,10 and each has at least a technical background, with some of them also being trained in the law. These Patent Office examiners administer the patent laws on a day to day basis. An applicant unsatisfied with an examiner's negative action has appellate remedies, first to the Board of Appeals in the Patent Office, and then if still unsatisfied, to the Court of Customs and Patent Appeals, a five judge federal court sitting in Washington, D.C.

There is only one patent office in the United States because it is an exclusive activity of the federal government.

States do not issue patents, although this has not always been true, and Massachusetts was the first to grant a patent in colonial days. In 1641 one Samuel Winslow was granted a ten year exclusive right by Massachusetts covering

his particular process for making salt. U.S. Patent No. 1 was issued to a Philadelphian on April 10, 1790 for his apparatus and process for making potash and pearl ash. U.S. Patent No. 4,000,000 was issued on December 28, 1976 to a man from Las Vegas for a process of recycling asphalt-aggregate compositions. Well over 1000 U.S. patents are issued every Tuesday, fifty-two weeks of the year. In 1976, 80,735 U.S. patents were issued, and in the same year 109,227 new applications were filed, which suggests that almost three fourths of the total applications filed resulted in issued patents.

A patent application today requires a minimum $65.00 filing fee, and if issued as a patent, a minimum issue fee of $112.00. These fees are paid to the Patent Office in accordance with federal law. They are above and beyond attorney's fees and charges for such things as making the required drawings.

Contents of a Patent

Before I summarize what has to go into a patent, let me tell you the theory behind patent disclosures. It is like a bargain or deal, a contract between the inventor and the people represented by the federal government. The inventor gives up a full and complete disclosure of his invention -- it is no longer a secret -- with sufficient details to enable a person skilled in the particular technology not only to make the invention but also to make it work and use it for its intended purpose. In exchange for this disclosure, the government gives the inventor a patent for 17 years.

We will talk about the rights that a patent grant conveys in a moment, but let's first briefly mention the four requisites of a patent application: a specification, an oath or declaration of the inventor, drawings when necessary, and the prescribed filing fee, which we have already mentioned.

The specification, which is the required written disclosure of the invention, has several parts which I will list: title, abstract of the disclosure, background of the invention, a summary of the invention, brief description of the drawings, detailed description of a preferred physical embodiment of the invention, and last, but certainly not least, the patent claims, which define in typically stilted yet very precise language the boundaries of the invention, just like legal descriptions in land deeds describe the boundaries of the real property conveyed thereby.

When an application ripens into a patent, there is the grant itself, and the document looks impressive, with blue ribbons and a red seal.

Enough of theory, let's consider an exemplary patent, one I feel is appropriate for this audience. It is U.S. Patent No. 3,821,715 which issued on June 28, 1974 and is owned by the Intel Corporation. This patent describes and claims a general purpose digital computer formed out of large scale integrated circuit chips: one chip being a central processing unit (CPU), the second a random access memory (RAM), and the third a read-only memory (ROM). Here now is the front page of the specification which includes the title: MEMORY SYSTEM FOR A MULTI-CHIP DIGITAL COMPUTER, names and cities of residence of the three co-inventors, Intel as the owner or assignee, the prior art references cited by the Examiner, an abstract summarizing the invention, and one of the figures of the drawing, a figure which is supposed to be most representative of the overall invention.

United States Patent [19]

Hoff, Jr. et al.

[11] 3,821,715

[45] June 28, 1974

[54] MEMORY SYSTEM FOR A MULTI-CHIP DIGITAL COMPUTER.

[75] Inventors: Marcian Edward Hoff, Jr., Santa Clara, Stanley Mazor, Sunnyvale; Frederico Faggin, Cupertino, all of Calif.

[73] Assignee: Intel Corporation, Santa Clara, Calif.

[22] Filed: Jan. 22, 1973

[21] Appl. No.: 325,511

[52] U.S. Cl. ... 340/172.5, 340/173 R, 340/173 SP, 307/238

[51] Int. Cl. G06f 13/00, G11c 11/44

[58] Field of Search ..... 340/172.5, 173 SP, 173 R; 307/238, 279

[56] References Cited

UNITED STATES PATENTS

3,460,094	8/1969	Pryor	340/172.5
3,641,511	2/1972	Crochi et al.	307/238 X
3,680,061	7/1972	Arbab et al.	340/173 R
3,681,763	8/1972	Meade et al.	340/173 R
3,685,020	8/1972	Meade	340/172.5
3,702,988	11/1972	Haney et al.	340/172.5
3,719,932	3/1973	Cappon	340/173 R

3,731,285	5/1973	Bell	340/172.5
3,735,368	5/1973	Beausoleil	340/173 R
3,737,866	6/1973	Gauter	340/172.5
3,740,723	6/1973	Beausoleil et al.	340/172.5

OTHER PUBLICATIONS

Schuenemann, "Computer Control" in IBM Technical Disclosure Bulletin, Vol 14, No. 12, May 1972; pp. 3794-3795.

Primary Examiner—Paul J. Henon
Assistant Examiner—Melvin B. Chapnick
Attorney, Agent, or Firm—Spensley, Horn & Lubitz

[57] ABSTRACT

A general purpose digital computer which comprises a plurality of metal-oxide-semiconductor (MOS) chips. Random-access-memories (RAM) and read-only-memories (ROM) used as part of the computer are coupled to common bi-directional data buses to a central processing unit (CPU) with each memory including decoding circuitry to determine which of the plurality of memory chips is being addressed by the CPU. The computer is fabricated using chips mounted on standard 16 pin dual in-line packages allowing additional memory chips to be added to the computer.

17 Claims, 5 Drawing Figures
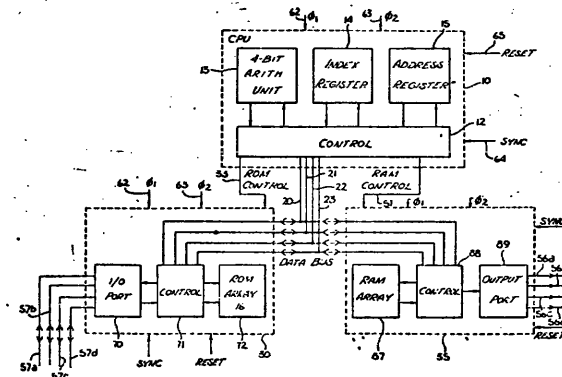
There are five drawings in this patent. In a case such as this, drawings are required and must illustrate every feature of the invention. The drawing on the front page, above, is actually Fig. 1 and is a general block diagram of the computer. The figure reproduced immediately below is Fig. 5, and is said to be a detail block diagram of a RAM chip. Since this invention concerns novel computer system architecture rather than circuit details, the invention is illustrated entirely in block diagrams. Where the invention concerns novel circuitry, schematic circuit diagrams would probably be required. Here now is Fig. 5.
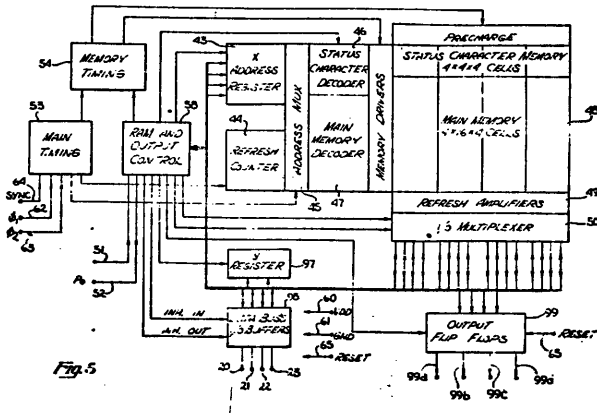
Fig. 5

From this Fig. 5 drawing, you can appreciate that it depicts a four parallel bit, dynamic memory having a total array size of 256 bits -- a very simple device compared to the 4K and 16K devices now available or in late product development.

There are two other illustrations I want to give you. The next is the first page of printed specification, and conveys the flavor of the language in which patent applications are usually written. While the language is clear, it is technical. It does not read as easily or as interestingly as a Vonnegut novel, for example.

3,821,715

MEMORY SYSTEM FOR A MULTI-CHIP DIGITAL COMPUTER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of digital computers.

2. Prior Art

Since their inception, digital computer applications have evolved from calculations through data processing and into the area of control. In recent years, with the development of the so-called "mini computer," applications, particularly in the control area, have greatly increased. Mini computers today are used at the heart of many systems since they are more flexible, can be easily personalized for a particular application, can be more readily changed or updated than fixed logic design systems, and, most significantly, the cost of such computers is much less than the cost of a large general purpose digital computer. Unfortunately, the size and cost of even the smallest mini computers has limited their use to relatively large and costly systems. Because of this, many smaller systems are fabricated with complex hardwired logic circuits.

The present invention provides a general purpose digital computer which may be fabricated for a cost considerably less than the cost of even the smallest mini-computers. As will be seen, the presently disclosed computer can provide the same arithmetic control and computing functions as a mini-computer with a few MOS chips with these chips being housed in standard packages. With the presently disclosed computer, it is anticipated that entire new applications, untapped because of the cost of mini-computers, will become practical. These applications include control functions, such as numeric controls, elevator control, highway and rail traffic control, and process control. The disclosed computer can be used as computer peripheral equipment to control displays, keyboards, printers, readers, plotters and terminals. Other applications for the presently disclosed system include computing systems and countless other applications within the fields of transportation, automotive uses, medical electronics and testing systems.

SUMMARY OF THE INVENTION

A general purpose digital computer which comprises a plurality of separate MOS chips is described. The chips are interconnected by a number of lines, including four bi-directional data bus lines. One chip includes a central processing unit that is coupled by the bi-directional lines to a plurality of memory chips which include random-access-memories (RAM) and read-only-memories (ROM). The ROMs are used to store the computer instructions and other data. A plurality of separate RAMs and ROMs may be added to the bi-directional lines. The memory chips each include decoding circuitry for recognizing a predetermined code, thus permitting the central processing unit to address a single one of the plurality of memory chips even though all the memory chips are coupled to the common data bus lines. In the presently preferred embodiment, each of the chips are mounted on standard 16 pin dual in-line packages and input and output information to the computer is read in through and read out from terminals on the memory chips.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a general block diagram of the disclosed computer. It illustrates the central processing unit, a single RAM and a single ROM.

FIG. 2 is a graph illustrating a single instruction cycle of the computer and is used primarily to describe the manner in which the memories communicate with the central processing unit.

FIG. 3 is a block diagram illustrating the interconnections between the central processing unit and a plurality of memory chips including ROMs and RAMs.

FIG. 4 is a detail block diagram illustrating a single ROM such as the ROMs illustrated in FIG. 3.

FIG. 5 is a detail block diagram of a single RAM such as the RAMs illustrated in FIG. 3.

DETAILED DESCRIPTION OF THE INVENTION

Referring first to FIG. 1, a computer built in accordance with the present invention is illustrated which includes a central processing unit or central processor 10, a random-access-memory (RAM) 35 and a read-only-memory (ROM) 30. In the presently preferred embodiment, four bi-directional data bus lines 20, 21, 22 and 23 are utilized to communicate information from the processor 10 to the memories and also to communicate information from the memories to the processor. Information may be read from the computer on lines 56a, 56b, 56c and 56d and information may be read into or from the computer on the input/output lines 57a, 57b, 57c and 57d. As will be explained in greater detail, the central processing unit or processor 10, the ROM 30 and the RAM 35, are each fabricated on separate MOS chips utilizing MOS technology and are interconnected by the various lines illustrated, including the common data bus lines 20, 21, 22 and 23. These bus lines may be fabricated on a printed circuit board along with lines 62, 63, 64, 65, 33 and 51, and then connected to the processor 10 and the memories 30 and 35 where appropriate. In the presently preferred embodiment the processor 10 and the memories 30 and 35 are packaged in dual in-line 16 pin packages as is commonly used in the semiconductor industry and, as will be seen, the pins on each package have been utilized such that the central processing unit is able to communicate with the memories and the entire computer is able to communicate through the ports provided on the memories with external circuitry.

While in FIG. 1 only a single ROM is illustrated, as will be explained in detail, numerous ROMs may be utilized and may be coupled to the common data bus lines 20, 21, 22 and 23. Additionally, a plurality of RAMs may be utilized and these memories would likewise be coupled to the common data bus lines 20, 21, 22 and 23. The central processor 10, as do the memories, receive complementary timing signals which may be externally generated, on leads 62 and 63 identified as ϕ_1 and ϕ_2 respectively, and illustrated in FIG. 2 as signals 66 and 67. Processor 10, in addition to developing other signals, develops a synchronization signal which comprises a pulse generated every eighth periods of the signals ϕ_1 and ϕ_2. This synchronization pulse is illustrated as signal 68 in FIG. 2 and is communicated from the processor 10 to the various RAMs and ROMs utilized in the computer on lead 64. The central processor also generates a ROM control signal which is communicated to the ROMs utilized in the computer via lead 33

Finally, I have included claim
1 of the patent below so that you
see how precisely claims are
written. The claims are the heart
of any patent and they are drafted
to be broad enough to cover the
invention adequately, but not so
broad as to be invalid for attempting
to cover the prior art. Here is
claim 1:

We claim:
35 1. A general purpose digital computer comprising:
a central processor disposed on a first semiconductor
chip;
a plurality of bidirectional data bus lines;
at least a separate first and second semiconductor
40 memory chip each defining a memory and each in-
cluding a chip decoding circuit for recognizing a
different predetermined code on said bidirectional
data bus lines and for activating a portion of said
memory upon receipt of said predetermined code,
45 said data bus lines interconnecting said processor
and said first and second memory chips for com-
municating said different predetermined codes
from said processor to at least one of said first and
second memory chips and for communicating data
50 signals for one of said first and second memory
chips to said processor;
whereby said processor may communicate signals to
said first and second memory chips and said decod-
ing circuits shall determine which memory is being
55 addressed.

If you have taken the time to read
claim 1, you will note that it is
one long, single sentence which first
describes structural relationships
and ends with the desired functional
result. The subject of patent
claims leads directly, I think,
to the subject of patent rights.

Patent Protection

In general concept, a patent
protects the idea of the invention,
however that idea may be expressed.
This protection is broader than
that obtainable under the copyright
laws which only provide a remedy
against copying the particular
expression of an idea. While pat-
ents protect inventions and copy-
rights protect expressions of
ideas, there is no prohibition
that in some circumstances a pat-
ent as well as a copyright could
be claimed for complementary
aspects of the same subject matter.
Usually, however, that which is
patentable is not copyrightable,
and conversely so.

A patent is distinct from a
trade secret, in that the subject
matter of the patent is known be-
cause it is published when the patent
is issued, whereas a trade secret
must truly be kept as a secret to be
protected. Also, a patent protects

the inventor against later inventors
who arrive at the same invention throug
independent effort. Trade secret pro-
tection may not be claimed against a
third party who discovers the secret
honestly and independently of the
owner of the trade secret. A good
example of a trade secret is the
Coca-Cola syrup formula. Anything
that can be reverse engineered is
not a suitable subject for trade
secret protection. Patent applica-
tions are kept strictly confidential
by the Patent Office before issuance,
and are not disclosed if abandoned.
Therefore, the subject matter of
a patent could be protected as a trade
secret before the patent issues.

A patent grants to its owner a
right of exclusion: that is, the
right to exclude others from making,
using or selling the invention during
the life of the patent. This is a
negative right. It is not the right
to make, use or sell the invention
claimed in the patent. Here is an
illustration: in claim 1 reproduced
above, the language calls for "a
central processor disposed on a
first semiconductor chip." Now,
it is probable that one cannot
"dispose" (that means "make") a
CPU on a semiconductor chip without
infringing patents covering the
fabrication of large scale inte-
grated circuits. So, in order to
make a single chip CPU, one would have
to be sure that patents covering chip
fabrication were not being infringed
or more likely one become licensed
under such patents to avoid any claims
of infringement. Another reason
that the patent grant is a right
of exclusion is that in some situa-
tions, other laws or public policy
prohibit the manufacture or use of the
invention -- examples might be cycla-
mate artificial sweeteners, and
refined cocaine.

The patent grant extends through-
out the United States and is enforced
by a suit for patent infringement
in federal district court. For
foreign protection, a patent must
be obtained or confirmed in each
country, which can be an expensive
proposition. I cannot possibly
cover the subject of foreign patents
in the time allotted, other than simply
to mention that whole area as a
separate problem that has to be
considered.

Time for Filing Application

A patent application may be applied for at any time, subject to some highly technical, but important exceptions. The application must be filed within one year of being patented or described in a publication anywhere throughout the world, and within one year of first being publicly used or on sale in the United States. For inventions made in the United States, foreign applications may not be filed without a U.S. Government license or awaiting six months after first filing a U.S. Patent application. Usually, it is wisest to file a patent application as soon as the invention is completed. An invention is deemed complete when it is either physically made and used --called an actual reduction to practice, or a patent application is filed with a full disclosure of how to make and use the invention -- called a constructive reduction to practice.

I cannot overemphasize the need to keep adequate notes of inventive ideas when the invention is first conceived. A bound notebook is ideal, and the entry ought to include a witness' name and date of reading and understanding the invention. Sometimes it is crucial to have such records when the same invention is made by two independent workers, and it becomes necessary to prove who was first. In any event, a policy of keeping an invention notebook is a must for any serious inventor.

Patent Searching

Before an inventor files a patent application it is usually wise to make a patentability search to see if there are prior patents or other references which disclose or clearly suggest the invention. Most patent searches are made in the public search library maintained by the U.S. Patent Office in the suburbs of Washington, D.C. A number of public libraries around the country maintain collections of U.S. patents as well as some classification information. In this geographical area, the Sunnyvale public library maintains copies of U.S. patents issued since about 1960. It is possible for inventors to make searches themselves in facilities like the Sunnyvale Patent Library, but the work is arduous, and the results are generally far less reliable than if the search were made by professional searcher in the Patent Office search room. Patent attorneys maintain close ties with searchers in Washington, and patent attorneys may be located by looking in the yellow pages under "Patent Lawyers" which is a separate classification from attorneys in general.

Patentable Subject Matter

Perhaps you have noticed that I have not yet told you what is patentable and what is not. This has been intentional. I have deferred that topic until now because it leads us right into computer programs and patentability, my last topic.

Patents are supposed to promote the useful arts. Therefore, patentable subject matter must be useful, that is, capable of being used for some purpose or function. An abstract idea, such as a machine for generating the sound of one hand clapping, would not be useful. So also, perpetual motion machines are said not to meet the utility requirement. Anything that simply will not work, that purports to defy the laws of physics or chemistry may run afoul of the utility requirement.

Assuming that the subject matter is new and useful, it must fall within one of the four categories of process, machine, article of manufacture or composition of matter.

Process means method and patents for processes may be concerned with methods for making chemicals, forming articles, and making measurements or calculations with electrical signals.

The difference between a machine and an article of manufacture is that a machine is said to have an inherent law of internal operation whereas an article does not. Another claimed distinction is that a machine has moving parts whereas an article does not, but this second criterion breaks down with personal computers which have sophisticated rules of internal operation but no moving parts

(except perhaps the cooling fan for the power transformer).

A composition includes mixtures of matter and also compounds. Chemical patent applications are subject to special rules and are drafted a little differently than are mechanical and electrical applications. There is a great deal of overlap in such areas of electrochemistry as the process of fabricating large scale integrated circuits on monolithic silicon wafers.

So the four classes of patentable subject matter are process, machine, article and compositions of matter. Here are some subjects that have been held not to be patentable: mere arrangement of printed matter, articles naturally occurring which are unaltered, methods of doing business and accounting; and, of most importance to us, the last one is scientific principles, including mathematics and algorithms, which brings me directly and inescapably to the patentability of computer programs.

Patentability of Computer Programs

Pure mathematics has traditionally been characterized as a part of the liberal arts and philosophy. It has been held by our U.S. Supreme Court[11] that a pure mathematical formula or algorithm is not patentable because a patent would wholly pre-empt the mathematical formula. In practical effect the patent would cover the algorithm itself, and deprive all but the patent owner of the right to apply the algorithm to any problem in any technology. In the case to which I refer the algorithm was a pure algorithm; it had to do with the conversion of binary coded decimal numbers into pure binary numbers. Its application was said to be entirely abstract, it operated only upon numbers per se and was not applied to any physical environment or phenomenon. This case, Gottschalk v. Benson, decided in 1972, has been interpreted by lower courts as well as commentators as being a very limited decision -- which is that computer programs involving algorithms which do nothing more than process numbers without any relationship to or impact upon the physical world, are not

patentable. On the other hand, the Patent Office has interpreted this case much more broadly. The present Patent Office position appears to be that if the formula or algorithm is the only thing new or novel, then what is involved is a non-patentable mental step, even though the algorithm is claimed to operate upon something in the physical universe. The analogy that the Patent Office used in recent papers filed with the United States Supreme Court[12] in urging it to review a lower court's decision was as follows:

"It is as if respondent [patent applicant] were trying to patent the Pythagorean Theorem as a part of a method, by adding a final step that suggested applying the solution of the equation to surveying. Such a last step is not an integral part of a new invention, but rather the non-inventive application of a mathematical result to existing technology; that fact is not changed by the facile device of so drafting a claim that it places an abstract mathematical algorithm within a recitation of steps performing conventional, existing technology."

So let me summarize my personal opinion as to where we appear to be at the time of the writing of this paper (early January 1978): (1) if the only thing that is novel and unobvious is an abstract algorithm that is not applicable to the physical world -- no patent. (2) If the novel algorithm is applied to the physical world by operating within a standard general purpose computer, the Patent Office will likely hold that it is unpatentable, but the Court of Customs and Patent Appeals might reverse and grant the patent. (3) If the novel software operates in conjunction with novel hardware, then the Patent Office may then grant a patent, assuming other conditions for patentability are met.

One argument that is heard every so often is that a unique computer program defines a unique computer structure when loaded into the computer's program memory. And, literally this is true. However, if this were the case, then the program which converted BCD to pure binary would have been held patentable (which it was

not in the Benson case) because the
computer which that program defined
would have been unique. So, it
seems safer to look to physical
contact, manipulation or impact as
an indicator of patentability for
computer programs, at least at this
point.

From my experience and observa-
tions, it is my opinion that the
chances of obtaining a patent involv-
ing a program are enhanced if one pur-
sures a strategy of emphasizing the
hardware and physical contact
aspects of the invention, if pos-
sible. For example, if the pro-
gram resides in the read only memory
as firmware, then one might claim
a read only memory arranged to define
the novel algorithm -- this way you
claim physical structure, not a
mental step, and hopefully maybe
you can avoid a rejection by the
examiner that the claim is non-
patentable subject matter.

One cannot mention protection
of software without noting that it
is a subject matter which is copy-
rightable. Whether the copyright
law provides adequate protection is
a matter of great national debate.[13]
Remember, we have already noted that
copyright only protects against copy-
ing of the expression of an idea, not
the idea itself. For more informa-
tion regarding copyrightability of
software programs, I refer you again
to Ken Widelitz' presentation.[14]

Conclusion
        In presenting this paper I
am reminded of a definition I once
heard for an expert. It has been
said that an expert is like the
bottom half of a double boiler --
it puts out all the steam, but it
really does not know what's cooking.
Those of us who work with patents
sometimes find ourselves generating
a lot of hot air without really
knowing whether those warm air currents
have any utility. I hope you have
profited from this presentation.

Today we have discussed the
need for our patent and copyright
laws to foster and promote the pro-
gress of science and useful arts by
protecting authors and inventors in
giving them exclusive rights to
their works for limited periods of
time. We have noted that for inven-
tions to be patentable they must be not
only novel, but also unobvious to the
ordinarily skilled worker in the

technology. We have noted that a
patent requires the inventor to
exchange a full disclosure of the
invention in exchange for the 17
year exclusive right to exclude
others from making, selling or using
the invention, as claimed. We have
illustrated the disclosure require-
ment by looking at parts of Intel's
patent for a four bit bidirectional
data bus microcomputer. And, finally,
we have noted the problems confronting
us in the troublesome area of the
patentability of software.

It has been my experience that
electrical engineers and technicians
tend to be skeptical about patents
for such things as computers. I
often hear, well, you can't patent
Ohm's law, and that, of course, is
true. But the development of ele-
gant unobvious hardware and hardware-
software combinations may be appro-
priate subjects for patents. Here-
tofore, the personal computer industry
as a fledgling industry has not been
faced with the need to deal with
patents and the patent system. With
the popular press projecting annual
sales in the personal computing
industry to reach 1.5 billion dol-
lars by 1985,[15] it is apparent that
this industry will come into contact
with patents on a more frequent
basis.

Let me leave you with some
figures that I think prove my point.
Intel, for example, has about 30 com-
puter related patents, Hewlett-
Packard has about 200 patents on
computer/calculators, and the IBM
Corporation has over 9,000 U.S.
patents now in force and effect,
covering a broad spectrum of sub-
jects from typewriter ink composi-
tions to large scale computer
architectures.[16] As a company grows
in size, apparently so does its
patent portfolio.

We have just begun to glimpse
the possibilities for personal com-
puting. But what I am waiting for
and would love to see invented, is
a truly personal computer, one that
interfaced directly with my brain.[17]
Then, theoretically, I could perform
complex mathematics, compose symphonies,
translate languages and hopefully
advance the state of humanity. This
is one of my hopes and speculations
for the future of personal computing
and it is people like you that possibly
will come up with an invention to bring
it into reality. Thank you for your
attention.

Foot _es

[1] The Copyright Laws, Title 17, United States Code; The Patent Laws, Title 35, United States Code

[2] _Eureka! An Illustrated History of Inventions from the Wheel to the Computer_, Holt, Rinehart & Winston (c) 1974, pp. 24, 39.

[3] U.S. Pat. No. 174,465, issued March 7, 1876 to A.G. Bell.

[4] Information kindly supplied by the Patent Department of AT&T.

[5] Burroughs Corporation, _Digital Computer Principles_, 2d.Ed. McGraw Hill, (C) 1969, page 94.

[6] _Eureka! An Illustrated History of Inventions from the Wheel to the Computer_, supra Fn 2, p. 231.

[7] K. Vonnegut, Jr., _Player Piano_, Dell Ed., (c) 1952, page 17.

[8] See, e.g., U.S. Patent No. 3,924,265, issued December 2, 1975 to Rogers for "Low Capacitance V-Groove MOS NOR Gate and Method of Manufacture."

[9] United States Patent Office, Office of Information Services.

[10] Ibid.

[11] _Gottschalk, Acting Commissioner of Patents v. Benson_, 409 U.S. 63 (1972).

[12] _Parker, Acting Commissioner of Patents v. Flook_, No. 77-642, filed Nov. 2, 1977, Petition for Certiorari from _In re Flook_, 559 F.2d 21 (CCPA 1977).

[13] The Report of the Software Sub-committee to the National Commission on New Technological Uses of Copyrighted Works, issued in June, 1977, opines that copyright is the most appropriate form of protection for software programs and proposes changes to Section 117 of the New Copyright Act. A dissent by John Hersey argued that since a computer program was only operable as part of a computing machine it did not belong in the category of copyrightable subject matter and proposed a new special law for the protection of software, a so-called Computer Software Protection Act, suggesting a 10 year period of protection for software.

[14] See Ken Widelitz' Legal/Business Forum column in the November 1977 issue of _KILOBAUD_, Page 14. For an excellent legal discussion on the overall subject, see "Computer Programming and Patent Law," _American Patent Law Association Quarterly Journal_, Vol. V., No. 1, 1977.

[15] _U.S. News & World Report_, Nov. 21, 1977, page 77.

[16] Information kindly supplied by Patent departments of Intel, Hewlett-Packard and IBM, respectively.

[17] Prof. Carl Sagan suggests this in his currently popular book _The Dragons of Eden, Speculations on the Origins of Human Intelligence_, Random House, 1977, page 205.

ABSTRACT

Copyright and Software:

Some Philosophical and Practical Considerations

by Kenneth S. Widelitz, Attorney at Law

10960 Wilshire Boulevard, Suite 1504, Los Angeles, CA (213) 477-3067

At the present time the National Commission on New Technological Uses of Copyrighted Works (CONTU) is working on a report which will recommend the manner in which a Copyright Law should be modified in order to protect computer programs. The committee's latest report, issued in June, 1977 is available from CONTU, Washington, D. C. 20558, telephone (202) 557-0996.

The latest CONTU report discusses the three currently available vehicles for protection of computer programs, namely, copyright, patent and trade secrecy. As the report states, "...copyright is designed to protect the expression of ideas while patent's purpose is to protect what are generally understood to be inventions -- in a sense the ideas themselves." Patents, on the other hand, protect inventions which must be useful, novel and not obvious with those familiar with the related technology. Trade secrecy involves the dissemination of information pursuant to contractual agreements by the terms of which the party receiving the information promises not to reveal it to anyone else.

Philosophically speaking, issues raised in discussing whether software should be protected by copyright involved the use of analogies in which the statement "computer programs or more or less like..." are made. The question becomes, is a computer program a writing and/or a mechanical device. Other "philosophical" issues are when is a copy of a computer program made? When it is loaded into memory or when it is dumped onto tape or hard copy. Another issue involved is when is a program based on a previous program a derivative work. That is, an author of a book written in English has the exclusive right to translate it into French (read BASIC for English, COBOL for French?)

The Copyright Act of 1976

The Copyright Act of 1976 became effective January 1, 1978. Section 117 of that Act specifically stated that it did not change the rights of an au-

thor of a computer program as such rights existed under the old law. CONTU, in its report, recommends a new Section 117. However, many of the new provisions of the Copyright Act do currently effect computer programs.

The new Copyright law makes the term of copyright equal to the life of the author plus fifty years. The new law also makes the concept of "publication" less important. The new law also defines when a work is "created", when a "copy" is made and what constitutes "a work made for hire."

Under the Copyright Act an author has the right to do any of the following: (1) to reproduce the copyrighted work in copies; (2) to prepare derivate works based upon the copyrighted works; (3) to distribute copies of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease or lending. Each of these rights are independent of one another so that any one may be retained while any other is sold or transferred.

The question as to what constitutes a derivative work is one of the toughest that computer programmers will face. It must first be understood that copyright protection does not extend to any idea, procedure, process, method of operation or concept. Copyright protection extends only to the expression of ideas, not the ideas themselves. Of course, some ideas, concepts, procedures or processes are so basically fundamental that copyright does not protect an explanation of them. For example, some sorting routines are so basic that they cannot be copyrighted. However, if a series of basic routines are put together so as to accomplish a specific task, that program is subject to copyright.

In order to obtain copyright protec-

tion, there are certain procedural technicalities. These are notice, deposit and registration. Notice requirements are simple. They consist of a C in a circle, the words copyright or the abbreviation copr.; the year of the first publication and the name of the copyright owner (i.e., Copyright, 1978, by Kenneth S. Widelitz.) There are rules relating to where the notice must appear, depending upon the type of work to be copyrighted. The Copyright law also requires that materials which are copyrighted be deposited with the Library of Congress. There are exceptions if the Library of Congress does not desire specific material. Registration with the Registrar of Copyrights is a prerequisite to the bringing of a law suit for the infringement of copyright.

Another consideration is the Doctrine of Fair Use. Fair use embodies that notion that a reasonable portion of a copyrighted work may be reproduced without the permission of an author for a legitimate purpose. The doctrine is most often applied to teachers who have xeroxed materials for distribution to students for educational purposes.

BECOMING A SUCCESSFUL WRITER ABOUT COMPUTERS
Ted Lewis

Every author of a successful book knows the essential ingredients of a "best seller", whether it be a "great American novel" or a book about computers.

First, the author must have something to say. This concept is often overlooked by an anxious novice eager to get his/her words in print. Yet, unless there is value in your words, the book or article will be useless. In computing there are many ideas and concepts that need to be clearly and concisely stated. Unfortunately, books on computing tend to be re-hashes of the same old concepts and ideas inherited from the first 25 years of computing. The microprocessor revolution has changed many of the ideas and motivations in computing. It is this change that may prompt you to write. If so, be sure to take a fresh and innovative approach.

The second essential ingredient to good writing is style. Clearly written books are as valuable as clearly coded programs. Everyone has a unique style, but as authors we owe it to our readers to suppress nonsensical idiosyncrasies and "in" human. Perhaps one of the most difficult tasks for an author of computer books is to avoid use of mnemonics, clever "buzzwords", and trite phrases.

Style also contributes to a book by forcing a structure on the writing. Chapters are the modules of a book; sections are subprocedures, and paragraphs are equivalent to control structures. Thus, a good book has a clean structure.

Style also means the book is organized into a sequence of increasingly complex topics. Initially, a ground-level introduction is used to familiarize the reader with important concepts used later. Don't include material that is minor. Do include material that will help the reader understand later concepts.

A final suggested ingredient in good writing is timing. A "good" book is not always a "good selling" book. The reason is timing. Computer books have a "life" of roughly 3 years. Furthermore, it may take 2 years to produce the book in the first place. This means the author must look into the future to see what will be viable and may be of interest to the future computing community before starting a new book. An author must be a prophet of things to come.

Predicting the future of personal computing is almost impossible. Therefore, writing is a risky business. For example, two years ago a wise writer should have expected a decline in hit building, and a rise in turnkey business systems and disk files. Today, there are several trends leading to "successful" topics for authors of 1980 books.

The foregoing suggests three fundamental concepts of writing a good book. There are many other aspects of writing not covered here. For example, the author-publisher relationship, how to locate a publisher, how to promote an idea, etc. All of these factors contribute to the success (or failure) of a writer.

Nonetheless, the place to start a writing career is with 1) something of value to say, 2) a polished style, and 3) a topic 2 years ahead of itself.

WRITING A USER'S GUIDE .

Douglas J. Mecham
Hughes Aircraft Company, P.O. Box 3310, Fullerton, California 92634

Abstract

Programmers write neat programs that they share with others; thus, there is a need to write useful user guides. From the user's standpoint he needs an easy and simple guideline to be successful. While this documentation task may seem difficult, the programmer writing the user's documentation for his program can also be a success easily and simply. The easy and simple approach is to learn a few easy thought processes. This presentation deals with eight major considerations in writing the user's guide.

Your use of microcomputers has revolutionized the world. No longer is a highly trained specialist required to use a computer. By virtue of your attendance at the Computer Faire you are sharing with others your experiences. These facts require you to communicate your ideas. How well do you do that? If you have to describe in writing how to use your neat program, how well do you communicate? The best idea in the world is not worth much unless it is communicated and used; in other words, your knowledge is worthless if it is not communicated; furthermore, your intelligence of that knowledge is not credited unless your ideas are used effectively.

The reason we should be concerned about the details of documentation is because we need to communicate information in writing about our ideas. This is important now because this is the decade of the user. More people, non-computer oriented people, have direct access to computer systems. These users do not tolerate the way the system wants to operate but want to dictate to the system how they wish it to operate. Good user documentation is the key to effective use of a computer system. This paper will provide some useful ideas to generate good user documentation.

To provide perspective of where user documentation fits consider the three parts of a computer system:

Hardware
Software
Peopleware.

It is within the category of Peopleware that documentation falls. The objective of the user documentation is to break down the barrier between the real user and the computer system hardware and software.

Written information is a personal thing specifically designed for particular situations; however, there are several aspects of it that can be identified. This paper discusses these aspects; in summary they are:

WHO--understand the user's thought process
SIMPLICITY--keep it simple
FORGIVENESS--make recovery easy
EXAMPLES--use plenty of meaningful examples
DETAILS--sink the details
RETRIEVAL--provide easy access
PERSPECTIVE--keep the user point of view at all times
APPEARANCE--it must look nice to read well.

Remember the primary objective of the user information is to make the user successful, easily and simply within his time/dollar constraints. End users are very impatient about time and money.

Who

Consider who the user/reader is. The usual frame of reference for a writer is his own thoughts but for a user document the frame of reference must be the user. You need to determine the user's psychological profile and understand his thought processes. This is not so difficult since you are well aware of how you think as a user of other peoples' software. The type of information presentation a user expects when reading a user document can be determined by choice of form, format, and vocabulary.

First, look at what information the user needs to perform his task. The capabilities of the software should easily define the parameters involved. Not all the information may be required for all users so the user document must present different amounts of information to meet each individual's requirement to complete his task successfully. Certainly there is a minimum amount of information necessary; this of course is the simple path. Then there is an average amount of information required; this would be the normal path a user would follow. There are always those complex tasks that require the user to need the maximum amount of information. Aside from these different paths, the most necessary set of user information needed is when an error or problem occurs.

Second, once the different sets of information have been determined they must be ordered in some manner. This implies a sequence of events. Again, this determination is a function of what the user needs; that is, the sequence of events as the user perceives them. An easy way to determine the sequence of events is to ask yourself "What is the first element of information the user needs," "what is the second element of information the user needs," and so on. Do not fall into the trap of providing too much information on the account of "what if;" consider only the critical information necessary. The user does not want to wade through superfluous information. The user document is not a design specification document and as such the flow of information in the document is by user task, not by technical item commonality or logic.

One of the user viewpoints often left out is describing what the user cannot do. Remember the user's perception of what he expects the software to do may not coincide with what the software can do. The user most likely will attempt to do what he thinks he can do with the software, or what he thinks he should be able to do with the software. Assume very little and leave little to the imagination of the user.

How ever you plan your approach to meet the above criteria you need to plan what information is needed, the information flow, and the information grouping. There is nothing that says that all information groups must be mutually exclusive; do not be afraid to replicate elements of information within several groups. The sequence is important to the user. The information should be in such a sequence to accommodate the novice or student (simple), the experienced user (average), and the computer systems nut (complex).

Notice that the real information needs are based on what the user wants, not on what you think he needs. It must be kept in mind that the user requires only that information necessary to get him to the next step to complete his task. An incredible erroneous concept in data processing is that user documents should define all their technical terms in the front of the document. Then...they expect the reader to remember and understand them for the duration. If you need to define a particular word then define it when you first use it in a meaningful context. Do not be afraid to redefine it later; most technical readers have a short memory so why make it difficult for them.

For any good user document the first paragraph should indicate:
  Who should use the document,
  Why he would want to use the document, and
  How to use the document (conventions and organization).
This way the user may readily find the information he requires without going through useless material. The user document is usually a reference type document, not a novel.

Vocabulary is a sore point with data processing persons, or for that matter persons in any other technical field. The reason for this is because "it is obvious what a term means." Unfortunately, the reader cannot always rely on context to figure out the meaning of a word. Consider the home computer user or any other non-data processing professional coming in direct contact with the computer. The words chosen must not let him think he is getting "computerese" all over him. While some relish the acronym, jargon is always hard to keep straight. Why not use words that are easily recognizable by the user. Using a meta-language (a specially-defined translation language) is a NO-NO--spell it out!

Take, for example, the error message ILLEGAL or FAILURE and consider the psychological ramifications on a law-abiding successful business man. The psychological impact of a user touching the computer is great! At first the user is afraid he will break something or "it" (the computer) will "do something" that will render him less capable to carry on in his life. Or "it" will destroy everything he has worked hard to achieve. The user may think the computer will violate his sense of goodness. One purpose of the user's document is to put the user at ease and dispel any such fears. Thus, the user's anthropomorphic sensibilities are alleviated.

Simplicity

The basic rule is KEEP IT SIMPLE! Whether the user is a sophomore or holds a PhD his desire, or his ability, to read is small. All the user wants to do is get his task completed, simply and easily. Simplicity may be used at all levels of user documentation. Technical information tends to be complex, whether it is or not. All too often numerous concepts and parameters are juggled before the user's eyes. A psychologist once told me that a person can generally only handle eight major items at once.

A user may understand sophisticated vocabulary and complex concepts. But, each time a user's thought process must make a translation or complex transition the probability for loosing information or concepts is significantly high. What the user does not need is confusion. A simple example of this are references to octal or hexidecimal values instead of decimal values. As you will see, simplicity also takes into consideration who the user is.

One technique to keep a user document simple is to take a "storybook" approach. That is, structure the information in a simple, straight-forward, step-by-step manner. When organizing the information choose a critical path for the user to get his task completed whether it is simple, average, or complex level. In order to meet the "storybook" criterion sentences need to be simple.

The "fog" factor of long sentences and large words requires education beyond most users and considerable translation. Remember that television is geared for the ten year old and it is a very successful communications media format.

If you should leave out information by assuming that the user should know it, chances are you will introduce confusion. Why make the user think more than he has to; certainly user documentation is not an examination. Why make it difficult for even the most sophisticated user.

Vocabulary plays a role in simplicity. To keep it simple use common words known to most readers...like the English language. Computerese and FORTRAN do not have commonly known vocabularies. If you do use common words chances are you will relate to your reader and he will understand your concept. For instance, to explain data base concepts use school information for an example rather than nuclear power; most people can relate to information about schools. While there may be some particular vocabulary words required, minimize their number. Why should a user learn a whole new vocabulary just to do his simple task?

Another method of simplicity is to make the wording tight (no extra verbage) but not terse. Leave out computer-oriented words. The five cent and ten cent words put together well are worth more than a fifty cent word that does not fit. When you choose the use of a particular word, be consistent and don't use another word that means the same thing or almost the same thing. This is especially true when defining new terminology; leave the word variety to the novelists. Consistency makes simplicity.

Forgiveness

Naturally, if there is a way a mistake can be made the user will find it. This is easily done because the user performs his task the way his memory tells him to, not the way the computer memory wants him to.

When there is a conflict the computer system should adapt to the user but this is usually not the case. Therefore, it is the user's manual that must make the computer system potable to the user's way of thinking, in the user's terms. Hopefully the computer system will forgive the user and give him another chance. The user manual is forgiving by discussing alternatives should a problem surface, and by not leaving the user to guess what he should do. Computer software usually relies on the user manual to bail the user out of his problem and put him on the right road to performing his task. For example, just think of all the error numbers you have seen as a result of a problem. The user manual must define each error number by a clear indication of what happened, what the current situation is, and what to do about it.

Both the computer system and user's document must allow for quick and easy recovery. Normally, you will waste more than 50% of your time guessing and testing your hypothesis as to why a problem occurred but a good user's document can often cut this effort in half. The vocabulary of the error messages must match the vocabulary of the user document discussion. Abbreviations and computerese are unwarranted.

Be kind to the user for he has problems. The user cannot and does not want to distinguish between errors (specifications violated), need for document clarification, or design change requirements. The user only wants to get his task done simply and easily. Unfortunately most vendors ask the user to make this judgement when he has a problem.

Examples

The first item the user looks for in a user document is an example. More specifically, an example that most nearly matches his task. Even when using the manual as a reference an example will often be needed. Thus plenty of examples in the user's document are most helpful.

Since the user is going to reference the examples so often, several important characteristics should be noted. First, the sequence of events within an example must be very clear and you must distinguish between what the user does and what the computer/software does. To make the example understood provide an explanation of the sequence of events or results right along side. Since an explanation does not leave the reader guessing; be explicit, you know what you mean, so tell the reader.

Examples should be simple, easy to follow, and consistent throughout the user document. Often times, considerable continuity may be achieved by creating several examples around the same topic matter. There is nothing wrong with duplicating examples or portions of examples. It is even desirable to build on a specific example duplicating a previous example to illustrate a sequence of events. This way the user can relate the different examples. Examples are made simple if the subjects are easily related to. For instance, if file records were discussed, the use of bank book accounting may be helpful since the reader can identify with it.

There is no place for clever, funny, complex, or wierd examples. These types only cause confusion. Good examples represent the manner in which most users would use the system. Additionally, the examples described must have been tested and work.

Finally, be careful of the symbology used in examples so the reader can distinguish between form, format, and literal requirements. For instance, italics are often used for a "type" of input such as name whereas upper case may indicate literal information as in the case of ZIP=. Specifying a carriage return is difficult so a symbol such as (CR) might be used since there is no symbolic equivalent. Be careful of specifying literals/strings in quote marks, not all systems expect the user to enter the quote marks with a string value.

For example, consider the message ENTER "YES" or "NO"? Should the user enter the quote marks too? If not, then leave them out.

The more examples you put in your user document the better.

Details

The rule for detail description is to "sink" it. In other words, do not get deeply involved in beginning paragraphs of a user's document, save in-depth discussion for last. Usually the user needs only the straightforward simple information to perform his task. Included in "details" are complex and odd situations which are needed in a small percentage of tasks. If too much detail is required to perform the task, perhaps software redesign should be considered.

In a user's document confront the user first with the easiest and most useful alternatives. Such an approach will minimize room for user problems to develop. Once the user is a success with the simple approach he is ready for more detailed material. Be careful to mention how to use an item as well as just a description of the item. Bulky material such as tables or long lists should not clutter up the text. Since such information is rarely read from start to finish and used normally for reference, move it to an appendix. Besides, the appendix is easier to update than the middle of a document.

How much detail should be put into a user document? Rules of thumb are: enough to assist the user in accomplishing "most" of his tasks and enough to solve "most" of the user problems. The detail material is where the user limitations, pitfalls, and idiosyncracies are discussed. The details are needed only after the user has become a success easily and simply at least once.

Retrieval

If information is useful the user must have easy access to it. The user document is primarily used for reference. Thus, when a user has an information need he thinks of an access point and looks for that point in the user's document. Again, what the user thinks may be different from your point of view so consider several access points to an element of information.

The user most likely will choose a word or words related to the task he is doing. The user will not likely select a word related to design or organization of the software. The implication is not only to index the user document information by user task but to organize the document information by user task.

An alphabetical index is mandatory in a user document of any size. Often a multiple level index is helpful so a user can reference a particular word as it is related in different contexts. Permutting words assists to cross reference material where the index item is more than one word. Likewise, there may be instances where pointers

("see Also") to other information may be useful to a user tracking down information. Vocabulary plays an important role in retrieval since not all users and authors think of the same term to describe a function or other item of information; thesaurus words are very helpful.

When considering the structure of a user document, simple sentences, paragraphs, and item lists are easy to access as opposed to long paragraphical information. Furthermore, the layout can be extremely helpful for physically retrieving information. Some of the latter techniques include shading of key words or syntax forms, marking paragraphs with bullets/dots, drawing rectangles around important information, or using separate fonts such as italics or script as well as bold face type, underlining, simple indentations, and titles.

Tabulation techniques can take the form of colored tabs with printing on them; such tab pages can have summary user information on them for easy access/reference. Simple inexpensive tabulation may be achieved by putting black marks at the edge of a page. Then when the edge of the document is titled the appropriate section is found using an index page in the front of the manual.

The need for easy and quick retrieval of information in a user's document is mandatory since all technical material is referenced after being read. Not being able to easily and quickly find information accounts for a significant amount of a user's wasted time and frustration. The attitude "it's all there...somewhere" is not helpful. If a user has difficulty finding information he will seek other software or computer systems.

Perspective

Often points of confusion develop when a user loses track of who is doing what, when, and where. For instance, when describing interactive dialog it is important to indicate when the user is to enter dialog and when the computer outputs dialog, i.e. who's turn is it. The relationship between several software functions can be lost through all the detail in each, thus a summary of the functions showing or describing their relationship is helpful to a user.

One technique to keep the perspective is to ease the user from a more global concept and scope in on the particular item to be discussed. Other techniques include diagrams, outlines, pictures, and hierachical structured paragraphs. Of course, complete, but simple examples provide good perspective with user related topics. Such complete examples tie together many subtask concepts so the user can see the big picture.

The user's primary concern is to perform his task WITHOUT getting involved.

Appearance

A useful restaurant adage is "if it looks good it tastes good." This also applies to a user's

document and I'll never have to eat my words. This approach is done by all the big computer companies. They publish great looking documents but all too often once you get into it the user's document is little better than useless. For example, one manufacturer published a very pretty 300 page user's reference manual without an alphabetical index but the system sold well (boy! did the users ever complain).

Appearance is effected by the page layout such as the spacing around and between paragraphs, the type font such as script for "smooth" appearance or bold for mandatory and easily recognizable items, and page titles. Bold face font may be used to emphasize key words. Round cornered heavy plastic tabs are impressive though expensive. Even more expensive is the art work done to illustrate and diagram user information but it can appear most effective. One of the most effective ways to create a useful appearance is to use color; although it is expensive. Colors have psychological effects on users. For instance, the error description section of a user manual might have a red tab. On the other hand a tab for the index may be yellow while the sections describing the user commands to execute useful functions might be green. Along this line, the printed page stock could be colored. The appearance of colors dictates how the document is to be used.

If the format appearance is done well a technical document, such as a computer user's guide, can increase comprehension. This is based on the fact that most technically oriented people tend to respond to ordered, sequenced, and logically arranged material.

Technical Information Structure

The following is a guideline for structuring written user information describing a subroutine, program, subsystem, or computer system.

Paragraph 1. Tell why the user would want to use this item of the system/software.
Paragraph 2. General form or format of this item.
Paragraph 3. What are the results of using the item.
Paragraph 4. Typical simple example with explanation.
Paragraph 5. Simple description of the most straightforward use of this item.
Paragraph 6. More detailed description of how to use this item.
Paragraph 7. Discussion of problems, limitations, and other pitfalls the user should guard against. Discussion of what to do if these problems should occur.
Paragraph 8. Description of special and more complex uses of this item.

Wrap-up

You may well realize by now that these documentation ideas for users also apply to many technical documents, including program documentation. By keeping these eight elementary concepts

in your mind you too can be a success with good user documentation easily and simply.

If we are really to meet a user's needs with our computer systems shouldn't we write the user's documentation before we do the design and development of our software/systems?

If the User:
— is looking for solutions
— needs a reference
— normally uses this document
— has a complex task
— finds a user problem
— has a special task

Then he will seek one of the following levels
Why use
General form of the solution
Description of results
Typical example
Simple description
Details
Problem and alternatives
Special uses

Different Needs Dictate Different Uses

References

Mecham, Douglas J.
  You and Written Information
    Technical presentation, HP3000 Users Group Meeting, May 1974, Chicago, Illinois.

Melby, Michael P.
  Written Communications: A System Approach
    Human Factors Soceity Bulletin, Page 1-3, December 1977, Volume 20, Number 12.

Pacific Printers PILOT
  Monthly magazine, Richard Zimmerman, Editor, M. L. Droubley, Publishers, 583 Monterey Pass Road, Monterey Park, California 91754.

How to Use Graphics and Tables, Booklet
  From Interpreting Graphs and Tables by Peter H. Selby, Copyright 1976 by John Wiley and Sons, Inc.

Himstreet, William C.
  Getting Your Words Worth
    Talk, Association for Systems Management Los Angeles Chapter Meeting, 1972.

Birkwood, Ilene
 The Technical Writers Survival Kit
  Hewlett-Packard, Technical Paper given at
  HP3000 Users Group Meeting, Seattle, Washington, September 1977. (Excellent reference.)

O'Hayre, John
 Gobbledygook Has Gotta Go
  Bureau of Land Management, U.S. Department
  of the Interior, U.S. Government Publication,
  1966, 0-206-141. (A must to read.)

Journal of the HP3000 Users Group
 (HP3000 Users Group Newsletter)
  Hewlett-Packard, Santa Clara, California.

65 Notes
 Richard J. Nelson, Editor
  2541 W. Camden Place, Santa Ana, California.

Acknowledgement

EDITING AND PUBLISHING A CLUB NEWSLETTER

Richard J. Nelson
Editor-Publisher PPC Journal
2541 W. Camden Place
Santa Ana, California 92704

Introduction

The fact that you read the title above and
have gotten this far indicates that you would
like to do some writing, have some ideas to ex-
press, or are presently involved in writing, ed-
iting, or publishing. The material that follows
is an amateurs viewpoint, and the only justifi-
cation I have for writing this paper is a suc-
cessful newsletter(1). A newsletter is a fre-
quent, less formal method of publishing timely
information for a fairly narrow readership.
Newsletters are periodicals that offer fast re-
sponse compared to a magazine that is typeset
and requires three months or longer leadtime
from idea to print to reader. The Editor is re-
sponsible for the content of the newsletter and
the Publisher sees that the Editor's material is
reproduced and delivered. For amateur newslet-
ters the Editor and Publisher are often the same
person. This paper will provide an overview of
the task of editing and publishing a club news-
letter. Specific tried and successful techni-
ques will be included and some references provi-
ded. This paper is not intented to cover the
fields of graphic arts, technical writing, or
printing, which would require several books. You
should, however, be able to understand the prob-
lems and have a base from which to start a news-
letter if you are so inclined.

Should I Edit a Newsletter?

The motivation to become famous by circu-
lating reams of world shaking information is
within us all. In the real world, however, Edi-
tors seldom become famous, or even well liked.
A few do, but don't be misled into taking on an
editing task if you don't like hard work and
little recognition. You should plan on at least
a year of editing if you are going to do it at
all. It takes that long to get into a produc-
tion mode and build a readership. Editors are
not necessarily writers; they often write the
material they produce, but they do not create
the material, except for the first few issues in
order to get started. There are exceptions, and
there are many successful newsletters (success-
ful here means profitable) which reflect the op-
inion of an Editor knowledgable in a specific
field. In the computer hobby field you will be

spending your time stimulating, organizing, pre-
paring, and printing. Computers are technical
and you should have some technical or other ex-
perience, such as software or applications, to
draw upon for deciding the content of the news-
letter.

In deciding if you should edit a newsletter
you should be notivated by a desire to serve a
cause, such as a club, or to make a contribution
to the field by providing needed information not
readily available to the intended reader. If
fame and fortune is your goal, don't take on a
club newsletter except as a training experience
for something more profitable later on. Be pre-
pared to work mostly alone and independently.
The mean time to burnout for your helpers will
be measured in weeks and seldom lasts longer than
two issues. If you haven't been too discouraged
so far, let's get into some specifics.

What Do I Publish?

A club newsletter will usually have some
specific goals, such as announcing meetings and
reporting on activities. Unless you are taking
over an existing publication that is well estab-
lished, you will generally have a great deal of
freedom on what type of material you publish.
Success, as defined in this paper, is having
more good material for each issue than space per-
mits, and having a continuous growth and demand
for what you produce. A club newsletter serving
a hobby readership must support itself in most
cases. You must provide material that people
will read. In the United States we have struc-
tured our communications media in such a manner
that we try to make it easy for the reader. News-
letters can't waste space with wide margins,
large type, and profuse illustrations. News-
letters usually have a small readership and lit-
tle advertising. If the material in the news-
letter is not needed (technical) or interesting
(well written and illustrated), it will not be
read. Deciding on what to publish is what makes
an Editor successful.

The first task is to identify the purpose of
the newsletter. If the club is dedicated to
computerized chess, the newsletter style and

content will be different than if the club is dedicated to small business systems. The chess enthusiast is interested in a very specific, very technical subject involving great detail. The business system user will be interested in the system and what other _people_ are doing with it. Most club newsletters are of the latter type, and one of the keys to success is to recognize the people aspect of a highly technical activity. For a computer club newsletter it is inappropriate to get too involved with non-technical topics outside of club social activities. Topics involving religion, social reform, etc., should be included only if the computer plays a role, and the people involved are recognized in their field or are part of the club membership. The decision of what to publish should involve a simple formula or mix. Various categories can be made; they will usually include:

A. club activities
B. human interest/applications articles
C. feature technical articles
D. reporting of events, products, news, etc.

Club activities will include meeting announcements, programs, officer changes, or any item concerning the club that the reader should know about. This information, like all information in the newsletter, should be easily found by using a format common to all issues. Format and organization will be discussed later.

Human interest and applications articles should involve club members if at all possible. Computers are interesting, but people using computers are more interesting. A newsletter is usually of the general interest type, as mentioned above with the business system example. A construction or assembly article is much more interesting if Mary Ann describes her own experiences along with the useful technical details the article provides. This encourages the reader to get involved, because he or she relates to Mary Ann. How other people are using their machines, or the justifications they made for buying their machines, are topics that readers will want to know about. These types of articles will require special effort on the part of the Editor to encourage, nurture, and coax into print.

Feature technical articles will form the real backbone of the newsletter. You will have to know who is doing what in the club to draw upon the members technical expertise. As Editor you may have to write the article after getting the information from the actual source. The person providing information gets the by-line, not you. The Editor rarely signs his name to articles even if he creates and writes the material. The reader can assume that all material that is

not from a specifically acknowledged source has probably been prepared by the Editor, or staff writer (not too common with club newsletters). Few authors will wish to remain anonymous.

Events, products, news, etc., requires that the Editor be well read in the many publications related to the newsletter topic. Scanning 30 to 40 publications a month for useful material for this category is not uncommon for an Editor. These publications may be freebees, or traded for being on your mailing list. If your newsletter is successful and contributes ideas, other Editors will want to exchange publications.

What Format Shall I Use?

The format of a newsletter, as the term is used here, is the layout of the material on the page. Margins, number of columns, typesetting, headings, and use of character size, upper and lower case, etc. A newsletter could be printed on a microdot, or it may be a simple page-wide column typewritten piece. Neither of these two extremes are recommended. Avoid special type styles, and DO NOT USE ALL UPPER CASE for text. It is unfortunate that most computer printers are not designed for human beings to read, but this is changing as computer and peripheral manufacturers realize that computers process text and the output should be readable by humans. If possible, type the text and use computer printed material for program listings and tables. The tradeoff is, of course, readability versus accuracy.

The choice of a format will depend on what equipment is available to you, how much information you want to squeeze onto a page and, to some extent, the readership you hope will read the newsletter. Gather at least a dozen different newsletters and study them. Try to get as many different kinds as you can. Look over the formats and observe the following:

a. how wide are the margins?
b. how many columns?
c. is typesetting used?
d. what type styles are used?
e. is colored paper used?
f. what kind of illustrations are used?
g. how many, what percentage of space, for photos?
h. are the pages bound (folded) or single sheets?
i. is it three-hole punched?
j. what is the content, and readership of the newsletter?

As you study various newsletters and answer the above questions you will be able to get a 'feel' for the type of format that you will want to use. Here are my recommendations:

Use a two-column format as a compromise in paste-up convenience and readability. Do not right-justify; most studies show that right-justification slows down the reader and, unless you are trying to make an 'image' of being a so-called, high quality piece, the extra production effort detracts from subject content. Reduce standard typewritten (10 or 12 characters per inch) produced text to achieve at least 1,200 to 1,500 words per page. Start page numbering from Page 1 which is also the front cover. Leave a wider left margin on odd-pages and right margin on even-pages for binding or hole punching. Make top and bottom margins equal. Place page number, newsletter name or logo, volume and issue or date, on each and every page. This should be standard practice for everything printed today because of readily available photocopying. Most people want to know the source of useful information and each page copied should have that information.

Masthead. At the top of Page 1 of each newsletter you will see the name and other information relevent to the newsletter. The masthead may be a simple name or it may be an elaborate design. The following information may be part of the masthead:

1. publication name
2. short description or explanation of the subject, scope, or purpose of the newsletter
3. LOGO
4. issue identification by date, volume, issue
5. cost
6. copyright symbol

Time. The readers time may not be an obvious consideration in choosing a format. A technical publication is usually read twice; initially, and later as a reference to obtain a specific part of the information. The layout and format, and especially the titles and illustrations, should be chosen carefully for descriptive accuracy and visual association. The effectiveness of the newsletter as a reference is directly in the hands of the Editor. Time is also saved by utilizing the reader's familiarity with the newsletter. A newsletter that is consistent from issue to issue is easier to produce and read if a formal format is established.

Formal Format. Write down a sketch of the layout that will define your format. Keep it handy so you may use it when you assemble the newsletter. The mechanics of the newsletter should be clearly defined and executed as a matter of habit so you can be concentrating on the details of the content. As Editor/Publisher you may often have to be proofreader, illustrator, artist, etc. Once you prepare your formal layout you won't have to think about margins, lines

per inch, column width, reductions, etc. A typical layout is shown in Appendix A. The figure shows the paste-up page and its dimensions along with the print columns and margins. The formal format should include the specifications for the newsletter and the paste-up. Reductions may be chosen as those used on the Xerox 7000. Reductions #2 or 3 are good. #3 is especially practical because the back of unusued, 132-column computer printout sheets (11"x14") make good paste-up sheets. The #5 reduction (61.5% of original) is the limit that any reader will tolerate and should be well justified before using. Most will find the print too small for casual reading.

Preparing the Text. After all of the above preparations have been made, the actual newsletter can be started. The many aspects of writing cannot be covered here except to provide a not-so-obvious truism; you learn to write by writing. The mechanics of getting from idea to typed text varies considerably from the Editor who only has pen and paper to the Editor who writes, edits, and types his copy. Unless you are an excellent speller, have someone else read your copy before starting the cut and paste stage.

Typing. The ideal typewriter is the IBM Correcting Selectric. The advantages of being able to use different type elements (balls) gives you a little more freedom in being able to express complex technical ideas in written form. The newsletter material is single-spaced and typed the column width as defined by the format specifications. This means that a single column typed on 8½"x11" sheet will fit with space in the margins for notes. Column length can be thought of as continuous from beginning to end. If you make a major error and wish to retype a single line or more, just make a notation in the margin to cut out during the paste-up stage.

Masthead. Your masthead will be the same for each issue except for Page 1 text and date, volume, number, etc. I suggest that you prepare a master page with all the material that won't change from issue to issue, and print 100 copies This page will serve as a special paste-up sheet for Page 1. This method insures a consistent print quality for your image-masthead. The idea approach is to prepare a paste-up sized master and print 100 copies of it for Page 1. The larger sized plates and press required to do this makes the cost beyond the normal club resources.

Photographs. Photographs cannot be used as originals for making plates for a printer to print. Printing involves putting ink on paper. There is only white paper and black ink. There is no gray or shades of black! A photograph has

many shades of gray and must be screened to provide a photograph (positive) that is made up of black dots that are either touching (black) or further apart (appears as gray) to give the appearance of shades of white to black. The size and spacing of the dots control the detail of the printed image. If the dots are small and close together, a higher resolution (number of lines distinguishable per inch) is possible. For low cost printing, especially paper plates, photographs should be screened 65 or 85 lines. If you take the photographs, or print them yourself, you should make them a little on the light side. Mount them on a piece of cardboard and take them to a graphics arts studio that will screen the photos with a 65 or 85 line screen. Ask for a screened positive. The originals will be returned to you so you can return them to the author. Photographs are easy to include in the newsletter, and once you have found a quality source for the screening, you should have some photos in every issue.

If your screened photograph is a Kodak PMT and you save your paste-ups for reprints at a later time, I suggest you wash them if your graphics arts studio does not. If they are not washed, they may turn yellow in a few months. Simply place the prints in the full bathroom sink and slowly run cold water to rinse them for 10-15 minutes. Pull out, place on the bathroom mirror and use a clean window squeegee to remove excess water from both sides, and then hang to dry.

The Paste-Up. If the newsletter page is to be reduced, the camera (or Xerox) copy must be larger. For example, this paper was prepared on a paste-up sheet with page size 10.24" x 13.25" which, when reduced to 83% of paste-up is 8½"x11". Don't get confused with the ratios; professionals work in picas, etc. Having an engineering background I simply use a ruler maked off in tenths of an inch and use a calculator. Two factors should be noted on your formal format sheets; one to determine paste-up size from newsletter size (greater than one), and one to determine newsletter size from paste-up size (less than one). The table below illustrates the relationships. The two factors are reciprocals of each other. The Xerox 7000 reduction #3 is used:

Paste-Up Sheet	Reduction % of Paste-Up	Multiplier for Newsletter Size	Multiplier for paste-up size
11"x14"	77%	0.77	1.30

Table 1. Paste-Up vs Newsletter Page Sizes

It is not practical to type directly on the paste-up sheet even if you have a wide carriage typewriter. The common practice is to cut and paste the material to the paste-up sheet. Rubbe cement is often used but I do not recommend that you use it. The best method is to use a hand waxer, or Scotch #810 tape.

Paste-Up Techniques. The equipment used to perform the newsletter paste-up is described in the following section. You have typed your articles in the column widths as determined by your format, and are ready to start your paste-up. Commercial paste-up sheets are available from printers or Graphic Arts suppliers. They will have light blue lines that serve as guides to attach the prepared typed text. You may not be able to find ones to suit your needs and will have to improvise. If a 77% reduction is used, 11x14" computer sheets work fine. Cuff off the holes at the end and prepare a stack for a years use. Add your column guidelines by using a light blue pencil and a cardboard template that has cutouts and a block of wood for a handle. The side margins will not be equal, so mark the top of the template ODD PAGE with the wide margin at the left; turn the template 180° and mark the top EVEN PAGE with the wide margin to the right. You may mark all pages the same way and rotate them as you use them, or you may mark each page O or E in sequence. The latter method takes longer, but helps prevent errors if a non-reduced page is used. Use one system and be consistent. The idea is to establish a system for the first issue and have the mechanics follow the system so you can concentrate on composing the page.

The paste-up sheet will be reduced by Xerox or camera by the printer. When a column of typed text is cut by papercutter I recommend leaving about 1/16" white space on each side of the column. This allows the printer to be able to remove any shadow if he is using an electrostatic plate maker. If you reduce your paste-up on a Xerox machine, I suggest you place a soft, thick, white ink blotter, larger than your paste-up sheet, on top of it. Press on the Xerox rubber cover to keep the paste-up sheet as close to the glass as possible - the ink blotter paper (or dozen sheets of newspaper) helps distribute the pressure to keep the whole sheet flat. This will reduce the shadows that may be picked up by the non-parallel light source in the machine.

A useful technique for cutting the typed text with a papercutter is to place a large white sheet of paper on the table under the blade. This reflects overhead light, and your hand placed above the text being cut provides a light contrast that shows a shadow of the bottom edge of the papercutter. A precise cut can be made using this technique. A single line of type can be

removed without difficulty. An Xacto knife is useful, but the papercutter method is faster and easier.

Attaching the cut text or photos to the paste-up sheet may also be done using a small sliver of Scotch Magic Tape #810. Using a tape dispenser, a pair of scissors, and tweezers, cut the tape as follows: pull tape out 2", cut 1/16" off the end to remove the jagged edge made by the previous cut of the dispenser. Hold the 'floating' end of the tape with the tweezers. Cut a 1/8" length with the scissors. Hold with tweezers and tape the photo, etc., to the paste-up sheet. This small amount of tape won't be a problem for the plate maker (larger pieces will reflect too much light), and using the tweezers to slip under the 'pasted' material it is easy to move if required.

Using a hand waxer is the best way to assemble a newsletter. The paste-up sheet may be waxed, but it is best to wax the material being laid down on the paste-up sheet. A stack of newspapers, cut in half to reduce table space, makes a good work area to roll the waxer over the text, etc. (upside-down). The newspaper is discarded once there is wax on it to avoid the disaster of getting wax on the text side of the paper.

Equipment. Preparing a newsletter requires some basic equipment which is essential if you expect to survive even one year. The first piece of equipment you should have is a large, good quality papercutter. Spend the $20 to $35 to get one that has at least a 12" blade. Next is that marvelous gadget - the waxer (2). A waxer is a small roller with a heated reservoir of hot wax which rolls a 2" wide layer of a special sticky wax over anything. The paper sticks because the wax has a sticky characteristic that holds paper to paper when a small roller is used to press the sheets together. Tweezers (Clauss #AA), Xacto knife, and large 18" ruler with 1/10" scale are important tools for an Editor doing his own paste-ups.

Printing.

Acceptable, low cost printing can be found in most larger cities. Southern California, especially the Los Angeles-Orange County area, seems to offer the lowest cost printing in the U.S. One hundred copies of an original costs $2.12, tax included. Two hundred copies, $3.18. This offset printing process uses an electrostatic process to make a paper plate. Check with local printers to see what is available. Visit their shops and see what equipment they have. If you reduce your paste-ups free on a friendly employers Xerox 7000, you will have a continuous problem of varying quality. Usually the printer

has a camera capable of handling most paste-up sizes and he will make the plate from the paste-up. I want a permanent file master and pay $2.00 per page for a reduced PMT positive print for each page. The printer makes his plates from the PMT. This has the additional advantage of being able to add full size titles waxed to the PMT to give larger headings, etc. It does add an extra day to the production process.

There are many methods of producing the many copies required to mail or distribute to your readers. Offset printing is best, Xeroxing or other photocopying methods, such as the 3M or Savin copier, may be used to obtain two-sided copies. Offset printing is lower in cost and provides better quality than photocopying. Ask the photocopy shop if they have a price and services list. Examine it - most likely it is printed! I once experimented with an issue reproduced on a Xerox 9200; the newsletter was 32 pages and I had 2,000 copies run. The pricing was a little higher than printing; delivery time about the same for collated sets. Quality, however, was inconsistent and I never went back. My friendly printer gives me his odd sheets (the 'extras' a printer prints to insure having enough printed properly on both sides) which make useful material for correspondence, paste-ups, etc.

Collating.

Most printers will collate your newsletter for an extra charge. If the newsletter is a club effort, you can do the collating at the same time you prepare the newsletter for mailing. A bookshelf makes a good collating device, or a homemade collator can be made. Cut slots for 9½"x11½" sheet metal shelves, 1" apart, for a small 'bookshelf' 9" wide. The height of the collator can be about 18" for a 16-shelf capacity. The horizontal sheet metal shelves can be slid into their slots 2" for the bottom, 3" for the next, etc., going up. Five to seven sheets can be collated with about 100 sheets per shelf capacity. A wet sponge moistens the index finger to pull the sheets out, and a quick collation of 300 sets of a 14-page newsletter can be done in an hour.

Printing the first page of the newsletter a different color gives it character and allows many issues to be stacked for storage and easy separation later.

Mailing.

One of the biggest problems is the mailing list and production of labels. I suggest the following scheme to maintain the mailing list and produce the proper labels for each mailing.

Assign each member/subscriber a sequential number. Type his name, number, and address in three or four lines centered on an Avery label sheet, #5351, intended for Xerox copies. Write the member's expiration date in pencil on the lower right part of the label. The 5351 label sheet has 33 1"x2-3/4" labels stuck to a waxed sheet. Reserve the upper-right label for a large number to identify the sheet. When mailing time arrives scan each sheet to find expired members. Peel off their label and place on the back of the sheet. Take your box of master labels to a Xerox machine and have them duplicated onto another set of 5351 labels. Peel and stick the duplicate set to envelopes or self-mailer newsletters at a rate of 450-500 per hour. When a member renews, place his label back on the front. Changes are easily made. Do not make copies in advance; things change too fast. A mailing list of 3,000 has proven no problem in maintaining in this manner.

Another scheme that has worked with older, slower photocopiers, is to photocopy Page 1 which is also a self-mailer and contains the address information. The mailing list is typed on a length of adding machine tape with convenient spacing. Two slits are made in the original placed on the photocopier window. After each copy the tape is pulled through the two slits to show the next address. The sequential numbering of the names allows a quick check that the list was complete. Both methods have been used with success - never does a member not get an issue mailed to him. The Post Office may lose it, but you have one addressed to him each issue.

Avoid sticking stamps if possible. It is a chore because the managers of our postal system have no concept of providing stamps in convenient form. Rolls are reasonable for various forms of mechanization, and go fast manually. Sheet stamps can be stacked five to ten in a stack, stapled at one edge, and cut into strips on the papercutter, leaving the whole sheet still attached (for counting and control) with the cut stopping ½" from the end. Tear off one stack strip at a time and apply using a sponge. The problem is that usually the required amount of postage is not a single stamp in roll form. Foreign postage requires extra time, so allow your extra charges to cover this expense. Send First Class to insure delivery -- remember, a newsletter is timely. Foreign is sent Air Mail. Other Article - and requires a non-sealed clasp envelope.

Obtain a mailing permit if you qualify. A mailing machine is OK if your club doesn't have to buy and maintain it. Mailing permit information, and mailing costs for various classes of mail, is very difficult to get from the Post Of-

fice, but give it a good try. Go to a Main Post Office after calling to make arrangements to talk to someone who handles business accounts. The usual window teller cannot help you much. Permits are reasonable in cost and allow you to print a cancelled logo on the envelope. Most newsletters are marked FIRST CLASS, dated material, and mailed with a permit.

Finances.

The club usually finances the newsletter. Costs for a year's operation are very predictable if a fixed issue size, including attachments (such as a member list, index, etc.),is determined. Allow 20 or 30 issues for Editors exchange and overprint enough for those special packages of back issues an Editor will want to swap with other Editors. You should not have to worry about finances or getting checks to pay the printer. If you do your planning properly before your first issue is printed, you will save considerable time and frustration. Printing costs, label, envelope, and assembly costs will be small compared to mailing costs. Allow for supplies such as tape, rub-on labels, liquid paper, black pens, and blue pencils. Typing can be professionally done without financial difficulty if your mailing list is several hundred. Do not take on a newsletter that can't support itself. Advertisements will help, but for a small operation it won't contribute much and are essentially donations by local businesses.

Conclusion.

Editing and publishing a club newsletter can be a rewarding and educational experience. Most people have no concept how time consuming it can be. The Editor will spend six to ten hours per page per issue if he does not do the typing, and his print density is 1,200 plus words per page with illustrations and photographs. The pressure of working to a deadline can be too stressing for some people. It seems that you relax a few days after crashing out the last issue, then you realize that you are getting late for the next one. Your reward for taking on the task must be self-fulfillment of getting good technical information to your readers and the pride of doing each issue a bit better than the previous one.

There are no special requirements or training that will make you into a good Editor. A certain feel and interest in information gathering and dissemination is required. Editing, writing, and programming are similar; you really don't know if your efforts will be successful and rewarding until you try.

See bottom of Appendix page for footnotes.

APPENDIX A - FORMAL FORMAT 77%

Paste-up sheet layout

```
                    |<----------- 11" ----------->|
                                 (8.5)
        1" (.77)
                                                              ^
   +--------------------------------------------------+       |
   |   +-------------------------------------+         |      ^|
   |   |          MASTHEAD AREA       :  :   |         |      ||
1.2"|  |                              :  :   |         | 1.7" ||
(.92)  |                              :  :   |         | (1.3)||
   |   +-----------------------------+--+----+         |      v|
   |   |                             |  |    |         |       |
   |   | Column is 53 characters     |  | ___v____     |       |
   |   | across @ 12 char./in.       |  | HEADING      |       |
   |   | This is 8.8 words per       |  | -------      |       |
   |   | column. (6 char/word)       |  |   ^          |       |
   |   | Column length is 12 in.     |  |   |0.9"(.69) |       |
   |   | Six lines/in is 72 lines.   |  |              |       |
   |   | Words per column is         |  |              |       |  14"
   |   | 72 x 8.8 = 634.  Words  --->|<-| .5"          |       | (11)
   |   | per page is 1,268.          |  | (.39)        |       |
   |   |                             |  |       .5"--> |       |
   |   |                             |  |      (.39)   |       |
   |   | Note:  Typeset text has     |  |              |       |
   |   | variable spacing and an     |  |              |       |
   |   | average must be used if     |  |              |       |
   |   | a comparison is made.       |  |              |       |
```
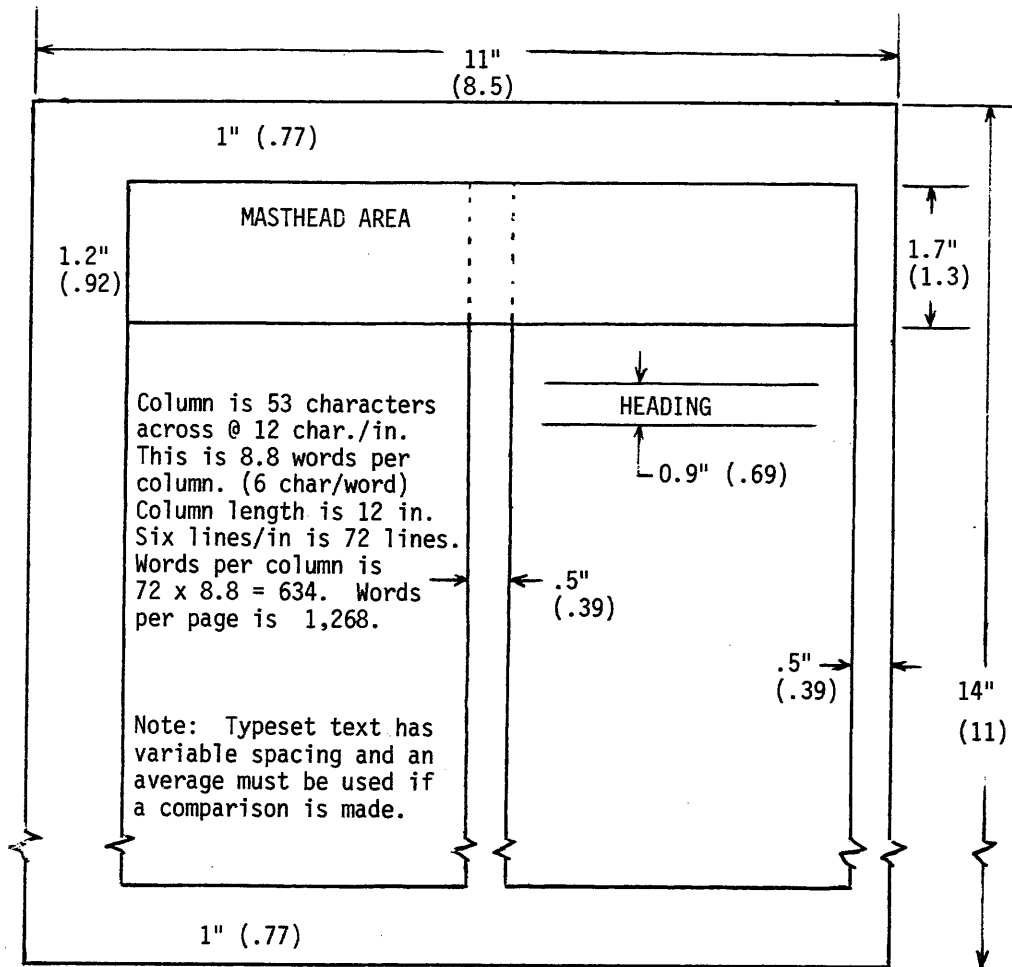
Odd page and front page, even page rotate 180°.
Note: Numbers in parenthesis ( ) are newsletter page dimensions.

This layout uses the reduction shown in table 1 of the text. It is a good
place to start if you are doing your first newsletter. It is effecient in
that 1200 words per page is still in a type size that is readable by most
people. Make notes to formalize your layout to insuure a consistant newsletter.

Footnotes.

(1) The newsletter is the PPC Journal, formerly
     called 65 NOTES, a monthly publication of
     PPC, formerly called the HP-65 USERS CLUB.
     PPC is a dedidated Personal Programmers
     Club for Hewlett-Packard Personal Program-
     mable Calculators. PPC Journal readership
     is nearly 2,000. PPC members contribute
     $15 per year for the programs, programming
     techniques, applications, and hardware modi-
     fications information published in the PPC
     Journal.

(2) Write Paste-Up Supply, 1113 Walnut Street,
     San Gabriel, California 91776, (213) 283-
     4610, for information on waxers and supplies

DEUS EX MACHINA
or
The True Computerist

by
Tom Pittman
P.O.Box 23189
San Jose, Ca. 95153

Several years ago, when the idea of a personal computer was still only a gleam in my eye, I made an observation about my work as a computer programmer. I suppose the same thoughts have occurred in the minds of painters, sculptors, composers and other artists down through the ages. I perceived that I was giving existence to something which had not existed before — I was creating ex nihilo, out of nothing. To be sure, most of the programs I wrote were mere copies or adaptations of other programs, and nothing new in themselves. And, the taxman to the contrary, there was no tangible substance to the work of my hands. But every once in a while I could stand back and look on my work and say, "See what I made!"

I do not wish to quibble at this point with those who claim that nothing is truly a creation. As I said, most of the programs I write are merely copies or adaptations of some other programs. I am not talking about those. Nor do I particularly wish to quarrel with behaviorists or social biologists who reduce every activity of Man to the effects of his environment or his genes. What I am getting at here is the particular feeling that only comes with knowing you have created something new. It is not quite the same as the feeling an expert technician gets in his craft, the sense of doing a job well. I have felt that often, and I continue to take a certain pride (if you will pardon my immodesty) in the high technical quality of my work.

There is a difference between the technician and the artist. The technician is building to an existing pattern or plan; the artist is making a new plan. The technician has a standard by which to measure his work; the artist is his own standard. Of the technician's work you can say "He did (or he did not) meet the requirements of the specifications"; of the artist's work you can only say "Ahhh!" or "Yecch!"

I wish the boundary between the technician and the artist were that clear-cut. It is seldom so. What the artist creates is, whether he likes it or not, subject to technical criteria. If he is painting a portrait or a landscape, you can apply the purely technical judgements to it of whether or not it adequately conveys an image of the subject. You can even determine if the paints have been mixed correctly, or if they are likely to deteriorate and change color with age. Is the perspective and lighting believable? Into this technical fabric, however, is woven the art, the quality that makes Dürer great and Cranach so-so.

I am a programmer, not a painter. It is much harder to see the "art" in a computer program. Donald Knuth sees it and the title of his monumental work on programming is "The Art of Computer Programming" [1]. I only hope nobody asks me to point to some program and say "this is art, not technique." Perhaps I am a coward and lack the fortitude to defend such an assertion. Perhaps there is no defense and I dislike making indefensible assertions. No matter. My point is that the assertion can be made, and that it has meaning (at least for most of us).

I raised the issue in connection with a certain feeling I got as I reviewed my work, when I saw it as a creation. You see, in that instant I as a Christian thought I could feel something of the satisfaction that God must have felt when He created the world: "And God saw every thing that he had made, and, behold, it was very good." [2] If man is, as Christians believe, created "in the image of God" [3], then perhaps I had learned something about God. In this I have a definite advantage over the painter and the composer: I can create something that will interact with me, as man interacts with God. So far it is a strictly intellectual interaction, and for the most part very predictable — my creation does what I programmed it to do, which is (usually) what I had intended. I consider my insight to be only a pale reflection of the devine, but...

Yet in my relationship to the computer and to the programs I write for it there is another dimension, wherein lies a very grave danger. The danger is that I will lose my sense of perspective, and forget the relationship between God, myself, and the computer.

Theodore Nelson in his popular book, Computer Lib, refers to a "computer priesthood." [4] By this he means that the computer technology has built up around itself a kind of mystery religion or gnosticism, with the computer professionals acting as the priests of that religion. Gnostic religions in history have had a body of secret knowledge (the word "gnostic" is derived from the Greek word for "knowledge") which only the insiders have access to. Just from a technical point of view this was, and continues to be, a very real problem. The use of computers in our time requires such a heavy technological background that outsiders are locked out.

To be introduced into this computer gnosticism has, until only recently, required skill with a soldering iron (not just "Which end is hot?" but the proper way to apply solder to microelectronic circuits, special soldering tools, etc.), understanding of how to read resistor codes and the cryptic markings on integrated circuits, proper work habits for protecting delicate MOS components from static electricity, and the ability to decipher inadequate instructions and third-generation xerox drawings. With a small but increasing number of exceptions, the novitiate computerist is required to know (but is not given instruction in) binary, octal, and hexadecimal number systems, machine language programming, real-time I/O control, ASCII code translation, and memory management. Unless he is willing to remain in the outer circle playing games that someone else wrote, the new gnostic is compelled to learn a foreign language something like Latin (BASIC) and be able to construct in that language esoteric hymns called Loops, Subroutines, Conditionals, Assignments, Input/Output and Data statements. I can assure you that the language requirement is not going to disappear for many years. We may get new languages, but that will only mean that the insider must know more, not less.

Until the advent of personal computers in 1975, Computerism was successfully restricted to the elect who went through the necessary training and were employed in the computer departments of those corporations, educational and government institutions rich enough to be able to afford them. It was a closed society. With the advent of

the truly personal computer the ranks have been opened up to admit something over 100,000 new converts, but the careful observer will notice that it is still a closed society. The outsider is not really given much reason or help to join. Almost all of the magazine articles are directed to intermediates, not beginners. There are a very few books aimed at the novice (but not the totally uninitiated!), but they tend to get lost in the vast majority of books for the more sophisticated. By contrast, the number of Christian books aimed at the novice and uninitiated far exceeds those that require a significant background in theology. Computerism is still a closed society, though much larger than it was five years ago.

So far I have been talking only about the phenomena of gnostic computerism, the appearances of the society. There is a deeper level that is much more serious. It is where we, the practitioners of this arcane art, begin to believe as we act. When we actually come to depend on the Computer to solve all our problems and to resolve even the mysteries of life, we have taken the final step toward making the Computer our god. At this level, opening up the secrets of computerism to all comers makes no difference at all; whether the Computer is a gnostic god or an evangelistic god is of little consequence, because we are talking about individual attitudes towards the machine. That is you and me, not "us" or "them." Institutions are formed from the aggregate of individual attitudes and beliefs.

Before I get into what the Computerist believes, let me say something about the nature of belief. All of us prefer honesty over deceit (at least in the other person), kindness over malice (when all other things are equal), and so on. In the words of Mammy Yokum, "Good is better than evil, because it's nicer." [5] But when it gets down to cases, with two of you out on the raft in the middle of the Pacific and food for two days, the choice of who eats the food and who dies of starvation depends on what you really believe. Or closer to home, it's rush hour and you are late to work; there is a long line of cars behind yours and you come to an intersection where a little old lady wants to cross the street you are on. This is where belief affects our lives. Your faith — your religion, if you will — is what makes the decision when it could go either way, but for different reasons. Notice here that I am not using the term "religion" to mean the religious institutions of society. Many of these function only as social institutions with little or no effect on the lives of the adherents. Instead I use the definition that equates religion with whatever is foremost in a person's thoughts and actions. In this sense everyone has a religion: some of us are Christians. Others are Egoists, "Moneyists", Scientists, Marxists, Sexists or Computerists. The focus of our attention is our god.

So what is it that the pious Computerist believes? Which way will his decisions go, when it gets down to the crunch? Let me list a few "articles of faith" that affect the every day life of the practicing Computerist:

1. The computer is more interesting than most people. I love to spend time with my computer. It is fun to write programs for it, to play games on it, and to build new parts for it. It is fascinating to try to figure out what part of the program it is in by the way the lights flicker or the radio buzzes. It beats dull conversation any day.

2. It is most important to be sure the computer is properly taken care of. When it finishes its present task, I must drop everything and go start up its next task, or turn off the disk drives to save wear and tear on the moving parts. If there is a power failure, first go shut down the peripherals (save the disk, turn off the paper tape punch, etc.), then come back and get a candle to light up the house.

3. The computer will be a big benefit in all kinds of ways. It is not connected up yet, but "soon" it will control the sprinklers, serve as a fire/burglar alarm, maintain the Christmas card mailing list, control the stereo tape deck, monitor the central heating, maintain the inventory in the pantry, provide menu planning, remind us of important dates, write form letters to people we don't like, educate our children, and so on. The key concept is that all these things are in the very near future. The computer has not yet fulfilled these promises, but it will very shortly now.

4. The computer needs just a little more (memory) (speed) (disk

space) (peripherals) (fidelity in its cassette drive) (better BASIC) (newer CPU) (noise suppression on the bus) (debugging on this program) (powerful editor) (bigger power supply) before it can do this or that.

5. The computer can make money for us on the side, and eventually it will pay for itself.

6. Spending all this time on my personal computer will give me job skills that will improve my wage-earning ability and make me eligible for a promotion.

7. The computer can be used in the company business to improve profitability.

8. There is no need to buy this software package or that circuit board; I can design one better.

9. Let's arrange our next vacation to coincide with the Computer Faire. Wow! What a way to spend a vacation! Then we can swing by these manufacturers and/or these stores and see the latest widgits.

10. To stay on top of the field it is necessary to subscribe to all five of the "Good" computer magazines. The other six are merely repetitious or uninspired or are beholden to their advertisers; they are not worth the subscription price. But when a good one comes sit down immediately and do not get up until I've gone through the whole magazine.

11. Never miss a club meeting. This is where it's at. The juicy little news bits, the how-to-fixits for the problem that has been bugging me for the last two weeks, hearing about the interesting things that I can do with my computer — that is the real thing! Besides, they might have some free software.

12. Visit the local computer store at least once a week. They are always getting new hardware in, and sometimes you can pick up some good rumors or a new book. You never know when you might bump into an Interesting Person at the store.

None of these claims, by themselves, are particularly wrong or indicative of misdirection. But taken as a whole, they reflect an attitude that the computer is, in Paul Tillich's words, "the ultimate concern." [6] The computer becomes the object of one's devotion, the provider of one's needs. The computer has become the Absolute, the god in one's life. It is the work of our hands and the image of our minds; are we going to let it become the Lord of our lives?

I would like to take exception to some of the "articles of faith" listed above. I have been there and I know the attitudes behind them. But I also know what is wrong with some of them.

1. Clearly the computer is a fascinating device. Its complexities are overwhelming. I said this at the beginning, and I do not deny it now. But being the product of our own imagination, the computer probably cannot exceed our own intelligence. It may be faster and more accurate. It may do some things (like play chess) better because of this greater speed and accuracy, but it can never give us true interraction on a human level. I realize I am going out on a limb in saying this; roboticists will gladly point to claims that humans would never leave the surface of the earth. Those claims were obviously wrong; I may be wrong also. But I can point to the difference between a technician and an artist with which I began this essay: We may be able to build robot technicians, but not robot artists. In any case it is unlikely to happen in your or my lifetime. The computer is a tool, and we should recognize it as such.

2. The computer is a delicate machine and as such it requires care and maintenance. It is relatively expensive (today) and abusing it is not economically reasonable. But it is still a machine; it is not as important as any human being.

3. The computer is capable of many kinds of benefits. But honestly now, how many of those things do you think you will actually get your computer to do? How many people do you know or have you heard of whose computer actually does those things, or even just some of them? Have you considered the transducers and mechanical linkages required to give the computer a meaningful selection over your music collection? Do you have any notion of the software effort needed to implement a computerized calendar or heating control (I mean beyond what is more easily done without a computer)? Have you ever stopped to think how much manual effort is required to

maintain a computerized pantry inventory? Have you considered the massive data entry requirement to build a data base for a decent menu planner? If educators and computer professionals working with large government grants cannot make much headway in Computer Assisted Instruction, are you going to master it in your spare time? Don't get me wrong. Many of these things are practical goals for computer implementation, but most of us, working on it as a hobby, will not get very many of these exciting applications working in any meaningful way.

4. In the microcosm, the need for a little more of this or that for the computer seems very reasonable. A few years ago the Sunday supplement of some newspaper reported a survey on the money wishes in America. The result was that the average American would be happy if he or she had $13.21 more. Parkinson's Law holds that expenditures will always meet or exceed income. The same law applies to computer memory, speed, peripherals, and so on.

5. Several of the magazines are touting the economic rewards potential in the personal computer. They are wrong. Suppose you did get a little extra money on the side computing bowling handicaps. What is to stop the local bowling alley from seeing the profit potential and buying their own computer? Maybe you will sell a neat game to a national distributor, but how many others are pushing games to the same distributors? Anyway, have you invented any neat games? Obviously some computers pay for themselves (mine does), but far more only promise to do so.

6. Right now there is a shortage of people with microprocessor experience. Five years ago there was a shortage of COBOL programmers, but there is a glut now. People with drive and dedication, who make themselves valuable to their company, have no trouble finding work. People who stay up late nights on their own projects and are too tired to give the boss an honest day's work, who spend hours on the telephone ordering parts on the company bill for their personal computers, will find trouble finding and holding any kind of job.

7. Yes, computers have improved the profitability of some companies. More often they have only promised to do so. You do not replace a bookkeeper with a computer; you give her a raise and call her a computer operator. It is still some time before we will see much business software available and usable.

8. This one is subtle. Of course you can design one better. The computer is above all things an optimist's machine. But you won't. You don't have the time to get around to it, or it seems to have some problems when you do get it built: it never seems to work exactly the way you planned.

9-12. By now the computer has moved out of the den and into the rest of your life. It will consume all of your spare time, and even your vacation, if you let it. It will empty your wallet and tie up your thoughts. It will drive away your family. Your friends will start to think of you as a bore. And what for?

A few years ago I was doing some technical writing for a major electronics firm, and I had described some control circuit in terms such as, "the device provides such-and-such functions..." One of the people assigned to review my work reprimanded me: "Only God 'provides', circuits just..." I can no longer recall the exact details of the exchange, but I have not forgotten the message. God provides. Electronic circuits in general, and computers in particular, are not God; they provide nothing. They may be works of art, a beauty to behold, but in the final analysis, they are tools and they perform certain functions at our command. If we forget that computers are only tools, perhaps we will also forget that people are not tools. When we know Who is God, we also know who we are, and what computers are. In the words of the Second Commandment,

Thou shalt not make unto thee any graven image, or any likeness of any thing that is in heaven above, or that is in the earth beneath, or that is in the water under the earth: Thou shalt not bow down thyself to them, nor serve them: for I the LORD thy God am a jealous God. [7]

References

[1] Donald E. Knuth, The art of computer Programming. Reading, Mass: Addison-Wesley 1975.
[2] Genesis I 31.
[3] Genesis I 27.
[4] Theodore H. Nelson, Computer Lib, p. 2. Chicago: 1974.
[5] Al Capp, Lil Abner Sunday comics approx. 1974.
[6] Paul Tillich, Dynamics of Faith, New York: Harper 1957. Most of this book is nonsense, but Tillich does have a good understanding of what constitutes idolatry.
[7] Exodus XX 4,5.

PEOPLES' CAPITALISM
THE ECONOMICS OF THE ROBOT REVOLUTION

James S. Albus

© New World Books
4515 Saul Road
Kensington, Maryland   20795

Abstract

Where are computers taking society?

Will industrial and business robots lead to:

Orwellian dictatorship?

Jeffersonian democracy?

A new aristocracy based on robot labor?

Who will own these machines, and who will control the economic wealth and political power they will create?

The great challenge of the coming industrial revolution will be to develop an economic system wherein prosperity can be achieved without waste, affluence can be made compatible with the limits to growth, and personal freedom can be preserved and enhanced in a world where most economic wealth is created by automatic machines.

Peoples' Capitalism is a plan for meeting this challenge. It is a formula for a new economic order which might best be described as Jeffersonian democracy for the post-industrial era.

The details of this plan will be described and a program for political action will be presented whereby Peoples' Capitalism could be introduced into any country in the world by the year 2007.

Epilogue to Scarcity

These are revolutionary times. Changes as profound as those resulting from the invention of agriculture or the domestication of wild animals are rushing us toward a new world. The human race is now poised on the brink of a new industrial revolution which will at least equal, if not far exceed, the first industrial revolution in its impact on mankind. The first industrial revolution was based on the substitution of mechanical energy for muscle power. The next industrial revolution will be based on the substitution of electronic computers for the human brain in the control of machines and industrial processes.

From the beginning of human existence, mankind has lived under the ancient biblical curse: "By the sweat of thy face shalt thou eat bread, till thou return unto the ground." Before the invention of the steam engine, virtually all economic wealth was created by the physical labor of human beings, assisted only by their domestic animals.

The first industrial revolution only partially lifted the ancient curse. Yet, even this partial reprieve had profound consequences. In all the thousands of centuries prior to the first industrial revolution, the human race existed near the threshold of survival, and every major civilization was based on some form of slavery or serfdom. Yet a mere two centuries after the introduction of steam power into the manufacturing process, slavery has become little more than a distant memory for the citizens of every major country. Today, a large percentage of the population of the world lives in a manner which far surpasses the wildest utopian fantasies of former generations.

There is good reason to believe that the next industrial revolution will change the history of the world every bit as profoundly as the first. The application of computers to the control of industrial processes will bring into being a new generation of machines; machines which can not only create wealth unassisted by human beings, but which can even reproduce themselves at continuously decreasing

costs. The potential long-run effects of this event are twofold: First, it will allow man to free himself from the dehumanizing demands of mechanization. The self-regulating capacity of computer-controlled industries will render it unnecessary for people to structure their lives around daily employment in factories and offices. The first industrial revoltion drew people away from the land and concentrated them in urban industrial communities. The robot revolution will free human beings from the pressures and congestion of urbanization and allow them to choose their own lifestyles from a much wider variety of possibilities.

Second, the introduction of the computer into manufacturing has the potential for removing material scarcity from the agenda of critical human problems. The technical feasibility of factories and industries which can operate unattended and reproduce their own essential components implies that manufactured goods may eventually become as inexpensive and unlimited by process complexity as the products of biochemical mechanisms in living organisms. Increased efficiency and flexibility of substitution between materials and processes could render currently projected shortages of fuel and materials largely irrelevent to the 21st century.

Unfortunately, the present economic system is not structured to deal with the implications of a robot revolution. There presently exists no means by which average people can benefit from the unprecedented potentials of the next generation of industrial technology. Quite to the contrary, under the present economic system, the widespread deployment of automatic factories would threaten jobs and undermine the financial security of virtually every American family.

This book is an attempt to address some of the fundamental problems of income distribution and capital ownership in a society where most of the goods and services either are, or could be, produced by machines rather than people. It questions the adequacy of conventional economics for the present, as well as for the future. It argues that the primary cause of the recent economic crisis is not a lack of resources or insufficient wealth-producing capacity but an unrealistic view of how wealth is created and an outmoded system of incentives which does not make use of what is available to produce what is needed.

I claim that, if we properly utilized our scientific knowledge and our industrial capacity, we could not only overcome the present economic crisis, but we could go on to eliminate poverty altogether and guarantee personal financial security to every individual. Furthermore, this could be done in a manner compatible with a clean environment and an ecologically balanced world.

The great challenge of the coming industrial revolution will be the development of an economic system wherein prosperity can be achieved without waste, affluence can be made compatible with the limits to growth, and personal freedom can be preserved and enhanced in a world where most wealth is created by automatic machines. This paper is an attempt to formulate a plan by which this could be accomplished. The proposals contained in the following paragraphs might best be described as a formula for Peoples' Capitalism, or as a blue print for Jeffersonian Democracy in a modern technological society.

Even the most casual observer of what goes on in the average factory, office, or construction project cannot help but notice that most of what is produced comes from machines and not human labor. Scholarly studies confirm this common sense observation, showing that the overwhelming percentage of productivity increases over the past two hundred years have resulted from technological progress, not from harder work or longer hours. Whether we like to admit it or not, most of what we have is produced by machines, not people.

Does it not then seem odd that more than two-thirds of our total output is distributed as compensation to labor? Might not such a large discrepancy between how wealth is created and how it is distributed distort the entire structure of the free market economy? Consider, for example, that distributing the benefits of two centuries of productivity increases primarily through wages and salaries has increased labor costs so high that employers cannot afford to hire workers even when there are jobs which need doing. Thus, we have massive unemployment even while:

*Cities need rebuilding,

*New sources of energy need to be developed,

*Pollution control needs to be expanded,

*Better health care delivery needs to be provided,

*The environment needs to be protected and services of every kind need to be improved.

Yet no one can afford to hire people to do these jobs.

On the other hand the lack of any alternative to wages and salaries as a source of income creates such strong pressures for job security that:

*Waste is encouraged,

*Featherbedding and restrictive work rules are commonplace,

*Pollution is condoned,

*Obsolescence is planned,

*Mass advertising of trivia is considered necessary,

*Unwise growth goes unchecked.

And much of what people get paid for doing everyday in offices and factories throughout this land could be eliminated without affecting the production of goods or services whatsoever.

Furthermore, even if make-work and waste could succeed in producing "full employment", there would still be millions of Americans outside the wage and salary income distribution channels. The employable labor force makes up only about 40 percent of the population. Thus, even though practically every business in the country could easily expand production, and would gladly do so if markets were available, the lack of purchasing power of people without jobs makes such expansion impossible. The result is that our enormous productive potential is never fully applied to our clear and urgent human needs.

Rising Expectations vs. Declining Resources

The two following charts show that long-term gains in productivity are closely correlated with investment. There is little reason to doubt that this correlation will continue into the future. In fact, recent technological developments

in computers and manufacturing technology suggest that mankind may be on the threshold of a new industrial revolution. Within two decades it may be practical for computer-controlled factories and robots to produce virtually unlimited quantities of manufactured goods, and to even reproduce themselves at continuously decreasing costs.
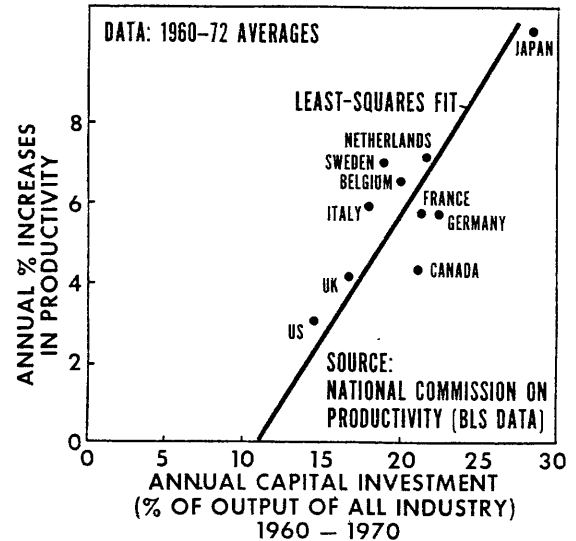
Figure IV-4. What causes a nation's productivity to grow? This chart shows that countries with a high rate of investment have high productivity growth, and vice versa. This implies that productivity growth is not serendipitous or beyond human control. Instead, it is the direct result of economic policies which promote investments in new technology and in more efficient plants and equipment.

CAPITAL INVESTMENT PER MAN-HOUR & OUTPUT PER MAN-HOUR (PRODUCTIVITY)
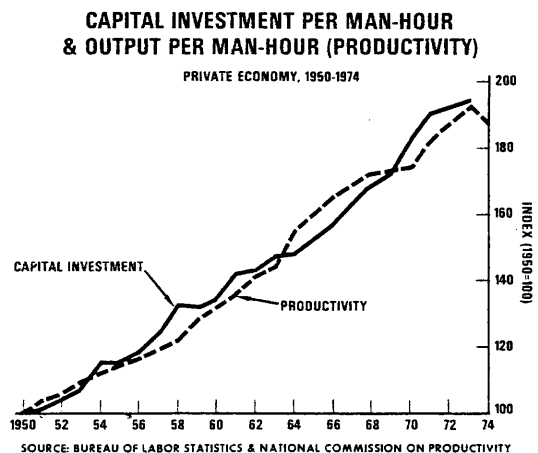
Figure IV-5. Productivity (i.e., output per man-hour) is closely correlated with the amount of sophisticated tools and capital equipment per worker. The data shown here, together with that in Figure IV-4 strongly imply that U. S. productivity could be increased by increasing the capital investment rate.

Investments in such technology almost certainly will produce major productivity gains for many decades to come.

Unfortunately, the present mechanisms for increasing investment in high technology industries (such as tax credits for big business) serve to increase the already enormous concentration of wealth and power in the hands of a few (A U.S. Department of Commerce Survey of Current Business report on stockownership in the United States dated November 1974 states that one percent of the families in the U.S. owns over 50 percent of all stock and that 5 percent of the families own 73.7 percent by value of all stock.), while leaving the majority of the population just as dependent on wages and salaries as ever. Clearly robot factories will be a direct threat to the economic security of almost every American unless some alternate means of investment financing and income distribution can be found.

Peoples' Capitalism is a proposal which addresses all of these issues simultaneously. It offers a simple, straightforward solution to each. Peoples' Capitalism could be instituted in the United States without any changes in our constitutional form of government. In fact, far from altering any of the fundamental principles upon which this country was founded, this plan would revitalize the free enterprise system, and realize the ideals of Jeffersonian Democracy in post-industrial America.

Specifically three new institutions are proposed:

1. A National Mutual Fund (NMF) is suggested to finance capital investment for increasing productivity in socially beneficial industries. The NMF would be a semiprivate profit-making investment corporation which would be authorized by Congress to borrow money from the Federal Reserve System. It would use this money to purchase stock from private industry for the modernization of plants and machinery and the introduction of advanced computer-based automation. Profits from these investments would be paid by the NMF to the general public in the form of dividends. By this means, the average citizen would receive income from the industrial sector of the economy quite independently of employment in factories and offices. Every adult citizen would become a capitalist in the sense of deriving a substantial percentage of his or her income from dividends paid on invested capital.

2. A Demand Regulation Policy (DRP) would be instituted in parallel with the NMF in order to provide sufficient savings to offset NMF investment spending. This would prevent short-term demand-pull inflation. The DRP would withhold income from consumers by mandatory payroll deductions and convert it into high-interest five-year savings bonds. Deductions would be graduated according to income (low-income persons would have little withheld, high-income more) and would be adjusted monthly according to a formula based on the best available indicator for inflation. The DRP would allow high rates of investment and the accompanying high employment and high production while preventing excess demand from forcing prices upward.

3. A Federal Department of Science and Technology is also suggested to focus modern technology more directly on problems relevant to human needs.

Peoples' Capitalism is simple in concept yet its implications are truly breathtaking.

> *It offers a solution to recession and inflation simultaneously.
>
> Increased availability of investment capital would get the economy moving again. Inflation would be controlled in the short run by DRP savings, and over the long run by increased productivity resulting from higher rates of investment.
>
> *It resolves the fundamental conflict between economic growth and environmental preservation.
>
> NMF dividends to individuals would reduce the pressures for jobs and growth at any cost. Increased efficiency in production would reduce waste and provide the resources for improved pollution control and environmental conservation.
>
> *It promises a degree of individual freedom based on personal

financial independence which is unprecedented even among utopian proposals.

Every adult citizen would possess a personal source of independent income. People would be economically free to live where they wished and to work at what they enjoyed.

*It offers a cure to poverty and old age security without taxing the rich to give to the poor.

NMF public dividends would be generated by wealth producing capital investments. The NMF would be a profit-making, income-producing investment corporation, not a tax-consuming welfare program.

*It achieves economic equity without destroying incentives to individual excellence.

The NMF would distribute the profits generated by high technology industry to everyone. Persons with ambition could afford to develop their talents, and society would be able, with clear conscience, to reward its high achievers.

*It opens up an entirely new road to economic development for emerging nations which completely by-passes the social dislocations of classical industrialization.

In developing nations, Peoples' Capitalism could finance automated industries and robot factories which would pay dividends to farmers and villagers directly. Economic development would be achieved without converting the rural population into industrial workers or concentrating them in congested urban areas.

PEOPLES' CAPITALISM: The Economics of the Robot Revolution is published by NEW WORLD BOOKS
     Kensington, MD 20795

                    $3.95 paperback

It is distributed by:

ADVANCED TECHNOLOGY RESEARCH ASSOCIATES
P.O. Box 456
Minneapolis, Minnesota   55440

                    and

MARKETLINE SYSTEMS
2337 Philmont Avenue
Huntington Valley, Pennsylvania 19006

THOUGHTS ON THE PROSPECTS FOR AUTOMATED INTELLIGENCE

by Dennis Reinhardt
DAIR Computer Systems
870 Garland Dr.
Palo Alto, CA 94303
(415) 326-1534

"... I have always discovered after the fact that, if anything, we didn't plan big enough.
I did not forsee the size of General Motors ..."

- Alfred P. Sloan, Jr.

SUMMARY

Some of the technical problems involved in producing a small computer with the hardware capacity of the human brain are explored. Starting with assumptions differing by two orders of magnitude concerning the capacity of the brain and by 25% for the long term growth rate in semiconductor chip density, it is hypothesized that the technology for manufacture of a chip or small system having the hardware capacity of the human brain can be projected for the period 26-48 years from now.

THE BRAIN

The measure which is most useful in comparing the capacity of the human brain with the capacity of the computer is information content, measured in bits or bytes. As more architectural information emerges from brain research, other more useful measures might be suggested. Carl Sagan (1977), quoting data from the work of Britten and Davidson (1969) has placed the information content of the human brain at 10^{13} bits(1). Isaac Asimov (1963) placed the figure at 10^{15} bits(2). Using the definition that 8 bits is 1 byte, we obtain estimates for brain capacity of $1.2*10^{12}$ to $1.2*10^{14}$ Bytes (abbreviated "B").

Since most personal computer systems have 4,096 to 65,536 B of high speed memory with 16,384 B being typical, we estimate the capacity difference between the two, to be $7.6*10^7$ to $7.6*10^9$.

Thus, one measure of the hardware gap between the human brain and a personal computer suggests it is impossible to ever achieve equality. Marilyn Ferguson, writing about discoveries in brain research, states: "A computer sophisticated enough to handle the functions of a single brain's ten billion cells would more than cover the face of the earth."(3) While such a computer would be massive, it is not true that the surface of the earth would be covered. A 3330 type disc contains approximately 300 MB of data and occupies about 2+ sq. meters, after allowance for access space, at a cost in the area of $50,000. Thus, a $1.2*10^{12}$ B system would need 4000 (!) disc drives and might occupy 8,000+ sq. meters (80,000 sq. ft.) and cost over $200 million. More exotic technologies permit cost and space requirements to be reduced substantially.

Without pursuing the issue of todays hardware cost any further, when might this kind of configuration reach a personal computer size, contained on an integrated circuit or small packaged system? The human brain has been evolving for millions of years while the computer has existed for only a few decades. Considering the enormous capacity gap between the two, as measured by the ratio of their byte storage capacity, what are the prospects for evolution of the computer over the next few decades?

HARDWARE

It has been the semiconductor industry which has paced the state of the art in computer technology with the development of increasingly complex circuits on a single integrated circuit chip. The chairman of Intel Corp., Robert N. Noyce, has said that "... the number of components per circuit in the most advanced integrated circuits has doubled every year since 1959, when the planar transistor was developed." If we were to extrapolate this rate of growth forward in time, we would have a low estimate of when "brain"

hardware could be produced. To do this, we solve:

$$2**N = 7.6*10**7$$

for N to obtain a value for N of 26 years. Thus, if the miniaturization trends of the last 18 years continue for another 26 years, semiconductor technology could be at the point where personal computers with a hardware capacity equal to the human brain in terms of number of bits could be produced.

Turning to another source, the chairman of Texas Instruments, Mark Shepherd, Jr., has presented data which exhibits a growth rate closer to 60% per year rather than 100% (5) and extrapolates his data forward for the next 20 years. If we use the 60% growth figure and the higher value for the ratio of brain capacity to today's personal computer, we can project a high figure by solving:

$$1.6**S = 7.6*10**9$$

for S, to obtain a high value of 48 years. In other words, we obtain a range of estimates for the brain computer of 26-48 years, depending upon growth rates in semiconductor complexity and brain capacity used for the calculation.

This range estimate is also subject to the caveat that the growth in complexity might slow or halt altogether, depending upon technical problems relating to ultimate physical limits, market resistance to processor chips which are software incompatible with previous generations, or escalating cost of the new production facilities. Also, neither of our quoted sources extrapolated the data as far as we have.

Determination of the required word size is another hardware problem which would need to be solved. The expression for the word size using our 1.2*10**12 Byte brain size is seen to be ln(1.2*10**12)/ln(2) = 40.1 bits. Evaluating the larger brain size yields a figure of 46.8 bits. Thus, about 40 to 47 bits minimum are needed to duplicate the addressing within the human brain. The implication for the brain computer is that it have a 64 bit, or similar instruction word length since there would be extra bits required in the instruction word for opcode, address mode, etc. At present, the standard

for the personal computer market is the 8 bit processor. The 16 bit processor probably has no more than 10% of the market. Shepherd had predicted the 128 KB 32 bit microprocessor for the mid 1980's. If such processors become available at that time, it is likely that boards or board sets with that capability could be introduced sooner since the microprocessor chip can be developed independently of the memory.

SOFTWARE

If indeed the hardware problems can be solved, how would the software for a computer of this capacity be developed? Using standard estimating techniques and assuming that those 1.2*10**12 B in the "brain" computer are produced by programmers working at an average productivity of 100 - 1000 instructions (64 bit word) per month (6), we find that between 12 million and 120 million man-years of programmer effort are required By way of comparison, the emergence of man on this planet is thought to have occurred less than 10 million years ago. Even worse, other studies suggest that programming effort rises exponentially with the size of the programming problem with an exponent of 1.5 (7).

We are forced to conclude that a "brain" computer will never be programmed in this fashion. Given this conclusion, what relaxation of assumptions makes it conceivable? Certainly, one way it could be done is if we relax the assumption that all 1.2*10**12 B have to be programmed in by a programmer. For example, if the programmer were able to specify the first 3*10**6 B and the computer learned the rest on its own, then we have reduced the task several orders of magnitude. Probably none of us knows the number of bytes required for a computer to be able to organize almost all of the information coming into it without programmer modification over a period of years. None of us will know until someone does it.

Even scaling the problem down to 3*10**6 B gives software development time estimates between 31 and 310 man-years. At present, continuous speech recognition systems require 4.5*10**5 to 9*10**5 B. (9) and are running only on fairly large computers such as the PDP-10. An overall estimate of 3*10**6 B implies that continuous speech recognition is about 10+% of the entire

job in a brain computer. But again, no one knows.

It might be argued that we have unnecessarily concentrated on programmer entry of algorithms when the bulk of the brain information might be made up of the "data base". This might be the crucial distinction between our view of automated intelligence and the state of the art in present software systems. In present practice, the "data base" is operated upon statically by the computer system. It makes no difference to the algorithms over time what the range of data entered is, how often it is entered, or its source. One unchanging computation fits all cases until the programming staff makes another release. We forsee an intelligent system modifying its behavior based upon the data it experiences. This means that the fundamental algorithms used in processing must be capable of evolution under the control of the computer itself. Part of the evolution can be controlled by the software manufacturer, but much of it must be local to the machine and its owner in order that the computer remain intelligent within its surroundings.

Within the "hobby" subculture of the personal computer market, software development takes place outside the confines of an individual's organizational affiliation. Often there is more creativity exhibited on one's own work than in the work done for an employer. Furthermore, there seem to exist methods with lag times of 3-6 months for transmitting software through most of the hobby market. With an estimated 50,000+ systems sold and somewhat less actually operating, there exists a large pool of people who could participate in developing a "brain" computer. It is already considered true by some in the personal computer industry that it leads other, more established segments of the computer industry in terms of accomplished innovation. It seems likely that progress will be made in the process of automating the intelligence acquired by man and that significant contributions can potentially come from the hobbyists in the personal computer market. Already some products have been introduced which automate some human like functions with modest memory requirements of 8,000-48,000 B. Examples are isolated word discrimination (9) and computer controlled speech (10). As this industry comes to understand how to implement functions concisely, the enormous software cost estimates for developing a computer brain can be lowered closer to practicality.

IMPLICATIONS

Having considered some of the hardware and software development problems to be solved in developing the brain computer, let us raise an issue suggested by this investigation: if only the first $3*10**6$ B of the computer are programmed and the remaining $1.2*10**9$ B learned and organized by the machine, then in what sense are the capabilities of the machine "predestined". Well over 99% of the machine's information has entered as a result of interaction with the environment. It might well be that the only practical way to develop a brain computer is for it to be self determining.

CONCLUSION

A personal computer with hardware capacity comparable to the human brain might be possible 26-48 years from now, and its near term ancestor, the 32 bit microcomputer can be forseen within the next few years, before the end of the 1980's. The software does not yet exist and it is possible that the cost of producing it might be prohibitive unless we are able to implement innovative approaches to a programming effort of this magnitude.

REFERENCES

(1) THE DRAGONS OF EDEN: Speculations on the Evolution of Human Intelligence. Carl Sagan. Random House, Inc., 1977.

(2) THE HUMAN BRAIN: Its Capacities and Functions. Isaac Asimov. Mentor Books, 1963.

(3) THE BRAIN REVOLUTION. Marilyn Ferguson. Bantum Books, Inc., 1973.

(4) "Microelectronics". Robert N. Noyce in SCIENTIFIC AMERICAN, Vol. 237, No. 3, pages 63-69; September, 1977.

(5) "Distributed Computing Power: a Key to Productivity". Mark Shepherd, Jr. in COMPUTER, Vol. 10, No. 11, pages 66-74; November, 1977.

(6) "The Cost of Developing Large-Scale Software". Ray W. Woverton in IEEE TRANSACTIONS ON COMPUTERS, Vol. C-23, No. 6, pages 615-636; June, 1974.

(7) THE MYTHICAL MAN MONTH. Frederick P.
Brooks, Jr. Addison-Wesley, Inc., 1975.

(8) "An Assessment of the Technology
of Automatic Speech Recognition for
Military Applications:. Bruno Beek,
Edward P. Neuberg and David C. Hodge in
IEEE TRANSACTIONS ON ACOUSTICS, SPEECH,
AND SIGNAL PROCESSING, Vol. ASSP-25, No. 4,
pages 310-322; August, 1977.

(9) "Introducing SPEECHLAB - The First
Vocal Interface for a Computer". Horace
Enea and John Reykjalin in POPULAR
ELECTRONICS, Vol. 11, No. 5, pages 43-
50; May, 1977.

(10) "Friends, Humans and Countryrobots
Lend Me Your Ears". D. Lloyd Rice in BYTE,
issue 12, pages 16-24; August, 1976.

BRAIN MODELING AND ROBOT CONTROL SYSTEMS

James S. Albus

New World Books
4515 Saul Road
Kensington, Maryland 20795

Abstract

The inventor of the neurophysiological model Cerebellar Model Arithmetic Computer (CMAC) which won in the 1976 Industrial Research Magazine IR-100 competition describes his work in brain modeling and robot control. CMAC demonstrates the capacity to learn, generalize, recognize patterns, perform associative recall, compute multivarient analog functions and decompose input commands into sequences of output commands in a context sensitive manner. Methods of implementing this model on a microprocessor are discussed.

Evidence is given that clusters of neurons with such properties are arranged in hierarchies in human brains so as to produce AND/OR task decompositions. At the lowest levels in the motor system these clusters transform coordinates and compute servo functions. At middle levels they decompose input commands into sequences of output commands which give rise to goal directed behavior patterns. Mechanisms by which feedback can alter these decompositions to compensate for perturbations and uncertainties in the environment are described. At the highest levels of the hierarchy are goal selecting and evaluating mechanisms which are used for planning and problem solving.

The possibility of implementing such a hierarchy on a network of hobby computers is discussed.

Computation by Table Look-up

CMAC is a computing device which accepts input variables and produces an output which is some function of the input variables. There may be up to twelve input variables which can be either continuous or binary in any combination. CMAC's internal operations are entirely digital and its output is a digital number which may, of course, be converted to an analog voltage.

CMAC computes by transforming each input variable into a set of intermediate variables which are combined to select a set of weights. These weights are then summed to produce an output. The CMAC output is thus a function of the input variables, and the value of the output for every possible configuration of the inputs is determined by the weights.

The computation of mathematical functions by table look-up is, of course, a commonly used technique for functions of from one to three variables. For example, trigonometric functions of one variable are often looked up in tables rather than computed by numerical methods. Particularly when combined with interpolation techniques, table look-up is a powerful tool.

For functions of four or more variables, however, conventional table look-up becomes impractical. If each variable can take on R distinguishable values the number of table entries required for N variables is R^N. Thus, even if each input variable is limited to as few as 16 values over its range, a table for storing a four dimensional function would require 16^4 or 65,536 entries. Clearly, table look-up is impractical for most functions involving more than four variables.

CMAC, however, does not require a unique table entry for each possible input vector. Instead, CMAC maps each input vector into a multiplicity of table entries. This is illustrated in Figure 1. The value of the CMAC output is equal to the arithmetic sum of the contents of the selected memory locations. Now the number of ways a set of "a" elements can be selected from a table with "b" entries is the number of combinations of "b" things taken "a" at a time. This, in practical cases, is much much larger than "b".

Thus CMAC can utilize a relatively few memory locations to represent a space defined by N input variables.

Of course, the fact that each possible CMAC input vector selects a unique set of memory locations rather than a unique single location implies that any particular location may be selected by more than one input vector. CMAC turns this necessity into a convenience by an algorithm which operates such that any two input vectors which are similar (i.e.. close together in input space) map into highly overlapping sets of memory locations as shown in Figure 2.

This gives CMAC the property of generalization, i.e., CMAC tends to produce similar outputs for similar inputs. In Figure 2, input vector $\underline{S}_2$ selects three out of four of the same memory locations as $\underline{S}_1$. Thus, the output $h(\underline{S}_2)$ will be similar to $h(\underline{S}_1)$ differing only by the contents of the single location which is not in common.

The amount of overlap between sets of selected memory locations is controlled such that as the input space distance between two input vectors increases the amount of overlap decreases. Finally, at some distance the overlap becomes zero and the sets of selected memory locations are disjoint. At that point input $\underline{S}_2$ can be said to be outside the neighborhood of generalization of $\underline{S}_1$. The value of the output $h(\underline{S}_2)$ is thus independent of $h(\underline{S}_1)$.

An example of a neighborhood of generalization of CMAC can be seen in Figure 3. In this example, the input vector consists of two input variables s_1 and s_2 with range $0 < s_1 \leqslant 360$ and $0 < s_2 \leqslant 180$. There is unity resolution along each variable axis so that R is 360 for s_1 and 180 for s_2. In Figure 3, 32 weights are selected for each input vector, and thus any two input vectors which differ by only one resolution elements will have 31 weights in common. Not until two inputs are at least 32 resolution elements apart do they map into disjoint sets of weights. If the weights are initially all zero, the value 1 can be stored at some point such as S_1 = (90,90) by placing 1/32 in each of the 32 weights selected by S_1. Following this operation one will find that a second input vector S_2=(91,90) will produce the output 31/32. This is because $\underline{S}_2$ shares 31 weights with the vector $\underline{S}_1$. A third vector $\underline{S}_3$=(92,90) or

$(S_4 = (90.92))$ will have an output 30/32 because of sharing 30 weights with $\underline{S}_1$, etc. The result is that the CMAC memory generalizes. Adjacent memory locations are not independent. and a plot of values stored at each point in input space shows the appearance of a stretched rubber sheet. Pulling one point to a particular value. as in Figure 3. affects adjacent points.

Generalization has the distinct advantage that training (or data storage) is not required at every point in the input space. Training on a suitably representative subset of input vectors is adequate. For example. if one wishes to store the function $\sin(s_1)\ \sin(s_2)$ to a resolution of one degree in a conventiona memory, data would have to be stored at each of the 360X180 (more than 64.000) possible inputs. In Figure 4. data was entered at only 175 input points scattered over the input space and the tendency of the memory to generalize fills in the gaps.

Of course. generalization is disadvantageous if radically different outputs are required for highly similar inputs. In most control problems however. such discontinuities do not obtain. Indeed most control functions have simple S-shaped characteristics along each variable axis. The complexity of control computation in multivarient systems typically derives from cross products which effect the slope of the function, or produce skewness. or non-symetrical hills and valleys in various corners of the N dimensional space. As can be seen from the function shown in Figure 6 these are the type of functions which CMAC can readily store and hence compute.

A Neurophysiological Model

A neurophysiological theory upon which CMAC is based was developed independently in England by David Marr and in America by the author. It was published first by Marr in 1969. The cerebellum had for some years been known as one of the areas in the brain responsible for controlling the limbs. hands. eyes and fingers in rapid,

precise. coordinated movements. It was known to receive neural input from motor areas of the cerebral cortex as well as feedback signals from the muscles. joints. tendons and skin. Marr and myself independently combined this knowledge with new information from an elegant series of experiments by Eccles. Ito and Szentagothai[2] into a theory of how these input signals act through the various cellular interconnections to produce outputs with appropriate values. The result was a theory of how conditioned reflexes could be stored and recalled (i.e., computed). This theory has rapidly become one of the most widely accepted working hypotheses among cerebellar neurophysiologists.

The CMAC formalism not only gives mathematical structure to the original Marr-Albus theory but suggests analytic procedures by which electronic circuits with similar properties can be synthesized.

For example. CMAC first transforms the single precise value of each input variable into a multiplicity of less precise values on a set of intermediate variables. This is analogous to the function accomplished by sensory end organs in biological systems. In the body, the angular position of a joint, the tension in a tendon. the velocity of contraction of a muscle, are all precise physical parameters each of which is encoded by a multiplicity of sensory organs into firing rates on neuron axons which are relatively imprecise information channels. CMAC emulates this in the $S \rightarrow M$ mapping by which the value of each variable in the input vector $\underline{S} = (s_1, s_2, s_3, \ldots s_N)$ is transformed into a set of intermediate variables.

Figure 5 illustrates the essential characteristics of the $\underline{S} \rightarrow M$ mapping. In this illustration the two input variables s_1 and s_2 are represented with unity resolution on the range 0 to 16. The range of each input variable is also covered by four intermediate variables of lower resolution.

In Figure 5, s_1 is mapped into a set m_1 composed of four intermediate variables.

$$m_1 = \{C_1, C_2, C_3, C_4\}$$

where

$$C_1 = \{A, B, C, D, E\}$$
$$C_2 = \{F, G, H, J, K\}$$
$$C_3 = \{M, N, P, Q, R\}$$
$$C_4 = \{S, T, V, W, X\}$$

For every value of s_1. there exists a unique set of elements m_1* one from each set of intermediate variables in m_1. such that the value of s_1 uniquely defines the set m_1*. and vice versa. For example. in Figure 5 the value $s_1 = 7$ maps into the set $m_1* = \{B, H, P, V\}$ and vice versa. Similarily. the value $s_2 = 10$ maps into the set $m_2* = \{c, j, q, v\}$. and vice versa.

In the cerebellum, incoming sensory neurons enter the granular layer where they make contact with granule cells. A system of negative feedback regulates the overall activity of the granular layer so that a small and relatively fixed percentage of the granule cells are allowed to become active. This is stimulated in CMAC by the combination of intermediate variables to select a set of weights. For the example in Figure 5, the set $m_1* = \{B, H, P, V\}$ and $m_2* = \{c, j, q, v\}$ combine to select the set of weights $\{Bc, Hj, Pq, Vv\}$.

Finally, in the cerebellum the Purkinje cell sums the influence of the active granule cells through a set of weighted synaptic connections. Similarly in Figure 5 CMAC sums the selected weights.

Bc	= 1.0
Hj	= 2.0
Pq	= 1.0
Vv	= 0
Sum	= 4.0

Thus, the input $S = (7, 10)$ produces the output $h(\overline{S}) = 4$. The particular set of weights shown in Figure 5 defines the function in Figure 6.

At every point in input space, four weights are selected whose sum is the value of the output. As the input vector moves from one point in input space to an adjacent point, one weight drops out to be replaced by another. The difference in value of the new weight minus the old is the difference in the value of the output at the two adjacent points. Thus, the difference in adjacent weights is the partial derivative (really the partial difference) of the function at that point. For example, in Figure 5, if the input vector moves from

S=(7. 10) to S (8. 10) the weight Bc 1.0 drops out and is replaced by Cc=2.0. The value of the output thus changes from 4 to 5.

CMAC provides a mathematical formalism which is simple. yet precise. and which accurately reflects the functional properties of a cerebellar output cell and its associated interneuron network. Furthermore. the CMAC formalism is sufficiently general that it can be said to closely approximate the functional properties of output neurons and their associated interneuron nets in a large number of cortical regions and subcortical nuclei. A multiplicity of CMAC's can functionally simulate large sections of cortex and entire processing nuclei. CMAC thus provides a mathematical tool which may be applied to analysis of sensory-motor systmes in many different areas of the brain.

Input to CMAC is a vector which defines an N dimensional space. At any instant of time the input vector defines a point in input space. If any one of the input variables (component of the vector) is time dependent then as time progresses the input vector moves through space defining a trajectory. We can now speak of an input trajectory through N dimensional input space. At any instant of time CMAC accepts an input vector and produces an output which is a scalar. A multiplicity of CMAC's produces an output vector.

Thus. for a cell nucleus represented by a multiplicity of CMAC's. at any instant of time there exists a one-to-one mapping from each input point (vector) in input space into an output point (vector) in output space. And correspondingly for every input trajectory there is an output trajectory.

Let us now explore how these basic CMAC concepts can be applied to the subject of higher level behavior such as is required in the performance of a behavioral sequence. or task.

A Sensory-Motor Hierarchy

Simply stated. any task can be broken down into subtasks in a hierarchical way until finally at the bottom of the hierarchy there are action primitives such as motor neuron firing rates. This hierarchical decomposition of complex tasks. or behavior. is not new having been suggested many times by workers in behavioral psychology. linguistics and various other fields such as military command and control, and manufacturing engineering. The entire concept of mass production is based on the principle of breaking down the task of making a product into a series of elemental operations which are so simple that they can be taught to low-skilled workers in a short period of time.

Task decomposition immediately suggests a control hierarchy where each level in the hierarchy accepts input commands (or tasks) from the next higher level and responds by issuing ordered sequences of output commands (or subtasks) to the next lower level. In extremely simple cases these subtask generators may merely produce sequences of pre-recorded outputs. But in more complex systems sensory feedback from the environment. or from the system being controlled. may alter the output sequences in one way or another. Sensory feedback may consist of interlock signals to provide sequential timing. or may incorporate any number of analog or digital variables which modify the transfer function of the subtask generator.

Suppose. for example. that the goal is to program an industrial robot to assemble a gasoline engine. This task can be broken down into sequences of simpler tasks such as fetch parts a and b. insert a into b while nulling off-axis forces. fasten part c to assembly ab and so on. Each of these tasks can be broken down further into sequences of elemental movements such as reach. grasp. follow specified trajectory, etc. These elemental movements can themselves be broken down into sequences of positions in physical space. Finally. each point along the physical trajectory can be transformed into a coordinae system defined by the physical structure of the robot and its actuators. This concept is illustrated in Figure 7.

At each level in such a hierarchy there are two types of input. First there are input commands from a higher level. At the very top these may come from the shop foreman or from a production planning and scheduling program. At all other levels. commands originate as outputs from higher levels in the hierarchy.

Each hierarchical level also receives feedback signals which report the position and motion of joints or convey information from sensors monitoring force, touch, or visual position of objects in the environment. In some cases this feedback is used for timing purposes in order to coordinate sequences of actions with conditions in the environment. Feedback may also indicate the recognition of patterns of events, or shapes and locations of objects in the environment, or even the movement of coordinate systems and frames of references. Particularly at the higher levels in the hierarchy, feedback is highly processed through many layers of an ascending hierarchy of CMAC like processing nuclei.

In biological systems as well there may be a sensory-data processing hierarchy, which runs parallel to and in the opposite direction from the motor-vehavior generating hierarchy. This processing hierarchy receives input at the lowest level directly from sensory transducers. At each level input vectors (and trajectories) are transformed into output vectors (and trajectories). These outputs are then passed on to the next higher level as inputs. The purpose of each processing module is to transform input vectors into output vectors which are optimally configured to serve as feedback to the motor-generating hierarchy at that level. We can therefore hypothesize a feedback link from each level in the sensory-processing hierarchy to corresponding levels in the motor-generating hierarchy.

Furthermore the efficiency of the sensory-processing hierarchy is enormously enhanced if there are complementary links from the motor-generating hierarchy to the sensory-processing hierarchy. This type of information pathway is what neuropsychologists call an "efference copy." This information tells the sensory processing hierarchy what the body is doing so that, among other things, it can distinguish sensory data resulting from movement of the body from sensory signals resulting from movement of objects in the environment of the eyes and a rotation of the room about the eyes. More generally, it enables the processing system to perform context sensitive filtering. and indeed, to do predictive filtering.

It is, of course, possible to turn off the lowest level of the motor hierarchy without disabling the entire processing-generating hierarchy. When in this mode, the generating hierarchy

can be used to produce signals which facilitate the operation of the sensory-processing hierarchy. Activity in the generating hierarchy may now be better characterized as hypotheses. instead of tasks and subtasks. Sensory input from the environment is now analysed in conjunction with hypotheses from the generating hierarchy. If the hypotheses are correct, they will assist in sensory recognition. If only nearly correct they can be "pulled" by feedback from the processing hierarchy. When a particular hypothesis is successful in generating predictions which match incoming sensory data the entire processing-generating hierarchy "locks on" to the incoming sensory data. This gives the hierarchy the ability to recognize and track lengthy sequences of input signals even in the presence of noise and interference from similar signals. This lock-on phenomenon at several levels gives the hierarchy the ability to recognize phrases and patterns with several different, but harmonious, periodicities. For example, the affinity of the ear for rhythmic patterns of music and poetry may arise from synchrony in hierarchical looping structure such as shown in Figure 8 where each of many different loops locks-on to rhythmic patterns at its own level. The cross-coupling in the sensory-motor hierarchy suggests a mechanism for the strong tendency of people to dance, or tap their feet in time with rhythmic sounds. The multiplicity of levels in the hierarchy suggests a mechanism for simultaneously locking-on to many different levels of rhythm and "meaning" in both speech and music.

Belief and Understanding

One may hypothesize that in humans the functional relationships stored in upper levels of the cross-coupled hierarchy of sensory-processing behavior-generating modules makes up what might best be called a "belief structure" which gives rise to goal-directed behavior. By definition the belief structure is isolated from direct contact with the motor-output sensory-input levels. This enables the belief structure to decouple itself from outward manifestations of behavior as well as from direct physical sensations of the external environment. It is thus free to

generate hypotheses, and imagine the consequences. It can send hypothetical goals to the mid-levels of the generating hierarchy which cycle through an imagined task, thereby stimulating the sensory-processing hierarchy into producing a facsimile of expected (or remembered) sensory experiences. These self-induced sensory feedback signals generate emotional reactions good, bad or neutral and these steer the highest level goal selecting mechanisms toward command vectors which produce rewarding emotional feedback.

Thus, a person selects goals and plans actions on the basis of what is stored in the transfer functions of his or her belief structure, i.e., the mid and upper-levels of the processing-generating hierarchies.

Similarly, a person interprets sensory experiences according to what is stored in the belief structure. People tend to see and hear what they expect to see and hear. They tend to interpret unfamiliar sensory input as "just noise" or without "meaning." Sensory experiences which correlate with patterns stored in the belief structure are reassuring and comforting. The world becomes predictable and presents few surprises, i.e., we "understand" and events "have meaning." Unexpected twists in familiar patterns elicit surprise, and systematic deviations from expected trajectories may cause relearning and the establishment of new trajectories of expectation. Sensory experience which does not correlate with stored patterns is disturbing and is rejected or avoided. If such stimuli cannot be avoided and if it is too unfamiliar or unpredictable to be reinterpreted through learning, then it may produce emotional distress, or neuroses.

A processing-generating hierarchy of CMAC modules thus provides a simple unifying structure which can be applied to a wide range of sensory-interactive goal-directed behavior including the selection of goals, the planning of actions, the imagining of results, the learning of motor skills, recognition of sensory patterns, and the construction of habits and beliefs. The fact that the basic building blocks for such hierarchies can be fabricated inexpensively with presently available technology suggests that it may now be reasonable to seriously address the feasibility of constructing machines with these properties.

Each CMAC module is an independent computing device. It accepts input variables, performs its computations, and transmits its output. Each module is trained, or programmed, to compute a relatively simple function of many variables. Training begins at the lowest levels first and must be well underway at each level in the hierarchy before it can begin at the next higher level. Just as in a child the abilities to deal with abstractions do not develop until after visual processing and motor coordination is accomplished, and the ability to read does not emerge until speech generating and recognizing have been developed, so in a CMAC hierarchy training must be well advanced at each level before consistent and clearly defined trajectories emerge to be used as feedback for the next higher level.

Networks of CMAC modules compute in parallel and control is distributed throughout the network in a manner very similar to the neural networks of the brain. The addition of new computing elements at each level serves to refine the precision of the computed output at that level. The addition of new levels in the hierarchy increases the complexity of the behavior which can be generated and the sophistication of the sensory processing which can be performed.

These properties suggest that it may be possible to implement a CMAC hierarchy for controlling a robot on a network of microcomputers, and to systematically increase the sensory-motor capacity by incrementally adding additional modules to the network. If the model suggested here is correct, the result may be that intelligence will evolve in a silicon and copper network in much the same way as it did in biological brains.

References

Marr, D., "A Theory of Cerebellar Cortex," J. Physical, London, <u>202</u>, 1969, 437-470

Albus, J., "A Theory of Cerebellar Function," Math. Biosciences, <u>10</u>, 1971, 25-61

_____. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," Journal of Dynamic Systems, Measurement and Control, September 1975, 220-227

_____. "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," Journal of Dynamic Systems, Measurement and Control, September 1975, 228-233

.Eccles, J.C., M. Ito, and J. Szentagothai, "The Cerebellum as a Neuronal Machine," Springer, Berlin, 1967

It is impossible in such a brief paper to properly acknowledge the source of all the ideas presented. A much more complete bibliography is contained in each of the above sources.
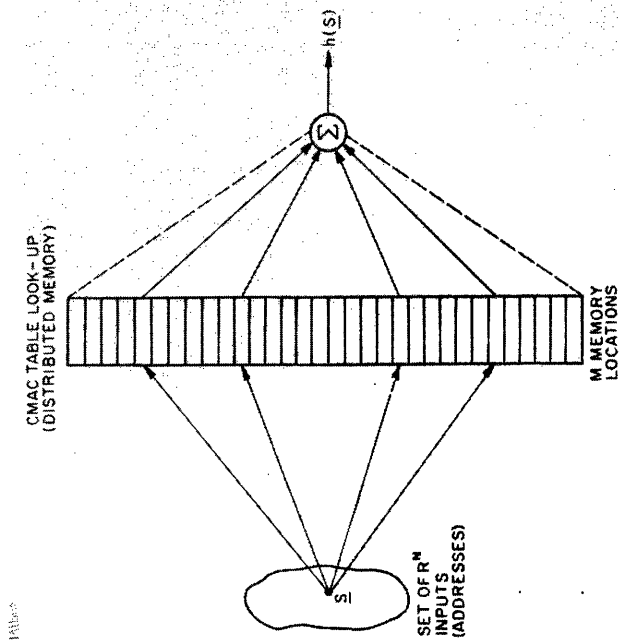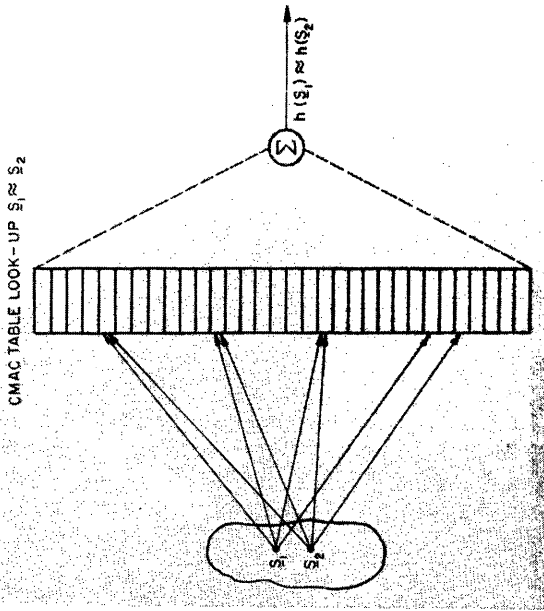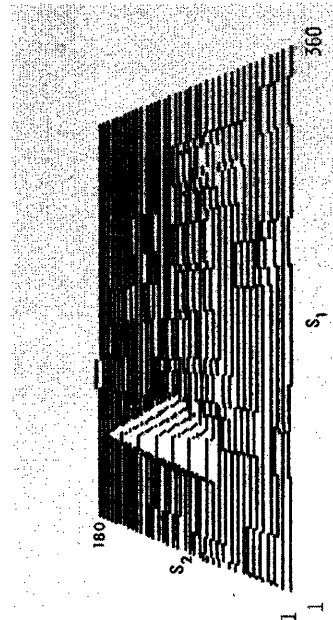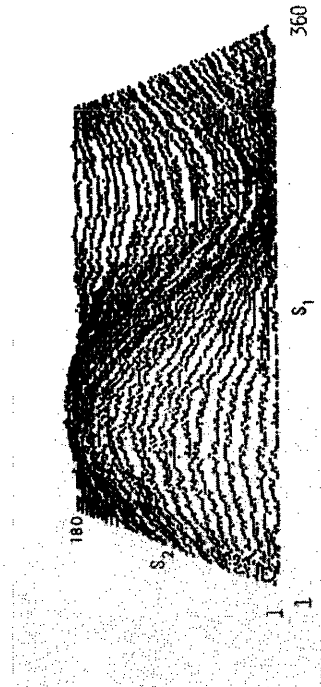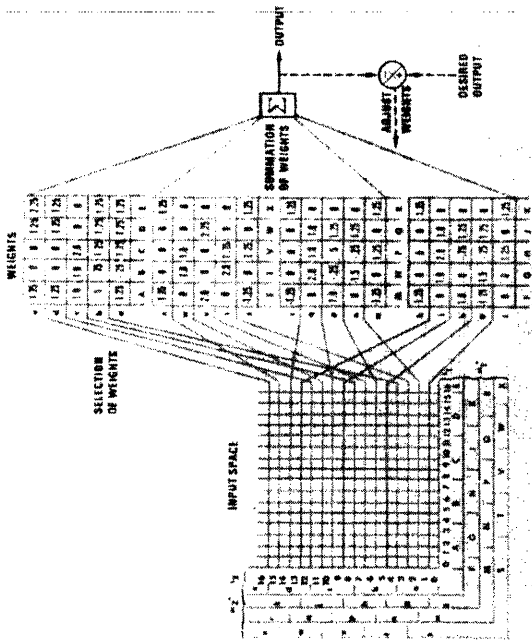
FIGURE 1

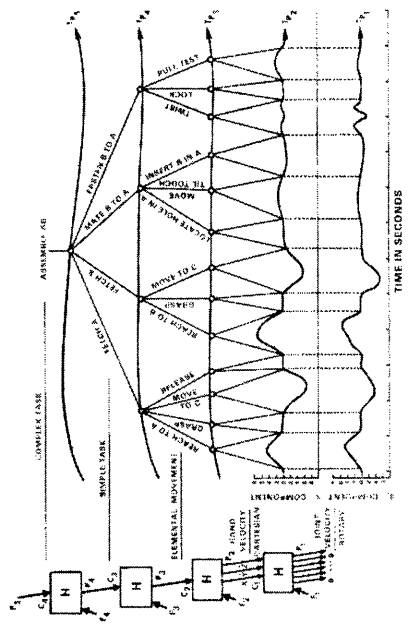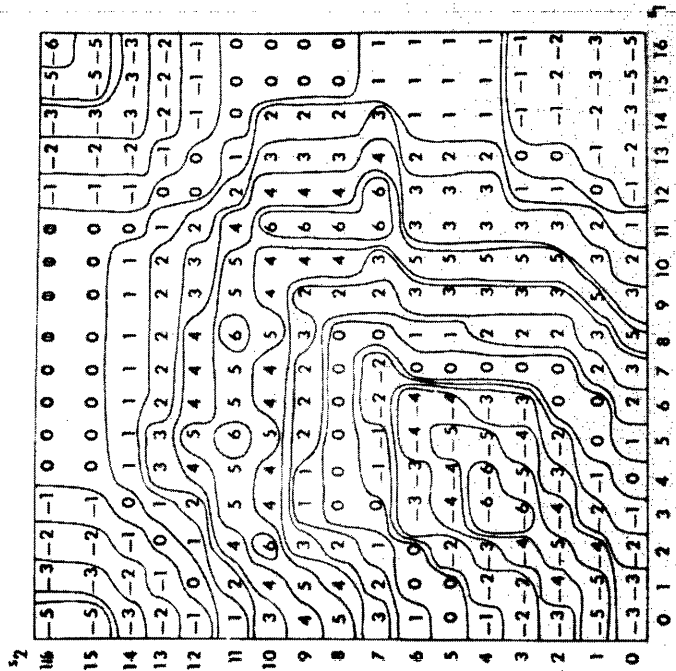FIGURE 2

FIGURE 3

FIGURE 4

FIGURE 5

FIGURE 7

FIGURE 6

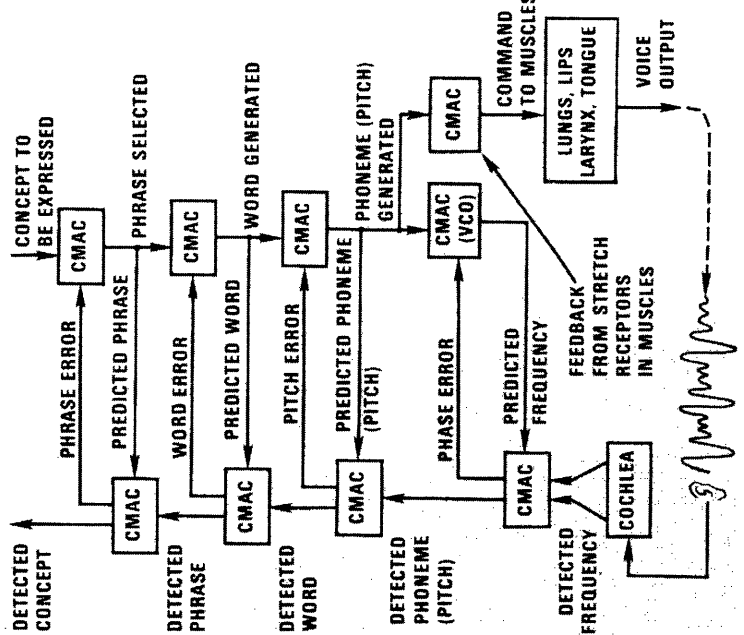FIGURE 8

A PEEK BEHIND THE PCNET DESIGN

by

Mike Wilber

320 Dennis Drive, Palo Alto, CA 94303

Abstract

Various aspects of the PCNET design are discussed. The emphasis here is not so much to expose the design as to show the motivation behind various of its aspects. The principal theme is shown to be that the PCNET embodies an attempt to maximize the accessibility of useful communication techniques to the broadest possible segment of the computer hobbyist community. On the other hand, people having enhanced power in their hardware or software can take good advantage of their advanced facilities when they use the use PCNET techniques for their communication.

1. Introduction

Dave Caulkins organized the PCNET Committee shortly after the First West Coast Computer Faire last year; its goal was to build a personal computer network (whence its name). After a few months of lively debate, it produced a design, which was subsequently implemented for test purposes. The design was then disassembled and reconstructed in such a way as to profit from the experience of the first iteration. The second generation design is now quite stable and is beginning to generate some second generation test results. At this juncture, then, it could perhaps be instructive to consider various aspects of the design and see just why they turned out the way they did.

2. The Ground Rules

The design of the PCNET proceeded from a number of ground rules.

0. The network should be cheap and reliable. Then people would be likely to freely share large files and large numbers of substantial programs. If they have confidence in network communication, they will be tempted to use it casually to spread the word of a good program, an interesting new idea or a reliable merchant.

1. The network should own no hardware, such as leased lines or service computers. Then, the network can come into operation without any necessity to raise or administer substantial funds, and it can begin operation with too few participants to justify a large expenditure. Also, with the network administration highly decentralized, it will likely be responsive to the needs of its participants.

2. Permanent service nodes will not be needed but will be accommodated if present. All the above arguments for decentralization also apply to this ground rule. However, permanent service nodes, such as the CIE's I've described elsewhere, can eliminate very strong incentives for specialized hardware that will enable nodes to automatically handle incoming traffic.

3. There should be no geographical restrictions. For example, the PCNET should be suitable for communication across national boundaries.

5. The PCNET should accommodate remote virtual terminals. This will allow people to test a program without having gone to the trouble of importing it and adapting it to a new home. This feature, while not yet explicitly included, has shaped some parts of the design.

3. Levels of Abstraction

The PCNET specification uses a number of levels of abstraction to describe the transmission of mail and files between network nodes. Two network nodes will typically be connected by a phone line. They use modems to send digital bits in the form of the audio tones suitable for telephone circuits. These bits serve in their turn to serially transmit binary data in 8-bit bytes. (Nodes using 5-bit serializers or that for other reasons cannot directly transmit 8-bit bytes use a radix-41 scheme to represent binary data with a restricted set of 41 characters.) These bytes are organized into transmission blocks containing error correction information sufficient to provide transmission (at very low error rates) for several streams of bytes. One stream is devoted to information used to control the communication at this level and the previous ones; the other streams transmit messages between the server levels of the nodes. Finally, these messages transmit mail and files between two nodes, with forwarding automatically provided when appropriate.

These levels of specification are rigorously separated, so that any of them may be changed with little or no disturbance to the others. An example of this modularity has already been seen in the way the choice between radix-41 and straight binary has no effect whatever on transmission at higher or lower levels of abstraction. Similarly, the interface between the modem and the phone line may or may not include provision to automatically answer (or even originate) phone calls with no effect on the other levels. By the same token, the modem (and UART) can be half duplex or full duplex, and the only level effected is that handling link transmission blocks.

This separation between levels will allow the PCNET to gracefully take advantage of new developments. For example, the telephone links can be replaced by radio links or even by links having the delays that are inevitable in the usual satellite communication. Character-at-a-time interactions can also be added, so that one node can serve as a virtual terminal to programs in another node.

4. Some other Design Issues

The format of a link transmission block is independent of the choice between binary and radix-41 in the sense that the radix-41 transformation is applied to a complete block before it is put onto the line. This does not seem to greatly reduce the effectiveness of the error detection information, and it leads to a simpler implementation than would be required otherwise.

Noise on the phone line can cause the receiving UART to lose synchronization with the incoming byte frames, completely turning the received data into garbage until synchronization is again established. We assume that the error detection information in a transmission block will have no trouble detecting any such garbage. In an attempt to limit the damage to single transmission blocks, they are separated by characters chosen for their utility in helping the receiving UART come back into synchronization with the byte frames.

We had originally assumed that the boundaries of the link transmission blocks and the stream blocks would be entirely independent; we soon discovered, however, that such independence could require a general coroutine capability in the implementation. In order to maximize the accessibility of the PCNET to the hobbyist community, we made it possible to negotiate the independence of the boundaries at those two levels. In addition, the default choice of that option is to align the boundaries so that coroutines are never required.

The tension between symbolic data and more highly coded data can scarcely escape notice in a system like PCNET, with its largely symbolic message header and more highly coded stream and link transmission block headers. The lower levels handle blocks much smaller than messages, so header compactness is correspondingly more important at the lower levels. Programs implementing the message server level, on the other hand, are more likely to have their functions expanded and so are more in need of data whose meanings are easily discerned by the people maintaining them.

Bizarre though it may seem, PCNET messages probably will normally be sent from their source directly to their destination or to a mail drop near the destination. That is, they will probably only rarely be sent through a number of intermediate nodes on their way to their destination. (The reason is a simple economic one: the late night phone rates are low enough and independent enough of distance that relaying doesn't seem worth the bother.) However, the PCNET specification is organized in such a way that either mode can be used, that any mixture of the nodes can be in use at any moment, and that the choice can be made independently by the sender each time it attempts to send a message, in case the first few attempts fail. The crux of the matter is that the connection from the original source of a message to its ultimate destination can be a simplex (i.e., one-way) connection, even though the connections at lower levels must be at least half duplex. On the other hand, there is nothing to prevent the original sender and the ultimate destination from having a full duplex connection; in fact they can profit from having such a connection if they have one.

The PCNET is largely immune to nodes capriciously entering and leaving network participation in another way too: successful transmission of messages can be verified from the ultimate destination all the way back to the original source. (This is at the option of the original sender.) Thus, a series of intermediate nodes can participate in the transmission of a message without their needing to accept any responsibility for ultimately successful transmission between the end points. Since intermediate nodes have only nominal responsibility for the traffic they handle, the PCNET should gain both a streamlined administration and considerable resistance to any possible spontaneous disappearance of nodes.

5. Conclusions

The most important consideration throughout the PCNET design is to maximize its accessibility to as broad as possible a segment of the hobbyist community. A consequence of this is that the design has special features that are included just so the design can be implemented simply and cheaply. It is also structured in such a way that simple implementations will tend to be reliable. On the other hand, when more highly specialized or costly hardware is available or the software can be more sophisticated than the minimum requirements, the PCNET design is structured in such a way as to put the advanced facilities to good use and deliver increased performance to the user. The design also leaves room for the addition of new features, not all of which were foreseen by the PCNET Committee.

6. Acknowledgments

Some of the opinions expressed in this paper are my own. However, the vast majority of the opinions, conclusions and observations incorporated here were developed wholly or jointly by the members of the PCNET Committee. Hoping to find a reasonable middle ground between the folly of evaluating the work of others and the arrogance of leaving their contributions unacknowledged, I feel compelled to say that I found most memorable the contributions of Dave Caulkins, Ron Crane, Peter Deutsch, Marc Kaufman and Robert Maas.

7. Bibliographical References

[1]. David Caulkins, "Design Considerations for a Hobbyist Computer Network", Proceedings of the First West Coast Computer Faire, Palo Alto, 1977.

[2]. Mike Wilber, "A Network of Community Information Exchanges: Issues and Problems", Proceedings of the First West Coast Computer Faire, Palo Alto, 1977.

Communication Protocols for a Personal Computer Network

Ron Crane
2101 Calif. St. #326   Mountain View, Ca. 94040

This paper summarizes the design requirements and architecture of a layered set of communication protocols for personal computers. Also included is the current state of development of this evolving set of protocols. A detailed description of the protocol will appear in the future.

The Environment of Computers

The introduction of low-cost minicomputers in the 1960's reduced the minimum size of an application for which automation was justified. The single chip microcomputer is making an equally significant step in the 1970's. Private individuals are purchasing small machines or terminals for their own use, in addition to businesses for applications such as accounting, order handling, and text editing. Small, medium, and large size computers are becoming widespread.

More and more machines are being dedicated to uses which involve transactions of some kind (editing text, mail, or programs ultimately sent to other individuals, doing accounting for transactions between buyers and sellers, etc.). The machines are used to create, modify, or process information which is then printed on paper or recorded on magnetic media which is then carried to its destination where it is often entered into another machine. This represents a growing environment in which machines are communicating with other machines, but with the transport time from one machine to another comprising a significant part of the total elapsed time for processing. The need exists for low-cost, quick, and direct machine-to-machine communication between an increasingly widespread base of machines.

The requirement of providing communication between any of a large number of machines implies some form of large and widespread communication network. To be a viable replacement or adjunct to physical transportation schemes presently used, the network must be available and relatively inexpensive. The only system currently available that is both widespread and has a very low minimum monthly charge is the dial telephone network.

For machines to communicate using the telephone network, they must be able to communicate both *with* the telephone system and *through* the telephone system to the distant machine. Once two machines have been connected via the telephone system, they must communicate with each other. This communication must take place on several levels. Modems communicate using a standard set of frequencies, the serial interfaces in each of the computers must transmit and receive bits in the same order and at the same speed, and the software in each of the machines must agree upon the meaning of various bit sequences so that *information* is transferred instead of a meaningless sequence of bits. A common language or communication standard is essential for widespread machine communication.

Personal Computer Network (PC Net) Protocols

The PC Net protocols come from the PC Net committee which is a group of computer professionals who banded together as the result of the personal computing network session at the First West Coast Computer Faire in 1977.

The PC Net protocols are layered into 5 functional levels. Two of the levels specify hardware while the remaining three specify software. Every machine must implement the basic or core protocol. Extra capabilities may also be implemented on various machines. The compatibility of two machines with respect to extra capabilities is ascertained on a per call basis via the core protocol, as is the decision to use one or more of the capabilities during the call. Thus, any two machines can always communicate with each other on a very simple level, even though some will find that they have few common interests after a brief interchange.

The figure on the following page illustrates the hierarchical structure of the proposed standards. Five levels comprise the standard (protocol).

Level 1

Level 1 specifies how the node (personal computer) interfaces to the telephone network. Both supervisory signals (dialing) and the modem signalling frequencies are specified. Bell 103 compatible modem frequencies were selected because of wide availability and relatively low cost. When faster units become desirable, this level of the protocols can be changed without affecting the other levels. In the figure, RS232C and RS366 are mentioned as often-used interfaces to modems and auto-dialers. These are not part of the standard, however, since neither interface can be seen by another node through the telephone network.

Level 2

The data transmission format specified by this level is asynchronous start-stop codes at 300 bits per second, again readily available in existing equipment. The interface to and control of the

Computer

5			
5	4	3	**Interface** 2
5			

Modem & Auto Dialer 1

Dial Telephone Network

1 **Modem & Auto Dialer**

Interface 2

Computer

			5
3	4		5
			5

RS232C *
RS366

Bell 103 Modem Standard &
Automatic Dialer

RS232C *
RS366

Level 1

Asynchronous Start–Stop Codes @ 300 BPS & Auto Dialer

Hardware **Level 2**

Software

Link Transmission Blocks (sequenced and error free byte delivery) and
Telephone line control

Level 3

Process to Process Stream Blocks (multiplexing of data link between user programs)

Level 4

User Program to User Program Standards   (Node Control, Mail, File Transfers, etc.)

Level 5

* *RS232C and RS366 specify typical modem and auto-dialer interfaces, respectively, but are not part of the PC Net
standard since the auto-dialer may be integral to the modem which may in turn be part of the computer interface.*

Personal Computing Network Protocol Hierarchy

automatic dial and answer telephone interface is not specified except that the function must be performed. This includes the possibility of manual operation. Timeouts will be set to allow manual operation.

Level 3

This level provides sequenced and error free delivery of bytes from one machine to another, as well as control of telephone connection setup and takedown. Blocks of bytes are transmitted on the link with a header and checksum in a fashion similar to synchronous protocols like ADCCP (Advanced Data Communication Control Procedure). A synchronous bit oriented protocol was not used because of the unavailability of hardware interfaces to implement it. Error control is implemented using a checksum and byte count instead of a CRC because the CRC is cumbersome in software and most asynchronous interfaces do not support it in hardware. Sequence numbers provide acknowledgement of correctly received blocks and in addition permit several blocks to be sent without acknowledgement, without getting out of order.

Two methods of transmission on the line are specified. Radix-41 is the default start up mode and full 8-bit codes are an option.

Radix-41 is a 2 byte to 3 character packing scheme proposed by Mike Wilber to avoid the problem of operating system intervention (interrupting on control characters and uppercase conversion of lower case characters) when PC Net protocols are implemented in BASIC or other high level language. The result of using this packing scheme is a reduction of 33% in throughput.

Transmission blocks are transmitted with only minor modification when using 8-bit codes. Special flag characters are used to separate transmission blocks on the link. A transparency rule is then used to permit transmission of this flag character if it occurs within the transmission block.

Level 4

This level is useful for nodes in which several user programs are running concurrently and are also all using the phone link. Level 4 performs multiplexing and flow control of data streams from each of the programs into the single phone link. Separate 8 bit fields specify source and destination process addresses for each process-to-process stream. Fields also exist for sequence and stream numbers. In small nodes this level can be merged with level 3. The fields specified for the level 4 block format still remain, but can be ignored by the simple nodes. Small nodes can still communicate with large nodes capable of

multiplexing, but only one process to process stream can exist at a time in this case.

Level 5

User programs exist at this level. They manage the communication tasks such as sending or forwarding mail or files, and interacting with people on whose behalf the communication is taking place. There is currently a mail and file transfer protocol written by Peter Deutsch and a global addressing scheme using latitude, longitude, and phone numbers authored by Doug Bourne.

Summary

The PC Net protocols are being designed to be easy to implement in currently available systems, but with room for growth and modification if there is demand for it in the future. The first two levels are essentially complete at this time. The top three levels have been specified and are undergoing revisions relating to interfaces between these levels. Detailed descriptions of each level of the protocol will appear in the future.

This document is one more attempt to explain the PCNet protocols to persons planning to write their own software. This document is written bottom-up so that node volunteers don't have to read it backwards to know what to implement in sequence. It also contains no completely worked-out examples (see [Maas WR2] for them, as well as a top-down approach), and no test data for debugging node software (see [Maas EXPERI]). It also doesn't fully specify timeouts and other obscure features of our protocols (see [Crane]; point of nomenclature - references of the form AAAA.BBB are to PCNet documents maintained as on-line files).). The two earlier primers, PROTO.PRIMER and PROTO.PR2 may or may not be useful to supplment this tutorial.

QUICK LISTING OF DATA PASSING BETWEEN ADJACENT LAYERS OF PROTOCOL:

This section will be mostly meaningless until later sections have been read, but due to our present indecision as to what certain things should be called there are some synonyms that ought to be pointed out before proceeding.

Phone-line: modulated carrier, Bell 103 standard.

Modem cable (optional): two-level bit-serial with start and stop bits, RS-232, not present if UART+MODEM on one board.

I/O or memory-mapped hardware interface: 8-bit bytes containing Radix-41 characters and framing, making up LTBs (Link Transmission Blocks).

Link-level midpoint: binary translation of LTBs, called "TBs" (Transmission Blocks).

Pure-binary stream: data portion of TBs, containing PPSBs (Process-to-Process Stream Blocks) packed end to end.

PPSs (dynamically created and destroyed): data portion of PPSBs that comprise that particular PPS.

Disk I/O: implementation dependent and optional.

(Note, the term "block" used by itself in this tutorial usually refers to a TB or LTB.)

QUICK LOOK AT ALL THE LAYERS:

Hardware (not detailed below):

For compatibility with existing software-service bureaus, Arpanet-host dialup equipment, manually operated terminals such as my Beehive used to remotely test software, and low-cost available personal computer serial-i/o interfaces -- our network will initially be geared to the Bell-103 standard rather than any of the brand-new bit-synchronous communications standards such as HDLC, and will run at 300 baud (110 optional) rather than 1200 baud.

The interface to the main body of software will consist of a GETCHARACTER routine that returns -1 if no character has arrived at the UART (Universal Asynchronous Receiver/Transmitter) within a reasonable time and a number between 0 and octal 377 (the ASCII, or rarely EBCDIC, value of the characer) otherwise, and a PUTCHARACTER routine that either waits until the character can be stuffed into the serializer or returns an error flag if it couldn't be stuffed immediately (wait-and-stuff would be used in halfduplex protocol, stuff-or-error would be used in fullduplex protocol). An alternative is fully-buffered i/o that is driven by interrupts, but with similar interface characteristics. Timeouts are not a fulltime-essential feature, rather are used to avoid telephone calls longer than necessary to either transmit a message or determine that the other node is sick, and to detect that a rare failure of halfduplex turnaround has occurred so that the deadlock can be resolved. It is expected that during 99% of connections not a single half-duplex deadlock will occur, thus with semi-manual operation all timeouts are optional.

No further discussion of the hardware and its interface will be done in the rest of this tutorial.

An important segment of the PCNet protocol relates to Telephone Call Management(TCM). One of the PCNet design goals is graceful sharing of a single telephone line between voice and data use. TCM covers the activities which must take place from the time the PCNet node phone line goes off-hook until the time the Frequency Shift Key (FSK) handshake between the two communicating PCNet modems is complete. TCM also deals with call termination - the activities from the end of data transmission until the phone line goes on-hook.

PCNet TCM is intended for use in three different modes; 1) Attended (manual control). This mode assumes people present who will answer all calls, switching PCNet calls to the computer when they occur. 2) Attended (computer control). This mode assumes people are present for voice calls, but that the computer answers all calls and signals the people to accept voice calls. 3) Unattended. This mode assumes that people are either absent or asleep; the computer answers all calls, minimizing audible ring signals inside the called premises.

TCM assumes that a sophisticated modem is available; one capable of detecting telephone signals such as ring, busy, dial tone, etc. Simpler modems with time-outs may be used with the penalty of slower and less efficient operation.

In present attended (computer control) and unattended TCM modes the computer goes off-hook for all incoming calls; if the call is voice and not data both people on the premises and the calling party must be given a ringing signal so that the voice connection may be completed. This is undesirable in that billing will start as soon as the phone line goes off-hook and before the voice call really begins. Also a special piece of equipment is required to generate the ringing signals. Some better way of differentiating between voice and data calls on the same phone line would be desirable.

Link level (bottom half of communication software):

Errors in transmission invariably occur when using modems over dialup lines. Thus a facility for detecting errors, requesting retransmission (implicitly in PCNet by a lack of affirmative response within a reasonable time), holding out-of-sequence blocks until the missing earlier blocks can be retransmitted, putting blocks into their correct sequence, and delivering verified and sequenced data up to the next level; has been included. In the default mode of operation, only one block can be sent at a time, eliminating the need for buffering and sequencing.

When implementing our protocols as user programs on existing computer systems, limitations in the characterset available as input to a node, and sometimes even as output, usually make it impossible to transmit arbitrary 8-bit bytes across the line. For example, most systems ignore the parity bit on input, and set the parity on output regardless of what the programmer actually supplied as the octal 200 bit of outgoing data. Furthermore many systems supply linefeed to the input stream after carriage return that comes in, ignore null, do strange things like killing or holding output when control-O or control-B is input, and interrupt the program completely when receiving control-C or control-Z. Some systems even convert all lower-case characters (octal 141 to 172) into upper-case! To avoid almost any possible conflict when passing binary data, a subset of 41 (decimal) characters called Radix-41 has been selected to be transmitted across the line. Two 8-bit bytes of binary data are represented by three bytes of Radix-41. A more complete discussion of the alternatives, and reasons for our belief that Radix-41 is the optimal method for our purposes, are in [Maas RAD41].

Many systems echo back anything typed at their input lines, and on most systems this echoing cannot be turned off completely. Also, most programmers using BASIC on personal computers are unable to handle multiple concurrent processes, nor fully-buffered i/o, thus a fullduplex mode of operations where data is simultaneously travelling in both directions at the same time, is infeasible. We have thus chosen a method of simulating half-duplex mode of operation on any full-duplex or echoplex or true-halfduplex line, and it is the default mode of operation. To almost eliminate deadlocks caused by the turnaround character being lost due to line noise, 3-out-of-5 majority logic is used to determine whether the other node has or hasn't finished transmitting. To avoid being confused by seeing your own echo, a different turnaround character is used for the two nodes in a link ("[" vs. "]"). Also, each TB contains a bit telling who sent it, so that echoed TBs won't be accepted by the node that sent them in the first place (in the event that echo is delayed due to buffering, turnaround get momentarily confused, or echo occurs while in full-duplex mode).

Node level (third quarter of communication software):

The interface between the Link level and the Node level consists of a "pure binary stream" which is a sequence of 8-bit bytes with all 256 possible values legal. Files can be transmitted using only the hardware and Link levels, however that leaves no method for starting and stopping a transmission. The Node level is a way to multiplex control information (start and stop of transmissions of files, as well as node-identification, network statistics reports, requests for fullduplex mode of operation, and advanced features we haven't even thought of yet) and data (one or more file-transfers, and maybe even additional services such as "TELNET" (link between a terminal and a remote interactive program)) along a single pure-binary stream.

Virtual circuits are established between a process on one node and a process on the other node, for example between a process having a file it wants to transmit and a process that accepts files and saves them on its disk. We call such a

circuit, at this level, a Process-to-Process Stream. Normally here will be one PPS in each direction so that the two processes can talk in duplex. Each PPS is broken into blocks, called PPSBs, so that it is easy to detect end of transfer without knowing the total data count at the very start (there is a bit in the header of each PPSB telling whether it is the last or not, and a count of zero data bytes is legal so that end-of-transmission can be sent at the very last possible moment if it is only then when the sending node realizes it really is done), and so that control messages and/or blocks of some other PPS can be multiplexed between PPSBs of any PPS in progress.

Optional features include keeping track of buffer allocation at this level so that if one PPS gets blocked the link can still be used to transmit other PPSs without losing data of the blocked PPS (i.e. no data from blocked PPS is put into the pure-binary stream until the other node announces that it has more room in its buffer to store it).

In the default mode of operation, only one transfer can be active at a time, no accounting for allocation of buffers is done, and any control messages or requests for additional transfers are ignored while a transfer is in progress. Also, the beginning of each PPSB is located at the beginning of a LTB and each PPSB is wholly contained in that LTB, so that an incoming block can be handled completely by a closed subroutine that preserves almost no state-information from one block to the next.

The result of all this is that pure-binary duplex communications are established between File Transfer Process (FTP)/MAIL server-processes on the two nodes, without preventing upward compatibility into multiple concurrent services occurring between a pair of advanced nodes, and without preventing an advanced node from talking to a very-simplest node. In the simplest implementations it is expected that all levels in the receive pipeline will be compressed into a single piece of code that checks incoming data for validity, discards anything containing transmission errors, discards any PPSB it isn't able to accept, ignores any control mesage it doesn't understand, and depending on a boolean variable either awaits a request for opening a file transfer or delivers data from the active transfer to some output file. The transmit pipeline would need a queueing mechanism so that replies from the FTP/MAIL server-process could be sent as soon as turnaround occurs, and TBs containing no data could be sent if no data is awaiting transmission.

Server-process level (top quarter of communication software):

A server-process is a subroutine or other chunk of program which actually does something useful, as contrasted with all the routines at lower levels that do the grungy stuff necessary to make server-processes possible. Thus the server-process level is the highest level of the protocol. At present we define two server-processes, "control" and "FTP/MAIL". A server-process is specific to one particular type of activity, whereas all the other levels of software are general-purpose for linking nodes and server-processes together. A "listener" is that part of a server-process which sits waiting for the other node to give it a request to do something, as contrasted with performing an action after the request has been received, or spontaneously initiating actions or requests, which are done by non-listener parts of a server-process.

The control-listener handles any PPSB addressed to process 0, checking to see if it knows the control opcode, and if so then acting on it somehow. In advanced nodes, an explicit negative-acknowledgment will be issued for anything it understands but rejects, and for anything it doesn't understand that is in affirmative mode. A simple node can ignore anything it doesn't like.

The control process also has the duty of generating control messages to be sent back. Semi-mandatory is identification of the node. Optional are requests to go fullduplex or binary (non-radix41) or unsynchronized (PPSBs and LTBs overlapped for increased thruput) or allocated (explicit allocation granted before data can be sent on a non-control message, except for initial requests for transfer which are "borrowed" from the later allocations).

The FTP/MAIL listener handles any new PPS that is addressed to process 1. These usually consist of requests for starting a file transfer. If the request is acceptable at this time, a duplex connection (one PPS in the other direction, in addition to the already-started PPS that requested the transfer) is established between a piece of code on one node and one on the other node. When the end of the original PPS occurs, end-of-transfer is signalled, and an acknowledgment and optional file-checksum is sent on the reverse PPS just before it too is closed.

Local forwarding/mailing software outside server-process:

Before a message or file can be transmitted to another node, it must be created somehow, and then queued for the server-process to handle. The queued version must contain a PCNet header telling where its ultimate destination is, and may also have in the header a decision as to where the next hop will be (i.e. to which node it will be forwarded next). The PCNet communication

software must then somehow get started, either by answering the phone, or by the system or another program or a human explicitly starting it up.

After a message has been received and the phone hung up (or possibly after the message has been received but while the phone link is being used for additional messages or other services, if the computer system is powerful enough to do so many things at the same time without slowing down the phone link), it must be decided whether the message is for local delivery or must be forwarded to yet another node. If forward, go to preceeding paragraph, else some delivery method must be chosen, such as appending to the start or end of a person's message file, writing as a new file with header stripped off and an anouncement of its arrival appended to the mail file, listing in hardcopy for manual delivery, or summoning an operator to telephone the addressee and recite it verbally.

Since all this occurs outside the telephone link, it can be handled at leisure by whatever combination of manual and automatic means the owners/operators of the node may decide upon. However, once the server-process accepts an FTP/MAIL request, it is mandatory that either an explicit negative-acknowlegment be given, or the message or file be correctly delivered to its intended recipient. Any node which accepts messages and then loses them without comment, will get nasty black marks in the master-network-directory, and other nodes will be warned not to trust the offending node, which will probably result in ostracism until the offending node is fixed. To avoid this, if an operator discovers that his system has crashed and obliterated from the disk some message entrusted to it, he (she) should immediately telephone the original sender to inform him (her) of the accident so that he (she) can re-submit the message to the network. (This of course implies that each node have some form of backup, such as a hardcopy log of message headers.)

DETAILED FUNCTIONS OF ALL THE LAYERS:

Radix-41 transmissions:
The mapping between the 41 characters of the Radix-41 characterset and the values 0 to 40, is nicely diagrammed in [Wilber, page 41], but can be summarized briefly. Open parenthesis, "(", maps to the value 0. Numerical-digit characters "0" thru "9" map to the values 1 thru 10 (note the offset, character "5" maps to value 6 for example). Uppercase alphabetic characters "A" thru "Z" map to values 11 thru 36. "*" maps to 37, "+" to 38, "-" to 39, and ")" to 40.

To be sure the reader understands the Radix-41 representation, the most commonly misunderstood part of our protocols, I will now present algorithms for converting to and from Radix-41.

Three characters of Radix-41, corresponding to the three numerical values they map to, are combined to make one 16-bit number. The first numeric value is multiplied by 41*41, the second by 41, and the third by 1, then these three products are added. (More efficient is to compute $V3 + 41*(V2 + 41*V1)$, which only uses two multiplications and two additions.) Going the other way, two divisions must be done somehow (the most efficient way is either a table lookup on 4 4-bit-bytes extracted from the 16-bit quantity, adding the four value-triples looked up in the four tables, or for each division a multiplication by an approximation to 1/41 followed by a slight correction).

To convert from Radix-41 characters to binary, first map Radix-41 to numeric values, then do the two multiplications to get a 16-bit number, then break that number into two 8-bit bytes. The mapping can be done efficiently by range checks for numbers and upper-case-letters (in either case a simple subtraction of a constant will finish the mapping) followed by a check for the other five characters if the range checks fail.

To convert from binary to Radix-41, first combine the two bytes into a 16-bit quantity (unless you are using the 4-table lookup method), then do the two divisions by 41, using the remainders and the final quotient as the numeric values in reverse order (i.e. the final quotient is the first value, the second remainder is the second value, the first remainder is the third value), and finally map to Radix-41 by a simple index into a table of 41 characters.

All this conversion between binary and radix-41 actually occurs as subroutines in about the second-next section in this tutorial, however the algorithms were discussed here so the reader understands what Radix-41 is and is not before getting embroiled in halfduplex turnaround and LTB framing.

(Note, an alternative to Radix-41, transparant 8-bit mode, is described in [Crane]. It will not be explained in this tutorial. The discussions below of HDX turnaround, LTBs, and dropping of the odd byte, apply only to the Radix-41 mode of transmission. Other sections apply equally to both modes of transmission.)

Half-DupleX (HDX) turnaround:
To avoid deadlocks when one or two characters at the end of a transmission is (are) garbaged, a 3-of-5 majority rule is used. Five turnaround characters are transmitted, and as soon as at least three of them have been received at the other end, that node begins transmitting

without waiting for idle line or the rest of the turnaround characters. REM made an arbitrary choice of open and close square bracket (in ASCII, octal codes 133 and 135, gotten by shift-K and shift-M on most terminals even if not shown on the key), which has been tentatively accepted by everyone in the protocol committee, and so has been written into our protocols. (If we discover a high rate of errors, and find two other characters not already used by our protocols but having much lower error-rate, we are willing to change this decision.)

To avoid overlapping an LTB with the tail of the half-duplex turnaround from the preceeding transmission in the reverse direction, or even in the same direction if the node starting to listen will see its own echo multiplexed with the beginning of the other node's transmission, each transmission begins with a series of five (or more) atsigns. (Atsigns were chosen, after considerable experimentation, for their remarkable property of fixing UART character-misframing, thus assuring correct UART sync for the LTBs that follow, even if due to hardware or interface problems the beginning of a transmission has the UART in some funny state such as transmitting half a character then being reset and immediately starting another character, or interleaving incoming data and echo of one's own outgoing data at the bit level (like my Beehive does when in halfduplex mode!!!)).

Finally, I will answer one of the most frequently-asked questions, which I always have to look up myself. Which node transmits "[" and which node transmits "]" The answer is that node 0 (the node that originated the call) transmits "]" and node 1 (the node that answered the phone when it rang, which implies an answering modem) transmits "[". Thus after node 1 answers the phone and starts up the PCNet communication-program, it transmits [[[[[to indicate it is listening for the first actual transmission, then node 0 transmits @@@@@ followed by one or more Link Transmision Blocks (LTBs) followed by]]]]]. Then node 1 transmits @@@@@ followed by its blocks followed by [[[[[. This alternation continues until one or the other node hangs up after either agreeing with the other node that they are done, or getting disgusted with the other node and giving up.

Link Transmission Blocks (LTBs):

In addition to the mapping from binary into Radix-41, each LTB is prefixed by an atsign, and followed by another atsign. Thus a total of six (or more) consecutive atsigns occur at the beginning of each transmission, adjacent LTBs are separated by two atsigns, and one atsign occurs immediately before half-duplex turnaround brackets. Atsigns serve two purposes, correcting UART mis-frame, and delimiting LTBs at the software level (in fact at

THIS level right here). Of the two atsigns between LTBs, the first one fixes the UART so the second will get through to software, and the second one (or the first if both get through) tells the software to finish any preceeding LTB that it was parsing and to start another as soon as a valid Radix-41 character (not atsign, not brackets) occurs.

An LTB as actually transmitted, therefore, consists of one atsign, 3*N Radix-41 characters (representing 2*N 8-bit bytes that are one TB), and one more atsign.

After a LTB (delimited by atsigns) is parsed, it is mapped into binary using the subroutine detailed earlier. If its length is not a multiple of 3, the mapping fails and it is rejected immediatey. Also if it is exactly 0 or 3 characters long (the former check is an easy way to determine the nothing between two consecutive atsigns that occur between LTBs, namely you pretend it is an LTB then reject it because it is too short to really be one).

If timing is critical, it may be necessary to buffer up a complete transmission without checking validity of LTBs, or even without parsing them at all. All checking can be postponed until after half-duplex turnaround. On the PET, using BASIC, even that trick loses, and it is necessary to make the main loop buffer up incoming characters without even checking for turnaround-brackets. When the main loop detects that no character has arrived, it lets a second loop, a co-routine, steal a moment of CPU time to perform one step in a loop that searches the buffer for turnaround-brackets. (Two routines are co-routines, as contrasted with one main routine and one sub-routine, if each returns to where the other left off, rather than one of them (the sub-routine) always being restarted.) Thus during the actual transmission the HDX-search co-routine lags far behind, then has time to catch up while the line is idle after HDX turnaround, at which time it finally sees the brackets and signals turnaround.

Odd/Even checksum:

This check, and the next three, can be done in any of the twenty-four possible sequences. The order shown here is random, but based on heuristics (actually prejudices). In actual fact the most efficient sequence is probably (1) drop-odd-byte (2) compare length (3) check orig/ans (4) compare checksums.

All the odd-numbered bytes are added up, modulo 256, and compared with zero. All the even-numbered bytes are added up and checked in the same way. (This can be done in a single loop by swapping the two running sums and always adding to the same variable.) If either result is nonzero, the block is rejected.

When transmitting a block, it's a little more

tricky because the two checksums obtained must be complemented before appending to the TB, and the appending must be "reversed" if there is an odd number of bytes so that the receiving node will get zero for each alternating checksum. See below for more on the extra zero byte added after the checksum when the TB has an odd number of bytes.

Originate/Answer flag:
The octal 200 bit in the first byte of the TB is the originate/answer flag. It equals the node number of the sender of the block, which is the complement of the receiving node number. (0=originate 1=answer, referring to original establishing of connection as well as to modem frequencies. Nodes not using Bell 103 protocol, for example radio links, must agree who is node 0 and who is node 1 before blocks can be exchanged.)

When receiving a block; if this bit isn't the complement of one's own node number, the block is rejected.

Drop odd byte:
If the TB-length, which is the complete second 8-bit byte of the TB, is odd, there is an extra zero byte (the last byte from Radix-41 to binary conversion, which is the next byte after TB-length has been exhausted) at the end of the TB which must be ignored. The easiest way to handle this is to decrease-by-one the local variable that tells the actual size of of the TB decoded from Radix-41 (before this step it will always be even, and if this step is performed it will then be odd like the TB-length in the second-byte already was).

It is reccommended, if you can afford the code, to first check this byte to be sure it really is zero, and to report an error if this check fails after the other three checks have succeeded.

Length:
The length as specified in the second byte should now equal the received-TB-size minus two (because the checksum bytes and the extra zero byte aren't counted, and the extra zero byte has already been deducted). If this step is performed before removing the extra zero byte, then the TB-length ROUNDED UP TO THE NEXT EVEN NUMBER is compared with the received-TB-size, instead. If the checksum has already been computed, and discarded from then received-TB-size, AFTER REMOVING THE EXTRA ZERO BYTE IF TB-LENGTH IS ODD, then you don't even have to offset by two, the two counts will exactly agree. In any case, when receiving a block, if the lengths disagree the block is rejected (thus two blocks concatenated, whose checksums will always combine to yield zero, or a block which has

been truncated but whose received part is all zero thus adding to zero, will be rejected. Even a steady stream of zero values, represented by the open parenthesis character, will be rejected, because the TB-length in the header of the TB will be zero whereas it should be at least two. Even if it it is two due to noise on the line, it is unlikely that a checksum of exactly 376 to offset it could be created by the same burst of noise without messing up the other interleaved parity.).

Sequencing of blocks modulo 8:
The first two bytes of each TB are the header. We've discussed everything in it except the two sequence fields. One (called SEQ) is the sequence number (of the TB it is in) modulo 8 and the other (called REVNAK in our current software -- this term as well as SEQ, RCVNAK, XMTNAK and XMTGEN may be changed in later documentation and software) is a sequence number for not-yet-received blocks traveling in the reverse direction. The latter is of interest to the transmit pipeline when we receive a TB containing it, and is taken from a globally-available parameter in the receive pipeline when transmitting a TB. As it sits inside (first byte, mask 007) the TB, I call it REVNAK which means "REVerse Negative AcKnowledgement". It comes from the RCVNAK variable of the transmitting node, and is stored in the XMTNAK variable of the node that receives it. Thus it is affiliated with the REVerse process to the one that is actually passing it across the phone line.

The SEQ field refers to the TB it is in, identifying its sequence number modulo 8 (i.e. a 3-bit number). When constructing a TB for transmission, a local variable XMTGEN is used to generate this field, then XMTGEN is increased by 1 (modulo 8) to be ready for the next TB to be constructed. The XMTGEN variable in each node starts at zero, thus the blocks sent by each node are numbered 0,1,2,3,4,5,6,7,0,1,2,3,... This block number, a 3-bit field in the TB header (octal mask 070 in the first byte), is then used by the receiving node to decide whether it has already gotten it, is expecting it now, or isn't yet ready for it. The first and third cases are generally indistinguishable. In either case the block is ignored except that since the checksum etc. all are ok it is known to be A BLOCK from the other node hence the REVNAK field can safely be copied to the XMTNAK variable the same as if it was a block that was accepted. (Normally the storing of REVNAK into XMTNAK is done just before the block sequence number is checked, but of course after all parity+length+extrazero+originate checking is done.)

If the received block is numbered exactly the same as RCVNAK (oldest still-not-received

block, initialized to zero) then the header and checksum are stripped away and what's left is passed up to the pure-binary stream (if the TB length was exactly two, then there are exactly zero bytes of data, so nothing is passed up) and the RCVNAK is increased by one and reduced modulo 8. These two steps, passing up data and updating RCVNAK, must be properly synchronized (if this software has multiple processes active at the same time) so that it is impossible to get two copies of the same block and accept them both.

If a node is capable of buffering more than one block at a time, then it is possible to get an acceptable block that isn't the very next one but which can be saved at the receiving node until the intervening one(s) get (re)transmitted to fill the gap. Then when the one matching RCVNAK finally arrives, after updating RCVNAK another check is made to see if the one matching the new value of RCVNAK has already arrived, in which case it too can be unbuffered and passed up to the pure-binary stream, RCVNAK updated again, and the test for already-arrived-RCVNAK-equal block repeated until it finally fails. (At most four blocks can be receivable at one time, thus at most three blocks already buffered can be emitted when the one before them finally appears. This limit of 4-out-of-8 is absolute, there is a counterexample (or scenerio) that demonstrates that with a window of 5 or more out of 8, a block can be mistaken for one 8 earlier or later and thus completely destroy function of the link. This fact won't be apparent until we've finished discussing the function of the REVNAK field and the XMTNAK variable.)

The function of the RCVNAK variable, as observed from the outside (in particular from the transmit pipeline which may run asynchronously at the same time as the receive pipeline when in full-duplex mode) is now describable. At any moment it equals the number of the oldest not-yet-received-and-completely-processed block.

Each time the transmit pipeline sends a block, at the last possible moment before transmission (just before converting to Radix-41), it copies the RCVNAK variable into the REVNAK field of the header, and recomputes the checksums. (Note this also occurs on retransmission of a block, thus it is always current in realtime at a moment just before the start of the LTB output.) The effect is that the REVNAK field is an implicit acknowledgement of all blocks (in the other direction) preceding the one numbered REVNAK, is an explicit negative acknowledgement about the one numbered exactly REVNAK, and is a "no comment" on all later blocks that might have been received and buffered or might not have yet been received.

We may define the time that an acknowledgement really happens to be the instant when the node receiving the block, after checking parity etc., stores the REVNAK field into the XMTNAK variable. Thus at any moment, the XMTNAK variable in a node equals the sequence number (modulo 8) of the oldest outgoing block that hasn't yet been acknowledged by the other node. When XMTNAK equals XMTGEN, it means that all outgoing blocks constructed earlier in this session have been sent and acknowledged. Otherwise it means the blocks numbered from XMTNAK up to but not including XMTGEN are either somewhere on the round trip out (as blocks) and back (as implied acknowledgements), or have been lost somewhere due to line noise or lost characters due to too-slow software or other problems. A heuristic algorithm in the transmit pipeline uses XMTNAK and XMTGEN as well as any other information available, to decide whether to retransmit a block that still hasn't been acknowledged, transmit a new one while waiting a little longer for the wayward block to get acknowledged, or perform half-duplex turnaround. In the simplest implementation, which is the default, a window of 1-out-of-8 combined with half-duplex mode makes the choice obvious, namely upon getting turnaround the program checks XMTNAK against XMTGEN. If XMTNAK+1=XMTGEN (mod 8) it retransmits XMTNAK block. If XMTNAK=XMTGEN it constructs a new block from queued data (if none there's some hair, consult REM and look at a listing of the program PCNSR3.SAI for details). The other six possibilities are impossible, thus indicate programing bugs or faulty hardware inside one node or the other.

<u>Pure-binary stream:</u>

The data portion of TBs arrives as a stream of bytes (or as successive arrayfuls of bytes, one array from each TB). It may be fed directly to the PPSB parser, or buffered first. In any case there should be an explicit control point through which all incoming pure-binary data passes, and another similar control point for the transmit pipeline, so that software below this level can be made almost totally independent of software above this level, and so that during debugging a trace can be installed to see what is getting thru these two points (rcv and xmt pure-binary) in the software. Why break the software at these two points? First, because it is the only place in the entire software, other than the UART interface, where a conceptually-clean place gets 100% of the data flowing through the software. Second, it is a natural place to splice different protocols together. During initial testing of the bottom half of the protocol it is common to put the software in loopback mode by feeding all incoming data right back out. Slightly later the lower half can be used almost-stand-alone for file-transfer using a simple

top-half kludge (halter-top?) that is initiated manually at each end or semi-automatically by a restriction on the characterset and a convention for detecting end of file. (The program PCNSR2.SAI is an example of this.) Another possibility is to replace the bottom half by some commercial data network that guarantees 100% perfect transmissions of 8-bit bytes, or by the bottom half of the DIALNET protocol, or by one of the new bit-synchronous communications protocols.

PPS Blocks:

At the node level, data is explicitly broken up into PPSBs, each of which is one segment of a PPS (Process-to-Process Stream). If necessary these can be multiplexed (complete PPSBs from one PPS located between those from another PPS, but a PPSB isn't broken internally, thus a PPS can be broken only at PPSB boundaries). The very first byte of data in the very first LTB, during any given session, is also the first byte of header of the first PPSB in that session (this applies separately to the two pipelines, one in each direction, transmit and receive). Once the pipelines have started, the PPSB-LENGTH field in each PPSB determines where it ends, and the next PPSB starts immediately after (the next byte of the pure-binary stream). Since the data in the pure-binary stream is 100% accurate, no errors can happen in parsing (except by failure of the equipment, which is assumed to not happen), so no other framing is required. In the default mode, with LTBs and PPSBs synchronized, there is additional redundancy that can be used to detect such "never-happen" failures.

The first PPSB in any PPS has two fields not in later PPSBs, namely a SOURCE-PROCESS and DESTINATION-PROCESS number, each 8 bits. The DESTINATION-PROCESS field is used to direct the PPS to the correct piece of code to start processing the PPS. After the PPS has started, that code passes control to whoever will really be reading the PPS, which may be a specially-generated entry point, or a throw-away sink if the PPS isn't acceptable for receipt. The SOURCE-PROCESS field is used in case the DESTINATION-PROCESS wants to open a PPS in the reverse direction, it uses the SOURCE-PROCESS of the original PPS to fill the DESTINATION-PROCESS field of the reply PPS.

Every PPSB has the following fields: BLOCK-LENGTH (used to parse PPSBs in not-delimited end-to-end format of the pure-binary stream), PPS-NUMBER (used to identify which PPS it is part of), BLK-NUMBER (sequence number within that PPS, starting at 0 for the first-PPSB which has the two extra fields described above), and LAST-BLOCK-BIT (1 if this is the last block of the PPS, so that the PPS it is in will be formally closed at the end of this PPSB, otherwise the PPS will be kept open waiting for more PPSBs).

Normally the block number is almost redundant. If it is zero for a PPSB that isn't part of a known PPS, it is assumed to be block 0 (rather than block 8 etc.), and the DESTINATION-PROCESS field is checked for legal values (0 and 1 currently, except when it is a reply to an IFTP), otherwise it is a non-first block and can be ignored. Note that at most 8 PPSBs can be sent in a new PPS before getting positive confirmation that the other node has actually accepted the PPS, otherwise block 8 might be confused with a new block 0 attempting to open a new PPS.

In advanced nodes with multiple link-levels over separate dialup lines to increase effective bandwidth, the PPSBs for one PPS can be distributed to different link-levels, and the block numbers used for reassembling them in correct sequence. But this is not likely to happen for quite a while!

Note that if the process originating (SOURCing) a PPS realizes after it has already sent the last byte in a PPSB with LAST-BLOCK-BIT zero, that the PPS should now be closed, it can simply send a PPSB with BLOCK-LENGTH equal to exactly 3 (i.e. 3 bytes of header and 0 bytes of data) and with LAST-BLOCK-BIT set. Thus it is never too late to close an open PPS. There are also methods to abort a PPS rather than closing it, usually to signal some error condition. This capability is included in the control messages listed in [Maas WR2].

Process-Process Stream:

After deciding where to send the data part of a PPSB, and stripping off the 3-word or 5-word header (5 for the first PPSB in a PPS, 3 for later PPSBs in the same PPS), the remainder of the PPSB (up to 255-3=252 or 255-5=250 bytes, minimum 5-5=0 or 3-3=0 bytes) is passed up to whereever it was supposed to go, to be interpreted further by whatever program is located there. Two such programs, the Control-listener, and the FTP/MAIL server, are defined presently.

Process#0 -- meaning of link-control messages:

The Control-listener gets any PPS that is addressed to destination process 0. Each PPS must consist of exactly one PPSB, with the LAST-BLOCK bit turned on (1). Any multi-PPSB addressed to process 0 is considered a violation of protocol. Each PPSB to the Control-listener consists of exactly one control message. The first byte (of PPSB-DATA) is the control opcode, and any remaining bytes are arguments. A list of currently designed control messages are in [Maas WR2]. Almost all them can be ignored safely by unfancy nodes.

Process#1 -- protocol for transferring a file (message or other):

All PPSs directed to DESTINATION-PROCESS 1 are fed initially to the FTP/MAIL listener. After a PPS has been opened, a slightly different entry address will probably be set up so that the rest of the PPS can be processed without constantly rechecking for same/different PPS number. Usually a dispatch table for currently-open PPSs will exist in the PPSB parser, and simply changing an entry in it will switch a stream to a different piece of code without affecting anythng else. Any new PPS addressed to process 1 will go to the original FTP/MAIL listener, which will then note that an FTP/MAIL is already in progress and either ignore or signal abort unless the computer can handle multiple simultaneous file transfers (a rare situation even on large computers).

The general protocol for transferring a file is as follows: The node wishing to send the file first sends a IFTP (Initialize File-Transfer-Process) request to process 1 of the node that will (maybe) receive it. That node then checks to see if the file is small enough to fit in available storage, and if not replies with a NO-I-WON'T IFTP message. If, however, the IFTP is acceptable, it replies with an I-WILL IFTP message. Then on the same PPS as the original request, the sending node transmits an FTP/MAIL opcode of PLEASE-DO TAKE-THIS-FILE followed by actual data of the file all the way to the end of the PPS which is then closed. Finally a reply of either I-WILL TAKE-THIS-FILE containing a checksum of the data received (zero if no checksum was computed) if it was successful, or I-WON'T TAKE-THIS-FILE with some error code if not. The receiving process should actually safely close the file it is writing into storage, before sending the I-WILL ... <checksum> confirmation of completely successful transmission. After the transmitting node gets the confirmation, and verifies the checksum is correct or zero, it may safely delete the copy it was reading from.

Local interface between FTP/MAIL-server and mail-forwarding queue:

This is mostly up to the operators of the computer.

Local mail-forwarding program:

General guidelines for forwarding of messages are in [Deutsch], [Bourn], and [Maas] (respectively, MAIL/FTP protocol, worldwide-addressing, and final-delivery). How forwarding is actually accomplished is up to the individual operator(s).

Local mail-creating/editing/receiving program(s):

When creating a message or initializing an FTP, the correct header should be put at the start of the file. The forwarder and the protocol program can then handle it as a chunk of data, examining the header when necessary to see what to do with it next. When delivering the message to the addressee, it is optional how much of the header to keep, how much to edit to make it prettier, and how much to purge. All this is up to the operator(s) of the system.

SUMMARY:

We hope that this tutorial has helped the reader understand most of internal workings of the program that at one end of a phone connection automatically maps files down through the layers of protocol and transmits them out the telephone line, and at the other end receives the LTBs and maps them up through the layers to construct a copy of the file. Formal specifications of these communication protocols are given in [Crane]. Specifications of how a message-forwarding network is built upon these protocols are given in [Wilber], [Bourn] and [Maas WR2]. These and other documentation and tutorials from the PCNet Committee are available at a nominal charge to cover the cost of reproduction and mailing. (Contact the author, or any of Dave Caulkins, Mike Wilber, Ron Crane or Peter Deutsch.)

At the time of final-editing of this tutorial (1978 January 16), protocols are working on several PDP-10 computers, and this software has been partially transferred to the PET (using BASIC) and to the Altair (using assembly language). It is hoped that we can have several micro-processor nodes fully-working and doing useful electronic-mail service by the time this paper is presented at the fair.

REFERENCES:

[Bourn] "A Proposal for Addressing Stations of the Personal Computing Network", by Doug Bourn, an internal working paper of the PCNet Protocol Committee, available by photocopy means only.

[Crane] "PC Net Communiation Protocols", by Ron Crane et al, a working paper of the PCNet Protocol Committee, in press (online file PROTO.PB1).

[Deutsch] "Mail Transfer and Forwarding", by Peter Deutsch, a working paper of the PCNet Protocol Committee, in press (online file SERPRO.TTY).

[Maas] "Final Delivery of Messages", by Robert Maas, an internal working paper of the PCNet Protocol Committee.

[Maas EXPERI] "Experiments Sub-Committee of PCNet Committee - Status Report", by Robert Maas, continually-updated file (online file EXPER.I).

[Maas FLO] "Flowcharts Showing Data and Control Interfaces between Modules in PCNet Protocol/Software", by Robert Maas, in press (online file PROTO.FLO).

[Maas RAD41] "Explanation of Radix-41 in PCNet", by Robert Maas, in press (online file RAD41.WRU).

[Maas WR2] "The Design of the Personal-Computer Network (PCNET)", by Robert Maas, in press (online file PROTO.WR2).

[Wilber] "A Design for a Network of Community Information Exchanges", Mike Wilber, presented at the first West Coast Computer Faire.

MICRO'S IN THE MUSEUM-A REALIZABLE FANTASY

DISNEYLAND ON YOUR DOORSTEP?

JIM DUNION, THE AMERICAN MUSEUM OF ENERGY, P.O. BOX 117, OAK RIDGE, TN

Abstract

The American Museum of Energy in Oak Ridge, Tennessee has recently begun a program to introduce and develop micro computer technology in a museum environment.

The museum has some data processing requirements, which, although not very elaborate, form certain pre-requisites for additional computer activities.

Micro-computers are being utilized in museum exhibits in two ways. Some are complete, stand-alone, terminal oriented exhibits. These exhibits allow certain energy topics to be introduced to the public, as well as provide a means for receiving direct feedback as to public opinion. Longer range plans call for building micro-computers directly into certain interactive exhibits.

Beyond the direct utilization of micros in exhibits, the museum is an active resource for computer technology. This is accomplished in several ways: by offering beginning classes in micro-computer technology and programming, by sponsoring computer activity events at the museum, and by actively soliciting and promoting community participation in museum activities and developmental projects.

Moving Micros into the Museum

The American Museum of Energy in Oak Ridge, Tennessee (AME), has recently initiated a program to introduce and develop micro-computer technology in a museum environment. The need for such a program ties in very closely with the current energy situation in the United States. The AME operates under a contract from the Department of Energy, and has as one of it's main functions to increase public understanding of science and technology, with particular emphasis upon energy issues. Since the energy program that President Carter announced contains a major shift in emphasis away from developmental energy programs towards fossil fuel usage and programs towards fossil fuel usage and conservation methods, the need to educate the public as to the real energy situation and what they can do about it is more urgent than ever. It is the feeling of the museum staff that microcomputers can play a major role in increasing the effectiveness of this educational process.

Studies performed at this museum and others indicate that a great deal of care must be taken in planning exhibits if they are to successfully transfer meaningful information to a visitor. The average time spent at an unattended exhibit varies between 30 seconds and several minutes. Thus exhibits cannot rely too heavily on written text, but must incorporate interesting graphics and audio-visual presentations. The current tendency at science and technology centers is to design interactive exhibits with which visitors can participate. Most exhibits currently manufactured employ hard-wired circuitry and electro-mechanical devices, making them expensive to design, implement and change. Again, the feeling at AME is that micro-computers can make a significant impact on exhibit and display technology. So, the AME decided to take the bull by the horns and instigate a program to develop and implement this technology.

Who's Involved

The primary motivating force behind this move is Robert F. Content, the chairman of the Museum Division of Oak Ridge Associated Universities. He is well acquainted with computer technology, having served as asst. dir. of the Lawrence Hall of Science at Berkeley. During his tenure there, he was responsible for developing the Hall's public access to computers program which has proved to be extremely successful.

However, at the Lawrence Hall of Science, as well as at most science and technology centers, there is an almost exclusive reliance on mini-

computers. It is now time to put micros in the museum.

The first step in this project was to acquire some computer equipment. The museum already had two Wang 2200's, and a PDP-8, but these are utilized for internal data processing and in a couple of existing exhibits. To get things going, four SOL's from Processor Technology and two Silent 700 terminals from Texas Instruments were purchased.

Next, a specialist in microcomputers and personal computing, Jim Dunion, was hired to head up this developing program. He brought with him a Compucolor 8001 with a floppy disk. At that point, the museum faced a somewhat unusual situation, equipment rich-people and software poor. With that in mind, the watchword of our program became - cooperative development. In a nutshell, we hope to solicit help from the local community in Oak Ridge and Knoxville in developing microcomputer technology in the museum. We provide the equipment , the educational training, and the projects. In turn, we hope to receive time and software from interested individuals.

Major Areas of the Program

We have broken our program down into four main areas.
1. Social
2. Educational
3. Developmental
4. Communal

The social aspect of our program is intended to generate interest in the museum's computer activities. A little showmanship, if you will. We have started a series of computer activities nights at the museum. These sessions are aimed towards familiarizing the general public with the capabilities and benefits of personal computing. Each session is keyed around a central theme, such as the recreational uses of computers or computers and art. A typical night will consist of a movie about some aspect of computers, a technical talk (in terms that laymen can understand), several systems running demonstration programs, and refreshments.

Our educational program will proceed in several discrete steps. Initially we are holding a two to three hour seminar called, An Introduction to Personal Computing. After presenting this talk several times, we will initiate two short courses,
1. Understanding Personal Computers
2. Beginning Programming in Basic

After these classes have been conducted a few times, we will then begin offering more specialized topics such as:
1. Game Playing with Computers
2. Advanced Programming Techniques
3. Special Projects
4. Computers for Kids

One of our goals for this part of our program is to interest local programmers and personal computing enthusiasts in teaching these and other courses. Hopefully also, some of the students that we train can then turn around and teach future classes.

The developmental portion of this program promises to be one of the more interesting aspects of our approach. We have identified four major research areas of interest for developing micro-computer technology in the museum. We have begun acquiring additional hardware to configure systems in different work stations for these projects. As local students, hobbyists or just interested individuals begin working with us, we will coordinate special assignments in these research areas to provide some training and experience for the workers, and of course, exhibits for the museum. The research areas are:
1. Verbal Information Systems '
2. Color Graphic Display Technology
3. Electronic Bulletin Boards
4. Automated Exhibit Technology

Verbal Information Systems

The primary goal of this project is to develop an energy information system that anyone could walk up to and ask certain questions about energy technology or energy policy issues. The catch is, we want the input to be spoken language, and the output to be synthesized speech, clearly not a trivial task. This project will involv voice recognition, speech synthesis, natural language understanding, data base management, and artificial intelligence. In addition to a SOL computer we will add a voice recognition unit and a speech synthesizer for the initial system configuration.

Color Graphic Display Technology

One of the systems we have available is a Compucolor 8001, which is a color graphic device. We already have a number of interesting games and demonstrations for this system, but the feeling of everyone who sees this system is that we are just scratching the surface. We need to develop programs that make the creation of displays and games easier. Such programs might include a complete graphics package (vector graphics are already provided), a rubber-band drawing system, an animation package, etc.

We already have some specific exhibits in mind to use this system. Particularly in light of the fact that Intelligent Systems Corporation, who manufactures the Compucolor, is preparing to announce the Compucolor II, a scaled down version rumored to sell for under $1000.00. We plan to place an orientation device in the lobby of the museum that will present certain information about the museum. One thing that will be displayed will be a floor plan of the museum. Visitors will be allowed to specify certain rooms of the museum on the display (probably using a light pen), and then receive more information about the exhibits in that room. We feel that if visitors have an overview of the museum before they start their tour then the visit will be more meaningful.

Another use for this system will be to provide interactive simulations of such things as global energy systems, household energy conservation, nuclear reactions,etc.

Finally, we are designing several games oriented around energy. These are right now called The Energy Game and Embargo.

The hardware for this project area is already complete.

Electronic Bulletin Boards

The Electronic Bulletin Board will provide a message posting system, text editing and word processing, newsletter preparation and information about on-going projects (both energy and computer related).

We are currently looking at text editing and word processing software for this project. In addition, we plan to install a telephone interface so that the system may be accessed remotely. Currently, our plans call for using one system to act as the phone access and also to control a hard disk mass memory device. This in turn will interface to the other systems for both remote access and downline loading of programs.

Automated Exhibit Technology

The purpose of this project will be to provide expertise in controlling electro-mechanical devices. Right now this is the most speculative of our projects. Our first goal will be to develop a small, inexpensive robot to work with. We will probably use the basic design presented by David Heiserman. We also want to install a visual input system. Naturally, we will be very active in the United States Robotics Society.

As is probably apparent from the description of these research projects, we hope to make them exciting and interesting enough so we can attract a lot of local talent to work with us.

The Museum as a Community Resource

As part of our program to attract co-workers, the museum is actively promoting the idea that we are a community resource. We are doing this in several ways. First, we are sponsoring and supporting local computer hobbyist groups. There is a club in Knoxville that has been active for some time now, and a club is just forming in Oak Ridge. The facilities of the museum are available (to some degree) to these groups. The actual mechanisms of this availability will have to be worked out as the program proceeds.

We have also initiated a lecture program that is geared towards local civic groups. The main topic is Computers and Energy (with a huge plug thrown in about the museums computer activities, naturally).

Finally, we are trying to acquire as many publications in the personal computing field as possible, so that they may be available for local hobbyists. We are also gathering as much software as we can, software in the public domain that is, and will transfer this software to interested parties. We are particularly interested in software relating to energy or simulations related to energy.

HELP

    The museum's program is still in
the developing stage. One thing we want
to emphasize is that we very much want
and need as much help as possible in this
endeavor. Advice, software, hardware,...
are actively solicited. We hope to
expand this into a national program, and
towards this end we will be publishing
a newsletter tying together the personal
computer movement and the Association
of Science and Technology Centers (a
group of 80-100 such institutions). This
newsletter will be available to hobbyist
clubs, schools, etc.
    The impact of micro-computers on
our everyday life continues to grow.
We think that their introduction into
a museum environment will be very
significant indeed. Perhaps in a few
years museum exhibits will resemble
Disneyland rides. With your help, and
our efforts, we'll put micros in the
museum.

THE MARIN COMPUTER CENTER - A NEW AGE LEARNING ENVIRONMENT

David and Annie Fox
Co-Directors of Marin Computer Center
70 Skyview Terrace   Room 301
San Rafael, CA   94903

Abstract

Marin Computer Center is a project of Ulenar, a non-profit, educational corporation, whose main goal is to bring the wonders of advanced technology (computers and the like) within the reach of all people.

We have set up 10 microcomputers in what was formerly the library of Oakview School in San Rafael, California. In a spacious, well-lit room, with beamed ceiling, orange carpeting a many plants, we've created the kind of comfortable environment that has never before been associated with computers.
We will describe how MCC came to be, what it is, and where we plan to take it.

How It All Began

Marin Computer Center was seen as a vision at first. We came upon the idea - or it found us - quite unexpectedly in mid-August of 1976. How strange it seems now, and yet very natural all at the same time.

To say that computers and the world they represented was far from the world that we inhabited then would be a gross understatement. At that time in our lives, and for several years prior to that time, we were "spiritualists" - lovers of the occult, psychic realm - followers of numerous "personal growth" excursions - always seeking. We considered ourselves very much the "humanists" - with our respective careers of teaching and counseling.

We felt that not enough people were coming in contact with new ideas about themselves, not enough people were growing in their personal lives. The question was, how to introduce the vast majority of Americans to themselves. We took a look around and noticed the beginning boom of video games. What if we developed a video game in which people could learn more about themselves and their relationships with others in the process of playing? Of course, the stated purpose of the game wouldn't be personal growth, that would just be a side effect of playing it.

From this idea we jumped to a fantasy of a huge complex similar to Disneyland. The main difference would be in the participation level of the visitors. Disneyland is fun but it is essentially a place where they "do it to you". You watch animated dolls while riding on a boat or go for a submarine ride and watch sea serpents looming at you. No one is given an opportunity to interact with the environment, to play with the environment in a way where some new and unique learning experience would result. We envisioned a technology playland where all this could happen. To actually "person" the deck of the USS Enterprise with other visitors and make contact with other worlds. To warp your own intergalatic vessel around the universe while looking through a three dimensional viewscreen and experiencing the force of acceleration. To feel weightlessness in a zero gravity room. The movies "Westworld" and "Futureworld" are the closest we've seen to this idea. Of course, the conflicts of man versus machine in those films represent the fears we wanted to help people overcome in order to make the most of technology.

With our long range goals set, we had to find something which we could accomplish with today's technology. The concept of the Marin Computer Center was born. We embarked -- whole-heartedly without a backward glance. It seemed as if we had been running full steam in one direction - then one day screeched to a halt for no externally apparent reason - and zoomed off at twice the velocity down a new road!

It may seem strange that two people with no technical background would be audacious enough to enter the hallowed grounds of "computerland", but somehow our naivete has served to make the whole thing unique and appealing in the eyes of others.

We created Marin Computer Center because we felt that there needed to be some educational facility that would bridge the gap between peoples' fears and their natural

curiosity about computers. It seemed evident to us that the rapid growth of the personal computing industry would result in a "computer in every home" by the early 1980's. Judging that as an inevitability and evaluating the prevailing attitude about computers, it seemed obvious that people needed a painless way to ease themselves into the Computer Age.

Many people feel that computers are cold, dehumanizing instruments of totalitarianism. The image of Big Brother and the "Computerized Society" seem to go hand in hand. At least that has conventionally been the fictionalized view. We would be the first to admit that in the recent past computers have been used in ways that have resulted in general feelings of powerlessness and compartmentalization. However, it is important to distinguish between computers (the species) and how they've been used. In other words, it is short-sighted to condemn a device simply because of the misuse and abuse it has suffered at the hands of people with something less than the "common good" in mind.

Alarmists and political paranoids argue that computers are potentially dangerous in that they can be used to store incredible amounts of very personal data and then recall that information at an astonishing rate. They become uneasy at the thought of the "Master Computer" controlled by the CIA.

The Computer is a powerful tool. And it, like many powerful tools throughout history, has been used and misused by people who seek power for purposes of both good and evil.

When the printing press was first invented, the church began to fear its use for the purpose of widespread propaganda against Church Doctrine. They launched their own campaign against the machine, condemning it as a tool of the Devil. One would have to admit that there have been some pretty libelous, degrading and socially unredeeming things that have been presented to millions of people in the form of the printed word. However, one would not be hard pressed to think of just a few of the beautiful, inspiring, and beneficial things we have experienced through our exposure to words in print.

So which is it? Tool of the Devil or Invention of Enlightenment? Actually the printing press is neither. The prnting press is just a machine that prints words on paper. The discussion is arbitrary and meaningless. The same is true of the debate about the potential joys and evils of a computerized society.

Computers are here to stay. And the general public needs to start taking responsibility for its own personal participation in the world of computers. Because they are such "all purpose" machines, it is up to us to decide which of their various purposes are ones that we want to support.

Marin Computer Center's main goal is to "introduce people of all ages to computers and the advanced technology which they represent in order that anyone might begin participating in the process of computer assistance for society".

When we started we felt certain that our objectives were valid and would serve a valuable function in this society. But lofty goals and innovative plans are meaningless if they cannot be manifested in the physical universe. And in order for our dream to take a real form we needed money.

Our quest for capital led us to dozens of private foundations. We spent six months peddling our grant proposals with no success to speak of.

For long periods of time our goal seemed extremely distant and as likely as a winning sweepstakes ticket. In the face of such overwhelming odds and dispair, were we discouraged? We sure were! Weeks went by and nothing happened - no forward movement; our plan was stagnating and so were we. Many times it seemed as if we continued with our phone calls and letters just to spite all the people who thought we were crazy to persist with an idea that couldn't get off the ground. And I'm sure we must have been. Crazy enough to continue persisting even though the Foundations weren't exactly beating a path to our door, we knew it didn't mean that money couldn't be obtained through another source.

So we did what most people do when they need money - we hit the banks. And lo and behold, with the help of a friend (with more financial credibility than we had) our loan application was approved!

That was in July of 1977 - a full eleven months after the whole idea was hatched!

In the two months that followed, we rented 5,000 square feet in a beautiful school building, ordered and received nine Sol-20's and one Equinox, obtained some programs, invited 300 people to an Open House, placed three ads in local newspapers - and held our breath.

Visiting the Center

On September 10th we opened our doors - at long last Marin Computer Center had crossd over into the physical universe. Over 700 people showed up for our Open House celebration and during the past six months they have steadily continued to come. Little children with their parents, neighborhood kids stopping in after school, handicapped children and adults, older people - all with their interest in computers to guide them.

MCC was created to give people an experience of computers and the advanced technology which they represent. In the first few months that we've had our doors open, we have in fact been providing that kind of experience in addition to many other kinds of experiences that we had not anticipated.

For example, on Saturdays MCC provides a place for families to come together in an attractive and calm environment for a unique "learning experience". We see them come in, wide-eyed and slightly apprehensive. They have heard about this place from friends of theirs (who had "a terrific time") so they thought they'd see for themselves. They don't have any idea what to expect and frankly, they've got their guard up. We greet them and make them feel welcome. We acknowledge the uncertainty they are exuding and they begin to feel that they don't have to pretend that they're feeling at ease when they're not - their anxiety is understood and then they begin to relax.

We tell the newcomers about our set-up, in terms that they can relate to. We talk about why we've created this center and that we're glad that they've come to explore. After talking for a while, we suggest a computer game that might interest them, load the machine and let them settle in for the fun of confronting a new learning experience.

Adults and children relate to new learning situations in totally different ways. We have learned much from observing people with computers. Children seem to be very much attracted to the CRT terminal - because of their familiarity with TV and home video games, children between the ages of 7 and 10 feel very much at home with our microcomputers. Their attitudes towards the computers are open, eager and an almost matter-of-fact acceptance of the things that the technology of today has managed to accomplish. Older children, while equally open, seem to be more apreciative of the wonder of it all. They have reached a point in their own cognitive development to be able to imagine in abstract terms what a computer is and how it manages to do what it does. (There is a greater preponderance of 14 year old boys who frequent the center than any other age group.) So although children of different ages may be experiencing the computers differently, they all are unanimous in their enjoyment of and fearless approach to the machines.

Adults, on the other hand, are less likely to welcome the challenge of this particular "unknown" with open minds. Adults come to the center with the whole gamut of preconceived attitudes, ideas and beliefs about computers. Their experience may have been in the form of a mistaken IRS refund, a cancelled magazine subscription that kept on coming or other annoyances that have been blamed on a "computer foul up". With these kinds of things in mind, many adults come to the computer center ready for a fight, it seems. They are sour-faced individuals who wish that the animal "computerectus" would go on the endangered species list and not survive. Then there are women in the 35-50 age group who feel intimidated by the "superior" intelligence of computers. They are embarassed that the computer will make them look foolish by knowing more than they do. And finally there are the older adults (in the 50-70 age range) who are bewildered by it all. They feel that the world is just moving too quickly and that they are being left behind.

After a direct experience with computers, one's fears are seen as groundless. Then the individual creates the opportunity for him/herself to really explore the computer as a new personal medium of creative expression.

One of the ways both adults and kids can do this is by taking one of our classes in computer programming. The class is really an introduction to microcomputers and the computer language BASIC. The course covers a brief history of computers - through vacuum tubes to transistors to integrated circuits to large scale integration; discussion of how a computer works and then right into learning the language and creating your own programs.

The class is for absolute Beginners - no prior knowledge is assumed or expected. Since we personally tiptoed into the field without the usual prerequisites we fully understand and empathize with the fear and general uncertainty people bring with them into our classes. Because of this empathy we are particularly good at creating a safe

learning environment for them to explore these "intelligent" machines.

Graduates of our courses have gotten right into the process of using computers in their lives for more fun, profit, and efficiency. Some examples are: the man who created a program to calculate the milk production of his goats, the teacher who used the course to create specialized curriculum for his junior high school deaf students, the woman who was in charge of the reservations department in a large airlines and wanted to have more knowledge of computers to increase her feelings of effectiveness in her job and the 14 year old boy who has created computer programs for the games of Yahtzee and Battleship.

One's success at survival has always been based on the ability to adapt; a willingness to change. With the world's increasing rate of change we've all got a challenge just to keep up with it. And more important than keeping up with it is to be a part of that process of change. We at Marin Computer Center are giving people a wonderful opportunity to participate in that area of change in today's world known as "Computers". By directly interacting with computers, people begin experiencing new feelings of freedom and confidence, replacing their former fear and confusing overwhelm.

All of these people have an experience at the Computer Center which enables them to step outside of their preconditioned feelings of hostility, fear and confusion and enter a new world. A world that is not the de-humanizing robot world that they first imagined - but a world of people and learning and change instead. It's an exciting new world, and there is a place in it for everyone. The child in all of us is fascinated by computers - the "New Age Toy, Tool and Servant" of Humankind.
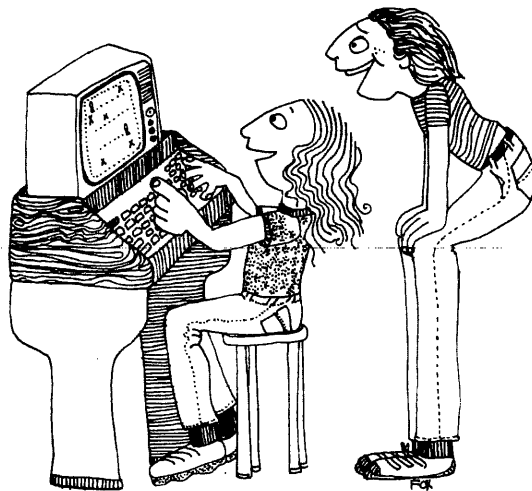
PERSONAL COMPUTERS AND LEARNING ENVIRONMENTS:

HOW THEY WILL INTERACT

Ludwig Braun, Professor, Department of Technology and Society,
SUNY at Stony Brook, Stony Brook,N.Y. 11794, 516-246-8418

Educational technology has gone from the log (for Plato and his student to sit on) in 400 BC through the book in 1454 when the Gutenberg Bible became the first mass-produced book, to the personal computer whose genesis is reckoned to be either January 1975 when the Altair was announced, or April 1977 when the PET was announced . (depending upon you definition and your loyalties).

Until now, students and teachers were involved in information trans- mission; while, now, they can begin to think in terms of information processing--with the enormously- increased intellect-enhancement this implies.

Because of the low cost and portabil- ity of computers like the PET and the TRS-80, educational computing sud- denly has become much more attractive than ever before. These computers are totally self-contained and need only an a-c outlet to operate. This means that learners can use computers anywhere without worrying about tele- phone locations, etc. The teacher can plan to bring the computer into the classroom or the office. Because personal computers are portable, society can think about putting them in neighborhood or school libraries so the kids (or adults) can sign them out. The $600 price is low enough that parents can think ser- iously about giving their children computers for Christmas or as birth- day presents.

The graphic capabilites of the PET and the TRS-80, although limited compared to very expensive graphics facilities, are very impressive at the price. Art teachers can give their students a new flexible medium to explore. One important advantage of the computer as an art medium is that the child can convert art in his mind into visual images without the manual dexterity required by clay, paints, crayons and other media.

The capability to speak and to re- cognize spoken words (a la Speech- lab and Computalker) provides alterna- tive input and output modes to permit children or handicapped people to use computers; and provides teachers of languages with computer support in a new and interesting way.

One of the most exciting capabilities of personal computers (to me at least) is the ability to connect to the real world through analog-to-digital(A/D) and digital-to-analog(D/A) converters. This capability permits us to develop simulations which provide much more realistic learning experiences than is possible by more traditional com- puter methods. With A/D and D/A con- verters, the digital computer becomes an analog computer with all the advan- tages of the digital computer and none of the disadvantages of real analog computers.

The most exciting possibility to im- prove learning environments is the combination of the personal computer and the almost-available video-disc system. Such systems have been called "intelligent video-disc systems" by Professor Bork of the University of California at Irvine. In Bork's con- cept, the computer and the video-disc player interact with each other and with the user. In such systems,the computer controls the video disc player, causing it to play a motion sequence, a set of still frames, an audio sequence or causing a computer program stored on the video disc to be loaded into the memory of the computer for subsequent execution. There is a feeling of ex- citement about the intelligent video- disc system, and the potential for dramatic impact on learning. Such systems are some time off (perhaps 3-5 years), but will be worth the wait.

The rate of development of personal computers and related peripherals is breathtaking. It is impossible to guess what new announcements will be made six months from now (or even at the Second Computer Faire), but the consumer and especially the learner gains with every one.

PERSONAL COMPUTERS AND SCIENCE MUSEUMS

Arthur Luehrman
Associate Director
Lawrence Hall of Science
University of California
Berkeley CA 94720
(415)642-4193

ABSTRACT:
    Science Museums have a unique opportunity to educate a broad public about the use of computers. Less than 10% of today's high school graduates have laid hands on a computer keyboard, and even fewer of their parents have done so. Yet the needs, in terms of jobs and personal development, for computer skills is growing rapidly. With 50 million visitors annually and ties to local schools, science museums can play a critical role, and a few have. For many years, the Lawrence Hall of Science has had a vigorous computer education program based on its 100-port time-shared computer,with terminals in about 50 Bay Area schools, on the exhibit floor and in the classrooms at the Hall. Inexpensive personal computers are expected to increase our potential impact tenfold or more in a few years. Among other projects to be described during the panel are these: (1) Putting a dozen computers in a van and driving to schools and clubs to conduct workshops, (2) Plugging into exhibits computers with programs that ask questions about the exhibit or suggest activities, (3) Offering classes in which students will be lent computers to take home and work with, and (4) selling computer and application programs in the museum store.

PANEL: Personal Computers and Learning Environments
TOPIC: Computers for Elementary School Children

Bob Albrecht
P.O. Box 310
Menlo Park, CA 94025

ABSTRACT

They are finally here! The $600 plug-em-in-and-use-em home/school computers are here! Not one, but two $600 computers are available!

> The Commodore PET
> The Tandy Radio Shack TRS 80

Both are complete computers — the $600 price includes computer, memory, alpha numeric keyboard, video (TV) display and cassette recorder.

> Bye, bye paper Tape!
> Hello magnetic tape cassette!

So, back to the classroom ... or the family room. For the next year or two or three I will be visiting elementary schools, helping students and teachers learn how to use, program and enjoy the PET and the TRS 80. You will frequently find me in the school learning center, resource center, or library where the computers will be an open resource, available to all students and teachers. Fifth and sixth grade students will learn to be "resource people" or "teacher aids". They will be available in the center to help others learn how to use the computer.

Right now, there are very few programs available for our use, but people are working on that. There are no instructional materials, so I am developing "teach yourself" style materials to help students, parents and teachers learn to read and understand BASIC. Instead of the usual "math" approach, I am using a verbal and graphics approach.

I will attempt to answer questions about teaching elementary school children how to use, program and enjoy the Commodore PET and the Tandy Radio Shack TRS 80 computers — at school, or at home. Questions mgiht include, but not be limited to the following.

How do I teach my 4th grade child or student how to program in BASIC.

How does an elementary school get started using computers?

How do we overcome teacher inertia?

Where do I get education software? Instructional materials?

What do I like about the PET? Dislike?

What would I change to "improve" the PET?

What would I change to "improve" the TRS 80?

What do I like about the TRS 80? Dislike?

Why aren't there any easy-to-learn computers?

What would you like to have in a home/school computer for kids?

BRINGING COMPUTER AWARENESS TO
THE CLASSROOM

Liza Loop
LO*OP Center, Inc.
P.O.Box 945
Cotati, CA 94928

Many teachers ask, " Why is it important for school children to become aware of computers? " My answer is this:

o The computer is the most significant technological innovation since the printing press. You may prefer to compare it to electricity. Either way, the computer is too important to be ignored by established education.

o Although not all school children will become computer scientists, they will all be consumers. They will have telephones, receive bills, and shop at supermarkets. Therefore, they will all be consumers of computer services. If nothing else, they need to understand that computer service systems are designed by people and can be changed by people.

o Most school children will vote at some time in their lives. In this capacity they will have to make decisions concerning acceptable and abusive uses of computer technology, including uses of computers by the government itself.

o Finally, most people will work during some period in their lives. Studies perdict that over 80% of the jobs available in 1985 will require some involvement with computers or their input and output. Since schools are supposed to address the problem of preparing youth to emerge into the world of adult responsibility, it is hard to justify not teaching about computers.

Most students can develop some sense of what a computer is and what it is not. Fourth graders will avidly play computer games and many will choose to write their own programs rather than accept canned material. Even much younger children rapidly develop an ability to manipulate a terminal provided the output is on their level.

What is Computer Awareness? It is the study of the computer system itself and the place of computers in today's world. It is significantly different from computer assisted instruction (CAI) which uses the computer as a delivery system for a variety of non-computer subjects. Computer Awareness includes units on operating the terminal, using games and other application programs (possibly some CAI), computer applications found in the locality of the school, computer related careers and vocations, introductory programming in high level language (perhaps Basic). On the secondary level it may go on to cover some electronics and Boolean logic. Computer Awareness is not Computer Science, Programming, Data Processing, Electronics, or Computer Maintainance.

Computer Awareness also differs from Computer Literacy. It omits the history of computers and deemphasizes skill in programming. It places more stress on experience with computer applications and less on vocabulary which is often meaningless to younger children and tends to be obsolete very quickly.

Computer Awareness students do write at least one program. However, it may be as simple as a picture program using only PRINT statements. The exercize is intended to develop a sense of the relationship between the user and the programmer rather than a means to promote skill in programming.

LO*OP Center, Inc. has tried a number of different approaches to the subject of computers in education. It ran a demonstration classroom and storefront drop-in computer center for two years. In September, 1977, the storefront portion of LO*OP was closed in order to allow concentration on teaching the Computer Awareness Curriculum on site at various Sonoma County schools and to write a companion text book for the course.

In talking with school staff it became appearent that many teachers wanted a prepackaged curricular module which they could import into their classrooms with a minimum of effort. Others expressed an independent interest in the computer field and shared in the belief that computers will become an integral part of organized education in the near future. LO*OP began to focus on the development of a Computer Awareness Package and on presenting this material in courses for teachers at Sonoma State College.

One constantly annoying obstacle to both LO*OP's storefront operation and to the development of an easily transportable Computer Awareness Curriculum was the lack of reliable, low-cost hardware. The Sonoma County Computer Club provided an on-going evaluation laboratory for microcomputers as one member after another bought and struggled with kit computers which never quite lived up to manufacturers' claims of power and ease of operation. LO*OP borrowed and field tested many of them in classrooms only to spend most of one period searching for a disconnected wire or another watching paper tape load.

The conclusion is that the Computer Awareness Package must include a fully debugged and assembled machine which speaks a high level language and comes up with not more than four or five instructions from the keyboard. Currently under consideration are Radio Shack's TRS80 and the Commodore Pet. Both these machines speak reasonable Basic as soon as you turn the power on and cost about as much as a classroom movie projector.

Within the next year, materials will be available so that every teacher may introduce Computer Awareness into his or her classroom. But, as we are all well aware, this project is only one tiny contribution in the field of computers in education.

Implications of personal computing
for college learning activities

Karl L. Zinn, Research Scientist
Center for Research on
  Learning and Teaching
The University of Michigan
109 East Madison Street
Ann Arbor, MI 48104
(313/763-4410)

The purpose of this article is to provide background information for a session on "personal computers and learning environments" at the Faire, and perhaps more important, to stimulate continuing discussion of this topic among those who see the Proceedings. What is said in the session builds on what is written here, but do not expect it to be the same. And six weeks after the Faire we should expect to have new information, new ideas, and new opportunities for expanded use of personal computing in college learning activities. Still this background information should be useful.

The domain of personal computing I consider more broadly than just small, single-user machines. I wish to include some experiences with timesharing systems in order not to overlook useful ideas about communities of learners and occasions for professional communication within such communities. If we limit discussion to small (inexpensive, portable, individually owned) machines we are not taking advantage of decades of experience with personal computing using costly, fixed systems having the power (processor speed, memory size, instruction set, and complexity) that will be characteristic of the small machine in a few years. I am convinced that the best personal computing is done today on single-user machines, but I haven't given up entirely on timesharing as a means to providing truely personal services. In any case, I want to have communication networks backing up single-user systems in education.

The substance of this background statement is presented in five sections. The first is intended to provide the educator some indication of why the personal computing revolution is so important to computer use (and information handling) in higher education. The second section should inform the computer specialist or enthusiast about kinds of uses in higher education. The third section offers a list of what I see to be needed to help along some needed changes in higher education. The last two sections provide a brief statement about the future and a list of references and suggested readings on computers in higher education.

SIGNIFICANCE OF PERSONAL COMPUTING FOR COLLEGE LEARNING

Many colleges and universities have acquired effective and economic time-sharing systems for use by faculty and students in teaching and learning activities. These can be expanded to accomodate the increasing amounts of use. Also, they can be extended to handle some new kinds of uses such as graphics and information retrieval systems. However, some kinds of computing will be accomplished more economically with small and inexpensive computers for use by one individual or project at a time. At the University of Michigan the Center for Research on Learning and Teaching is exploring various roles for these microprocessors and personal computers within college learning activities.

Aspects of the revolution
The impact of personal computers on education in general and instructional activity in particular will be considerable. Three aspects of the personal computer revolution may help

establish this case: numbers, access, and control.

Personal computers will be used in much larger numbers than instructional computers have been or will be otherwise. The major computer-based instruction facility so far has been the PLATO Computer-based Instruction System at the University of Illinois in which the number of simultaneous users are counted in the hundreds. Changes in the system will expand the number of users to thousands. The most common timesharing systems in use in colleges today are built by Hewlett-Packard and Digital Equipment. Although these are small systems (four to 60 users each), the total number of simultaneous users served by the hundreds of systems throughout the country is counted in the thousands. With continuing sales to colleges this number will expand to the tens of thousands. At least 100 times more personal computers will be used in education activities. The first line set up to mass produce computers was designed to put out 20,000 per month. With three companies now delivering machines, the really large (potential) producers have not yet announced their products. I expect nearly half a million units to be sold in the first year, and in a few years the number used in education will be counted in the millions. When the measures of expense and capability of a technology improve by two or three orders of magnitude the effects are more than just quantitative. Personal computers will bring about qualitative changes in education in the home and perhaps in colleges.

The second characteristic of personal computers is responsiveness. The machine is as available and responds as quickly as a personal and portable electric typewriter or television. But more than that, it responds effectively to more complex directives, rearranging and processing information in a personal way not possible with a shared system designed for the average user. Furthermore, the high rate of transfer of data between memory, processor and display screen opens up new opportunities for real-time animations as well as data analysis.

The third aspect of the revolution I want to emphasize is personal control. The owner of a personal computer determines the uses of one or more personal machines, and shapes their characteristics to personal needs and preferences. Perhaps the most important aspect of control is the intangible consideration of self-determination: the owner can literally wrap his or her arms around the machine, carry it about, paint it purple, and love it as a pet, as well as reprogram it according to personal preferences.

Detailed characteristics of personal computers and their current uses are receiving careful attention by CRLT staff at the University of Michigan. These notes attempt only to list a few characteristics and limitations.

Limitations

In 1977 the limitations of personal computers may have outweighed their observed advantages for use within college learning activities. However, 1978 brings more reliable and powerful models of these inexpensive devices. Indeed, we can expect the capabilities to at least double each year without an increase in production costs for at least the next twenty years.

Reliability has been a major problem with many microcomputer systems to date. Now that personal computers are being mass produced, the integrated systems perform much more reliably than their predecessors which were usually built by hobbiests from kits. However, the construction of peripherals for printing, storage and the like does not measure up to the same standard of reliability. The major problem faced by manufacturers of peripherals is a need to bring out a very low-cost device, something comparable with the low cost of the microcomputers themselves. The electromechanical devices for printing, drawing, or storing information on magnetic tape can not be both cheap and reliable. However, the prospects are good for reliable peripherals using new technical developments which will eliminate entirely the moving parts!

Speed of processing is a consideration. Happily, most instructional applications involve simple programs and the observed response from the personal computer is not distinguishable from that of a timesharing system. However, some applications require many computations (eg, for reducing data or analyzing

text), and the delays may just be too great for a student interacting at the slow speeds of an inexpensive, personal computer.

Personal computers do not now provide inexpensive access to large data bases, and updating the data base is an expensive operation if every user has an individual copy. Therefore, the use of personal computers in records handling, data reduction, and shared files is not advisable with 1977 technology. This could change by the end of 1978, however.

Compatibility among different personal computers imposes some limits on exchange and marketing of programs. Actually this limitation is no more severe than for sharing programs among different computers made by the same vendor. Some of the recommendations given in the third section of this paper will help remove this limitation.

Probably the standards that emerge will depend on certain features of the technology: inexpensive, read-only storage; processors which are less expensive than the medium of curriculum storage; and audio-visual media which incorporate digital logic and storage.

Approaches to applications

Within current limitations of reliability, speed, storage, and compatibility CRLT staff already find many applications for personal computers within the University of Michigan today. Machines becoming available in the first half of 1978 will make valuable additions to the resources for learning. Some machines acquired by individual faculty members are being used in classroom demonstrations. At least three curriculum development projects are using personal computers in connection with laboratories (preparation for regular lab sessions, information processing aids in carrying out the work, and data generators for simulated laboratory activities). Libraries are considering use of microcompters in providing services on campus. Many individual students have purchased machines, and some of these are being used for word processing (preparation of papers), computation, and information organization and retrieval. The School of Music is using a microcomputer system for music analysis and synthesis, and some

electronic music composition.

Some thought about the future of computing in education and society may help college teachers take a useful perspective on computer use by students today. This topic has been written on at length; I will list just three aspects I believe to be very significant for educators considering the general topic of computers in education: computer literacy, home entertainment, and computer-based videodiscs.

A general literacy about computers is likely to be as widespread as knowledge of driving a car or skills for use of a library. Already electronic calculators have become so inexpensive and straightforward to use that nearly anyone can acquire a convenient device to carry in a pocket, purse or wear on wrist. The impact has been primarily one of accomplishing calculations more reliably. As general information processing facilities become as common as the portable home typewriter or television set (indeed, they will no doubt be incorporated within home typewriters and television sets), a general literacy about their use will develop rapidly. To some extent this literacy is already being accomplished in computer courses in intermediate schools throughout the country. Self-instruction at home will take care of the rest. All college students will know how to AND EXPECT TO use computers

This year, microprocessors have become a significant part of the home entertainment industry. Computers are evident in video games which duplicate arcade devices, and in board games which play a fair game of chess or checkers or othello or backgammon. As general-purpose computers become incorporated in home entertainment centers, the capabilities for simulation, modelling and records handling expand rapidly. An important part of the home market will be self-education. Many companies will be selling learning packages which include computer programs to run on the home computer as well as supplementary materials and guidebooks.

New developments in videodiscs may revolutionize personal computers. For some years companies have been working on one version or another of the home videodisc. One of the driving forces behind this movement is the expected

market for old movies and special interest programs in the home. However, in order to solve some of the technical problems of retrieving the bits of information which make up a video signal, the researchers turned to certain digital techniques which are of considerable interest to computer designers. It appears now that including a microprocessor in the videodisc player will make it more practical, reliable, and economical.

Computer control of the videodisc player provides an exciting resource for educational use of microcomputers. A small archive of movie clips (30 minutes total) or a very large archive of color still frames (54,000) or any combination can be searched and controlled by a personal computer. This means one can retrieve, display at regular speed, slow motion or still frame, and move to another section of the file based on instructions from the push-button controls and a history of interaction with the particular learner. Furthermore one can set up the personal computer to interpret the 54,000 frames of video as nearly a trillion bits of digital memory. This reduces considerably the present restrictions on microcomputers for handling large data bases.

These future prospects are quite exciting and probably will bring significant and desired benefits to higher education and to society. A summary of trends concludes this section.

Trends in computing, communication and the locus of education

A number of changes in the technology of computing and communications, and changes in education and society, have increased professional interest in computers in learning and teaching. Four points are listed here: The low cost of microcomputers merit rethinking uses of technology in education. Inexpensive communications may shift the role of centralized educational computing. Incentives will improve for commercial production of materials for training and continuing education. Home computing, combined with other technology, will support a trend toward more education in the home.

The low cost and easy use of microcomputers are providing access to highly interactive computing including graphics for many more people and projects. Video techniques, such as the microcomputer-based videodisc player, are enhancing graphics and low-cost storage for instructional computing (Bork, Luehrmann and Schneider, in press). The economic and social pressures on educational institutions are forcing decisions by state and federal agencies, and by individual institutions, to support additional use of technology.

Inexpensive communications have been promised via satellites and optical fibre telephone lines. Very low cost and very large memories are proposed for educational time sharing systems. When these developments are realized, the cost of centralized educational computing systems will drop significantly. This cost improvement will help restore some of the education market for timesharing taken over by personal computers, but also it will increase the use of personal computers through low-cost support networks.

Commercial incentives are emerging in the U.S., now that packaging is becoming more practical and the educational market is becoming interlinked with other computer uses. For example, recreational uses of personal or home computers include some instruction for the very young learner. And professional uses by the adult practitioner include some new opportunities for learning. New markets for training and continuing education have focused attention on the need for improved instructional design.

People who work on computers in education will have to pay attention to the dramatic development of personal computing. Already considerable education does take place in the home, and more money is spent on home education materials and correspondence in the U.S. than on institutionalized education directly. I am sure a great deal of computer use in education and training in the next decade will be through inexpensive but rather capable machines purchased for personal use in the home and office.

All these current developments can not be covered in the pages of this article or in the corresponding session at the Faire. However, this article can be used together with others in the

Proceedings and the suggested readings to expand your view of microcomputers and uses of personal computing in higher education and to establish means to keep in touch with a diverse and rapidly changing field.

KINDS OF USE

Learning about the computer
Learning about the computer is the most rapidly growing area of computer assisted learning. It's use in CAL follows quite naturally from the introduction of computers into many parts of society and into homes directly. As experience with computers becomes commonplace for a class of learners, educators can use the computer and its procedures as metaphors for other processes and constructs. Education about computers provides tools useful in other learning. The following paragraphs comment on computer literacy, professional development, in-service training of educators, and personal computing.

Computer literacy for all students is established now in some colleges. Computer programming may be required (or assumed and offered only as a non-credit course) so computing skills can be assumed in mathematics, sciences, engineering and some other professions. The extent of computer literacy throughout the U.S. is still very low, but inexpensive computing devices and the popularity of the activity with students will increase the percentage dramatically in the next few years.

Professional development regarding computer use is common for many areas already; accountants, bankers, engineers and others whose professional activities depend on automatic computation and information processing are refreshing skills and obtaining new ones. Additional professions are finding computer assistance indispensible, and information is offered through special courses and institutes for architecture, law, medicine, and others.

In-service training of teachers is essential if public education is to catch up with the rapid change of technology. University institutes offer skills training during the school year and in the summer. Sound information and constructive attitudes regarding

computers in education are essential to success of new programs in public schools.

Personal computing in homes shows the greatest prospect for impact on learning about the computer. By the end of 1978, personal computers will be in use in nearly half a million homes and offices. Many of these devices may be used only with packaged application programs, as are the preprogrammed video games; but all the equipment will be programmable and include convenient local storage for saving user-designed programs.

Learning through the computer
In the past, the core of CAL activity was drill, practice, diagnostic testing, and question and answer tutorials. Today these continue to be the major modes of learning in operational U.S. systems (e.g., the PLATO System of the University of Illinois and Control Data Corporation, Computer Curriculum Corporation systems, Hewlett-Packard and Digital Equipment timesharing systems, and the Univac system). Even though these modes will be overshadowed for a time by a dramatic growth of learning about and learning with the computer, they will remain strong instances of computer application, readily accepted because of their familiarity, moderate cost, and convenience of total systems.

Learning with the computer
The computer as an aid to learning, and an adjunct tool for the learner, is taking on new dimensions now that computing is inexpensive and portable. Hand calculators used in the laboratory, classroom, and study hall have taken on the characteristics of computers: stored programs, program libraries, peripheral storage, printers, graphics, etc. As these least expensive computing devices take on general characteristics, the general-purpose computers are decreasing in price to match the calculators. Practical uses include simulation, gaming, problem solving, and creative activities.

Simulation and gaming have always been popular with teachers and students. Now these entertaining activities are appearing in academically respectable textbooks and laboratory materials.

Problem solving activities once

required knowledge of programming, unless cleverly imbedded in a tutorial sequence which prompted for the parameters and equations or whatever was needed. Now problem-oriented languages, and familiarity with keypress sequences on programmable calculators, put problem solving in reach of any learner experienced in the discipline. Students can be asked to turn to a problem solving facility on the CAL system (or a calculator at hand) to carry out some computation or modelling activity.

Creative activities in education are aided by computers very nicely, at least in experimental systems. Perhaps the most impressive overall is the first approximation to the "dynabook" developed by Alan Kay and his colleagues (1977). Young children are able to sketch, animate, compose music, arrange words in poetic forms, and carry on many other creative activities usually reserved for specialized and advanced users of computers.

Learning support systems

Computer managed learning is expanding within the U.S., albeit quietly and often without any note of the computer role. A significant percentage of schools are using computer systems to aid in classroom management.

Information management goes beyond instructional management to help the student as well as the instructor with information needs. Guidance systems have become quite popular, including SIGI of Educational Testing Service.

The automatic generation of learning and testing materials by computer will soon become a common activity among computer aids. Some tools are already quite popular, especially computer assisted test generation (Lippey, 1974).

Some pointers to current work in the United States have been provided within the text. The next section provides a sample of applications in various areas of instruction.

Areas of instruction

The broad range of computer uses can be shown by selecting some of the less likely uses in six areas of instruction: math, sciences, social sciences, arts and humanities, languages and communications, and the professions.

The few instances given here represent only a small part of all the computer aided learning curriculum materials.

Students in a mathematics course have used a simple computer langauge (LOGO) to generate a mathematical system building from primitive elements.

In a laboratory course in chemistry, preparation for use of titration equipment is aided by conceptual experience provided economically to individual students who use the graphic animation capabilities of the PLATO System before they go into the laboratory.

A simulated laboratory provides research experience for undergraduate students in psychology. The computer is used as a data generator. The value of the simulation depends on the activity of a classroom research community and the effectiveness of the teacher as a consultant.

In the arts and humanities computing serves as a medium of communication as well as a tool for creative work as part of learning. Perhaps the most interesting instructional use in humanities today is as an aid for scholarly work by the student. Graduate students of literature have used preprogrammed applications packages in exercises to determine authorship or analyze style. Undergraduate students explore rules of language through computer generation of poems and stories.

In language learning, aid in practice of skills is the dominant computer use. Contrary to the idea that the cost of development needs to be distributed over many students, two professors at Stanford University are programming computer assistance for a dozen specialized courses which have such low attendance that the department of Slavic languages can not afford to staff them. With this assistance (tutoring, drills, and practice exercises) the professors plan to handle a larger number of students and in a greater variety of courses than previously possible.

Preparation for professional work accomodates as much computer use in training and education as anywhere. For example, management games are very popular in business education and natural resources; simulated cases are used in law and medicine; and design

exercises depending on computer assistance are common in engineering and architecture. One of the most innovative applications is computer assistance for advanced seminars which bring together graduate students from different departments for study of a problem area (e.g., energy conservation, regional planning, or technology assessment). Each participant uses computer assistance for organizing information from diverse and sometimes unfamiliar areas, communicating with others in the seminar between face-to-face meetings, and drafting working papers for review by the group. The organizers of the seminar keep the group focused on the problem without minimizing important background material, and call on resource people who might not otherwise have time to participate except for the convenience offered by computer-assisted conferencing. This enables them to respond at any time of day, any day of the week, and from any user terminal which can connect to the computer or network handling the conference.

NEEDED DEVELOPMENTS

Needs and issues

Further development of computer assisted learning in the U.S. will interact with a number of needs and issues related to the technology as well as to educational applications. Three such issues are listed here.

Microelectronics technology, with rapidly decreasing costs which depend on greatly expanded usage, is forcing producers to find new markets for computers and related technology in education as well as throughout society. Appropriate uses in education require planning for and managing the design of the technological aids and their introduction into educational activities and institutions. Personal computing is being marketed strongly in the U.S. and will be taken up in the next few years by many people for entertainment and small business purposes. The equipment in which industry is investing large sums of money for the personal computer market will not serve educational purposes well without attention to considerations of design specifically for educational purposes.

Higher education faces serious problems of financing, access, credibility, and the like. Certain of these difficulties can in part be eased by appropriate uses of technology, if resources are available at the right time and place for research, development, evaluation, demonstration, diffusion, and operation (Levien, 1972). Otherwise, significant opportunities to aid all learners, and in particular the disadvantaged, the handicapped, the gifted, and the isolated learners, will be lost.

General literacy in computing and information systems will be required by all, consumers as well as marketers, employees as well as managers, and learners as well as scholars, if society is not to be disrupted by a revolution encouraged by rapid growth of technology needed to support today's "information society." Computer literacy can begin in elementary school, and sooner. However, because of the rapid introduction of the technology in many areas, computer literacy training must be carried on in colleges, professional schools, certification programs, on-the-job training, community education, and public media reaching the homes.

Of course, other problems and issues may be at least as important as the three listed above. However, this summary statement does express the nature of the current situation for computer assisted learning in higher education.

Areas for possible action

One major need at the national level is for coordination of planning and funding. Many important matters, e.g., goals, standards, and credibility, can be accomplished only through national discussion and action. The funding necessary to meet the needs in this area can no longer be handled piecemeal through many different agencies. The commitment to excellence in education and to effective use of technology must come from the top. Information systems in education and society cut across many areas: research, development, handicapped, gifted, elementary, and professional. Useful information and good advice on these matters has been accumulating for over 15 years in the form of recommendations of national

commissions (National Academy of Sciences, 1966, PSAC, 1967, Commission on Instructional Technology, 1970, Carnegie Commission, 1972), professional organizations (e.g., CBMS, 1972) and review projects (Zinn, 1970, Anastasio and Morgan, 1972, Hamblen, 1972, Levien, 1972, Mosmann, 1976). A new commission or conference can begin with the recommendations of earlier efforts, review the present state of technology and institutions, and take thoughtful action. Action is necessary now in response to the pressures and problems; furthermore, benefits are more easily justified today in light of the dramatically improved economics for applications of microelectronics and telecommunications.

Designated centers with good support could provide sites of excellent training, development and research (Carnegie Commission, 1972). Potential users urgently need the most current information, the best training, and an optimal environment for development activity. Residencies would provide an opportunity for individuals to get away from regular responsibilities to initiate new work. Research opportunities would be increased significantly by bringing together creative individuals with necessary resources and a variety of learning environments. Large-scale exploration of technological opportunities and basic concepts of learning would become practical. Alternative CAL systems and approaches to curriculum could be compared within the same environment.

Immediate action to give educational uses for computers an identity different from data processing would in some organizations facilitate obtaining equipment necessary for meeting institutional goals at lesser costs. Instances include public education as well as military training, and state and local support as well as federal.

Immediate action to recognize computing and information processing as a significant part of basic education would set in motion the process of curricular revision necessary to the information age. In a few years all students in public schools would become familiar with computers, programming as well as applications, by about the eighth year of school. The college teacher could then assume long

familiarity with computers for entering students.

Curriculum development requires special attention, since computer use in education is a new industry, as yet untested and lacking incentives for developers and distributors. Commercial publishers can not be expected to initiate high risk ventures, and yet they may be left behind if the computer vendors try to provide curriculum materials. Universities and colleges have much to contribute since most textbooks originate there today. Individual authors need to see rewards, both academic and economic, for their efforts.

The social implications of dramatic changes in availability of information and automatic processing require attention. Every elaborate clipboard or binder will have a small pocket for a specialized calculator. Every reference book or procedures guide will have imbedded within its cover an information processor suited to the subject. Each desk encyclopedia will include sounds and animations and a microprocessor which conducts searches of the entire text as well as selects from the contents, index and cross references. Planners need to consider the implications of new modes of representation and communication with machines, new skills for learning, problem solving and creative activities, new roles for teachers and managers, new situations for learning at home, on the job, and in the community. Administrators need to plan systems so that increased dependency on information machines, for assessment of ideas as well as retrieval of information, will not become an inappropriate crutch.

Communication between people and machines needs careful attention. As long as the students (or other casual users) need to type on a keyboard and watch for text and numbers and simple diagrams to appear on a special screen, these machines will have a rather narrow application in training and education. However, when a user can talk to the machine and get a response not just in printed text but in spoken words and other sounds, and can see the effects of his or her directives in the actions of equipment such as models and tools, then the computer will fit in to a much larger world of learning.

Additional areas for possible action at least as important as the eight listed above can be added. For example, through computers special opportunities become available for the gifted student. Dramatic improvements in communication and learning are achieved for the handicapped: Kurzweil's reading machine and Telsensory's talking calculator and Braile output device for the blind, and similar specialized equipment for the deaf.

Other aspects of the technology provide facility for producing speech, processing knowledge, and building personal skills for learning and performance.

FUTURE

New modes of representation and communication

Future developments will extend the modes of communication possible between the learner and the machines. Speech and other sounds will become suitable for entry into the machine as well as for output from it. Gesture and other motions will be interpreted usefully. These developments will bring immediate benefits for those lacking some standard sense or accepted communication means. For example, computer-assisted communication will revolutionize Braile for the sightless, and provide random access to audio and other personal notes. Already an application of microcomputers provides speech to those who have lost it through accident or cerebral palsey. Physiological measures will be encorporated as input, opening up new channels for persons lacking the motor control necessary to operate typewriters or to speak.

Information will be represented with improved graphic and audio means using networks and other data structures. The exploration of knowledge will be more directly available through manipulation of structure, organization, and dynamic interactions by the learner. The student will, with assistance of information processing routines, work effectively within a personalized and dynamic information base. This development will depend on considerable literacy in infomation handing using computers.

New skills of learning, problem solving and creative production

Computer assistance will help learners arrange multiple views of text and graphics. Skills of speed reading will be extended by aids for immediate comparisons and cross references among sections of text. Facility with graphics will extend far beyond the multi-media shows of today. Authors of textbooks and reference materials face new challenges in applying the technology and anticipating improved skills of users.

Problem solving skills will expand in more directions than can be anticipated. Induction may be facilitated by interactive, computer-assisted deduction. Proof of the four-color map problem by students with computer assistance is only a beginning. More creative solutions to engineering problems will be especially effective.

Artistic creations will similiarly be extended beyond our present abilities to comprehend and appreciate. For a primitive example, consider today's dynamic sculpture for which a sound-light score interacts with the movements and speech of its observers. Young artists will find many new opportunities, and the world of creative art will be opened to those previously excluded by physical handicaps.

Previously untapped capacities for learning and performance will be put to use. A computer-based lab in which learners explore their own abilities may help in areas such as inter-hemispheric communication ("using both sides of the brain") and enhanced mechanisms of recall and pattern recognition. Unanticipated and dramatic benefits may follow from the development of synergism of the human user and various machine information systems. Some of the most significant benefits may be obtained with the enhancement of communication within communities of all sizes.

The United States does not have the only innovative projects working in these areas, as is apparent when reading other articles reporting on developments elsewhere in the world. Extending communication, processing knowledge and building personal and interpersonal skills are important areas for future development anywhere in the world of personal computing and learning today.

SUGGESTED READINGS ON COMPUTERS IN EDUCATION

Anastasio, Ernest J. and Morgan, Judith S. (1972) Factors Inhibiting the Use of Computers in Instruction. Educom, Princeton, N.J., USA.

Blum, Ronald (Ed). (1971) Computers in Undergraduate Science Education. Conference Proceedings, Commission on College Physics, College Park, Maryland, USA.

Bork, Alfred M., Luehrmann, Arthur W., and Schneider, Edward. (in press) Report of a Conference on Intelligent Videodiscs. University of California, Irvine, USA.

Brigham, Christopher R., Kamp, Martin and Cross, Kenneth J. (1975) Index of CAI Medical Education, Lister Hill Center, National Library of Medicine, Bethesda, Maryland, USA.

Carnegie Commission on Higher Education. (1972) The Fourth Revolution: Instructional Technology in Higher Education. McGraw-Hill, New York, USA.

Commission on Instructional Technology. (1970) To Improve Learning: An Evaluation of Instructional Technology. R. R. Bowker Co., New York, USA..

CONDUIT. (1973) Documentation Guidelines. CONDUIT, P.O. Box 388, Iowa City, Iowa, USA.

CONDUIT. (1977) Pipeline. CONDUIT, P.O. Box 388, Iowa City, Iowa, USA.

CCUC (1977) Proceedings of the 1977 Conference on Computers in the Undergraduate Curricula. Michigan State Unniversity. CCUC, 124B Linquist Center, Iowa City, Iowa, USA.

Hamblen, John. (1972) Inventory of Computers in U.S. Higher Education, 1969-70, U.S. Government Printing Office, Washington, D.C., USA.

Harrison, Shelly A. and Lawrence M. Stolurow (Eds). (1974) Educational Technologies: Productivity in Higher Education. State University of New York, Stony Brook, New York, USA.

Hunter, Beverly, Kastner, Carol S., Rubin, Martin L., and Seidel, Robert J. (1975) Learning Alternatives in U.S. Education: Where Student and Computer Meet, Educational Technology Publications. Englewood Cliffs, New Jersey, USA.

Kamp, Martin and Brigham, Christopher (Eds.) (1973) special issue of Computers in Biology and Medicine.

Kay, Allen, and Adele Goldberg. (1977) Personal Dynamic Media. Computer. 10, 3, 31-41.

Levien, Roger E. (1972) The Emerging Technology: Instructional Uses of the Computer in Higher Education, A Carnegie Commission on Higher Education and Rand Corporation Study, McGraw-Hill Book Company, New York, NY, USA.

Lippey, G. (Ed) (1974) Computer-assisted Test Construction. Educational Technology Publications, Englewood Cliffs, NJ, USA.

Luehrmann, Arthur W. (1972) Should the Computer Teach the Student or Vice Versa? Proceedings of the Spring Joint Computer Conference. AFIPS Press, Montvale, New Jersey, USA.

Main, Dana B. (1976) Experiment Simulation (EXPER SIM): The Development of a Future-Oriented Pedagogy, in D. Bailey (Ed), Computer Science in the Behavioral and Social Sciences. University of Colorado, Boulder, Colorado, USA.

Norton, A. Kent and Luehrmann, Arthur
W. Jr. (1975) Project COMPUTe: A
Mechanism for Producing and Distributing
Instructional Material. 993-997 in
Computers in Education, O. Lecarme and
R. Lewis, Eds. North-Holland,
Amsterdam.

Mosmann, Charles. (1976) Evaluating
Instructional Computing: Measuring Needs
and Resources for Computing in Higher
Education. The University of
California, Irvine, USA.

National Academy of Sciences. (1966)
Digital Computer Needs in Universities
and Colleges, A Report of the Committee
on Uses of Computers. National Research
Council, Washington, D.C., USA.

Nelson, Ted. The Home Computer
Revolution. Published by the author.
Distributed by The Distributors, 702 S.
Michigan, South Bend, IN 46618.

Nevison, John M. (1976) Computing in
the Liberal Arts College. Science.
194, 396-402, October.

Nievergelt, Jurg. (1975) Interactive
Systems for Education: The New Look of
CAI. Proceedings of the IFIP 2nd World
Conference on Computer Education.
Amsterdam: North-Holland.

PSAC. (1967) Computers in Higher
Education. President's Science Advisory
Committee, U.S. Government Printing
Office, Washington, D.C., USA.

Seidel, Robert (Ed). (1975)
Proceedings of the Ten-Year Forecast for
Computers and Communications. Human
Resources Research Organization,
Alexandria, Virginia, USA.

Seidel, Robert J. and Hunter, Beverly
C. (co-inventigators) (1977) Academic
Computing Directory: A Search for
Exemplary Institutions Using Computers
for Learning and Teaching. (First
Edition) Human Resources Research
Organization, Alexandria, Virginia, USA.

Suppes, Patrick. (1975) Impact of
Computers on Curriculum in the Schools
and Universities, in O. Lecarme and R.
Lewis (Eds.), IFIP Second World
Conference on Computers in Education,
Part 1. North-Holland Publishing
Company, 173-179.

Suppes, Patrick. Smith, Robert and
Beard, Marian. (1977) University-level
Computer-assisted Instruction at
Stanford: 1975. Instructional Science.
6, 151-185.

Van Dam, Andries. (1976) Computers in
Teaching: An Application of Hypertext.
Final Report to National Endowment for
the Humanities, Brown University,
Providence, Rhode Island, USA.

White, James. (1977) Your Home
Computer, DYMAX,
Menlo Park, California, USA.

Zinn, Karl L. (1970) An Evaluative
Review of Uses of Computers in
Instruction (Project CLUE: Computer
Learning under Evaluation). Final
Report U.S.O.E. contract no.
OEC-5-9-32-509-0032, University of
Michigan, Ann Arbor, Michigan, USA.

Zinn, Karl L. (1977a) Free and
Inexpensive Materials on Computing in
Teaching and Learning Activities: An
Informal Appraisal. Center for Research
on Learning and Teaching, University of
Michigan, Ann Abor, Michigan, USA.

Zinn, Karl L. (1977b) Computer
Facilitation of Communication within
Professional Communities. Behavioral
Research Methods and Instrumentation, 9,
2, 96-107.

Zinn, Karl L. Computer Assisted
Learning in the United States. In a
special issue of the British Journal on
Programming and Educational Technology,
in press for spring 78.

GETTING IT RIGHT: NEW ROLES FOR COMPUTERS IN EDUCATION

Thomas A. Dwyer
University of Pittsburgh
Pittsburgh, PA 15260

Introduction

Computers are powerful tools.
Really good ideas on how to use
them are much scarcer than one
would imagine, however. This is
partly because (up to now) com-
puters have been very expensive,
and controlled by institutions that
couldn't afford to depart very far
from traditional ways of doing
things. In education this has
meant trying to use computers
mostly as "teachers" (CAI). When
all the costs of pursuing this
unimaginative idea are totalled up,
and the results are compared to what
what was promised, the scorecard
isn't a very good one.

Is there a better way? If so,
what is it, why will it be better,
and is there assurance that this
time we can "get it right"? This
paper argues that the answers are
to be found in personal computers,
and that the right way to use them
is staring us in the face. To see
why, let's examine a remarkable
learning experience many readers
have had.

The Secret Unveiled

Imagine you have just arrived
at an airport in a strange city,
and now need to reach your final
destination by way of unfamiliar
roads. One possibility is to take
a taxi. This option is direct and
efficient, and it can have the
bonus of being a personalized tour
along a tried and proven route. It
has all the potential for being a
first-rate educational experience.
Yet, it is not likely that upon
arrival you could pass a test asking
for an accurate description of the
route just taken. Your "individu-
alized" treatment will have gotten

you to your destination, but you'll
still be a stranger to the territory.
The ingredients for a true adventure
were missing.

Consider now another option.
Suppose that you rent a car, and
drive yourself. The simple act of
moving into the driver's seat will
have a profound effect upon the hun-
dreds of interactions about to take
place as you move into the role of
problem-solver, becoming an adven-
turesome and necessarily creative
learner. This option comes at a
price, of course. There will be the
need to find and negotiate for a car,
ask directions, study a map, choose
between alternatives. There are also
likely to be mistakes -- "ineffi-
ciencies" by some standards. Land-
marks missed, or instructions mis-
understood will mean backtracking
and revised planning. Questions will
have to be asked, time will be lost.
And the cost of having exclusive use
of a car will be higher. But in the
end, the person who goes "solo" will
have learned things about getting
from A to B that are accessible in no
other way.

The paradox we see from this ill-
ustration is this: the guidance of
others may very well inhibit the best
kinds of human learning. The con-
clusion it suggests is that people
have far more instrinsic talent for
the business of learning than they
have for the business of describing
it, or bringing it about in others.
Even the ability of a small child to
learn to deal with the incredible
complexities of a constantly changing
world makes our ability to explain
how it all works seem pale by com-
parison.

Even less impressive is our
success in promoting human learning

through institutions organized expressly for that purpose. Placing students within the walls of carefully designed schools would certainly appear to be the best way of assuring that they get transported along paths (called curricula) that visit many important educational points. The predicament faced is that for most students these are only visits, and dimly remembered ones at that. They never get to really know the territory.

Enter the Computer

What has all this to do with computers? The answer is both simple and perplexing. In reviewing the history of computers in education one finds that the majority of effort (and money) has gone to promoting their use as expensive educational "taxi-cabs." This effort has gone under the name of Computer Assisted Instruction, or simply CAI.

In CAI, driver and passenger are viewed as separable entities, with the "driver" turning out to be an impersonal conglomerate of lesson designers, computer programmers, and hardware vendors. Students are relegated to the role of individual passengers. There is, of course the promise that they will be the beneficiaries of personal, customized transportation to the most exotic of educational worlds. In reality, the promised tour is at times more like a shuttle, with repetitious visits to the unimaginative lands of drill and practice. The alarm raised in Anthony Oettinger's 1969 book Run, Computer Run, with its conclusion that "every effort to introduce technological change into education has revealed how profoundly ignorant we still are," was not without basis.

Enter the Personal Computer

But there is also good news, and it's getting better all the time. It comes partly in the form of a new approach to technology, and partly as a growing body of examples of student accomplishment that indicate an exceptional idea is at hand. It's a flowering of the "drive-yourself" option (called "solo-mode computing") in schools, made possible by personal computers.

The new spirit is all the more remarkable in that it pretty well made it on its own. The funneling of large funds into CAI made early solo-mode computing an uphill battle for all but the most determined and adventuresome. It is to the credit of the ingenuity of exceptional teachers and their students, that the door to solo-mode computing was gradually inched open. What first came through the crevice was at times amateurish, at times quite professional. But it was always inspirational and exciting, and it caught on.

And now a new force from a completely unexpected quarter promises to swing the door wide open. It's in the form of a personal computing movement that looks upon computing as a personally desired and appreciated enterprise. It's a use of technology that is (to use the words coined by Ivan Illich in his book De-Schooling Society) "convivial" rather than "manipulative" vis-a-vis the aspirations of people. It's the "drive-yourself" option for computing come true.

The personal computing movement is important because it can bring a new and different use of technology into our schools without major expenditures. If it finds a mixed welcome at first, there need not be concern. This is because it is a use of technology that can literally exist in a home, a storefront, a community center, the corner of a library or museum.

Actually there is hope that personal computing will find a welcome in schools, because it is in the hands of the people who use schools, pay for them, and care about them. It will thus be the first application of technology to education that need not sell itself as "new and revolutionary." It guarantees nothing. Yet it assures everything, because it's in the right hands, free spirits who know the value of learning on one's own

THE ROLE OF THE MICROCOMPUTER IN A PUBLIC SCHOOL DISTRICT

Peter S. Grimes
Curriculum Supervisor
San Jose Unified School District
1605 Park Avenue
San Jose, CA 95126
(408) 998-6124

The purpose of this paper is to apprise the reader of how San Jose Unified School District is utilizing personal and home type microcomputer systems in its instructional program. San Jose Unified is considered to be somewhat of a bellweather in the educational use of microcomputers because it established a district policy for the rapid introduction of microcomputers into its secondary schools over two years ago when few educators knew the possibility even existed. The paper addresses the following topics:

- A general description of S.J.U.S.D.'s instructional use of microcomputers.
- A brief rationale for the development of S.J.U.S.D.'s microcomputer policy.
- An explanation of why we promoted instructional microcomputing before identifying a curriculum.
- A more detailed look at how we are using microcomputers.
- A brief exposition of some of the things we have learned.

San Jose Unified School District is a large (38,000 ADA) urban school district nestled in the center of the great Santa Clara Valley at the southern end of San Francisco Bay. We have seven high schools, seven junior high schools, thirty-seven elementary schools, and a regional vocational center.

Our educational computing facilities include ten time-share terminals (PDP 8/E), fifteen microcomputers (one SOL, three POLY 88, one CROMEMCO Z2-D, eight IMSAI 8080 and one PET), and a Hewlett Packard 2000 system at our vocational center. (Since the program at the vocational center is dedicated almost exclusively to vocational data processing, the following remarks refer to our microcomputers and the PDP 8/E time-share mini.) These terminals are generally distributed as follows: one or two terminals in each junior high, two or three terminals in each senior high and one roving terminal for our elementary extended learning program.

Our use of these terminals is presently organized around the following themes. For junior high school, the emphasis is computer literacy for a large number of students, with some schools offering BASIC programming as an elective. The senior high school objective is computer programming using a variety of machines: programmable pocket calculators, desk top printing calculators (with optical card readers) and general purpose microcomputers. Our language of choice is BASIC with the probable addition of disk FORTRAN in the near future. We also anticipate the development of an introductory computer science elective as the need develops.

We are told that San Jose Unified is somewhat unique in its district wide policy to encourage instructional computing through the purchase of small microcomputer systems. The dramatic thrust of this policy is evidenced by our purchase of fifteen such units in the past two years and a probable similar expansion over the next several years. As a school district, we are quite serious about promoting general purpose computing through small microprocessor based systems.

I believe it is important to reflect upon the origin of this policy. Well over two years ago, it became quite apparent to us (residing as we do in the heartland of the digital electronics industry) that small general purpose computing systems would become relatively inexpensive and thus well within the budget capabilities of individual schools and school systems. (This judgement has been thoroughly vindicated with the marketing of the $595 PET in late 1977.) This belief led us inexorably to certain conclusions:

1) Inexpensive computing would rapidly lead to a demand for the inclusion of computer programming and computer science into the public school curriculum.
2) The ordinary citizen would very soon have to have some knowledge and skill in the use of many types of computing devices; i.e., pocket calculators,

programmable calculators and computers.

3) Small general purpose computers would rapidly become important as instructional devices (learning games, simulations, tutorials, drill and practice).

4) Profound changes in the mathematics curriculum would become inevitable; i.e., increased stress on decimal notation, flow charting, algorithmic programming, iterative and recursive techniques, computer evaluation of functions, the demise of the slide rule, and vastly increased use of mathematical processes heretofore requiring calculations far too complex and time consuming for common use.

In our mind, the digital electronics revolution had a certain inevitableness about it. These things would happen! They would happen quickly! The sooner we became involved, the more control we would have over the situation! We installed our first microcomputer (an IMSAI 8080) in March of 1976.

San Jose Unified has become somewhat of a bellweather for schools and districts contemplating their first entry into educational microcomputing. We have received many inquiries about what we are doing with our microcomputers. These inquiries have a common element. What is your curriculum? What was your planning? What do students do with the machines? There seems to be a feeling that to justify the investment, the use of the machines must be totally maximized within the shortest period of time. It is our opinion that this seeming urge to document the need for computer resources and specify a curriculum is premature at this early stage. That is why we do not as yet have an identified curriculum. Our belief in the inevitability of educational computing does not imply that we have gazed into the crystal ball and have thereby seen the future. We really don't know exactly how these microcomputers will ultimately be used in our schools. Nor does anyone else! All we know is that we must teach more about computers to more and more students and that we need computers to discover how to do this. We think that teaching BASIC programming is a relatively safe thing to do at the present time.

Why do we say that detailed early planning is premature? Because _very_ _few_ of us in secondary public education have any real experience with computing, even mathematics teachers. We have gone back to school ourselves! We are learning BASIC and FORTRAN, assembly programming, and something about computer science in general. We are finding out more specifically how our mathematics courses will probably be

impacted by the advent of inexpensive calculators and computers. We are also finding out what the present student demand is for computer electives so that we will have a basis for projecting the future. We are trying to be creative. We are fearful of making detailed plans before we are knowledgeable enough to do so. We feel very strongly that teachers should develop the curriculum and that they would not be motivated to do so without the actual physical presence of a computer. (Teachers, too, can be very pragmatic. It doesn't make very much sense to make a large investment of after work time and effort unless the resulting skills and knowledge can be put to use.) What we have done, then, in San Jose Unified is to provide the computer resources so that teachers could make the necessary discoveries and obtain the necessary background to begin identifying and meeting the needs created by the computer revolution. This development takes time. That is why we became involved so early. And so...what are we doing and what have we learned?

We have introduced programming courses in high school. These vary from one to three semesters. A typical sequence would be, a) introduction to computing using programmable hand held calculators and printing desk top units, b) BASIC programming with microcomputers, and c) advanced BASIC programming. In junior high school we have taken three approaches. First, a few schools are offering a one semester BASIC programming elective for grades 7-8-9. Second, after school computer clubs have been organized in which BASIC and system programming is taught in a more recreational context. Third, we are developing a plan for computer literacy where most students prepare a program, enter it into the computer and successfully run it. At the elementary level we are testing the waters with one roving microcomputer serving our grade 4-5-6 extended learning program (MGM) students. So far these students have been exposed to a variety of learning games, simulations and drill and practice routines. We have also taught the more motivated elementary student some rudiments of BASIC.

We have also learned a good deal. There is definitely a need for computer instruction. Five of our high schools are offering one or more semesters of BASIC programming. One high school has employed a computer science teacher and anticipates that he will be teaching full time in this area within the year. Although we have found no suitable texts, a large amount of reference material exists which has made it fairly simple for teachers to assemble a body of graded programming

experiences (mostly in mathematics). Up to 300 high school students are currently involved, a number likely to grow as our computer resources expand and as computer programming becomes more and more recommended or required for college matriculation.

Our experience in junior high has been quite similar. Although there is not as great a demand for programming as an elective, the computer clubs have been extremely popular and wide scale computer literacy is being developed as a goal for our grade 7-8-9 mathematics program. Our student contact here is much more difficult to measure because junior high computer instruction tends to be much more infrequent and diffuse. Still, we would estimate that the district impact here exceeds five hundred students and is rapidly growing.

The results of the elementary program are difficult to assess. Only a few students have profited from their exposure to BASIC. This is understandable, however, considering the lack of computer training or knowledge possessed by most elementary teachers. It would appear that the computer's future in the elementary grades is as a learning device in the areas of gaming, simulation, drill and practice and tutorial instruction. The elementary program has been more successful where older students have been available to serve as cross age tutors.

With regard to hardware, we have found our microcomputers to be thoroughly dependable. Time between failure seems to be in excess of one year. Most of our machines are running on 16K of RAM with 8 or 10K BASIC interpreters. Our system storage medium is audio cassette tape. This has proven to be quite satisfactory. Students store their programs on paper tape or in cassette cartridges. We are experimenting with teletype vs. television monitor I/O and have reached no conclusions. We are finding our Cromemco Z2-D machine to be the most versatile, by far, of all our microcomputers. Students have their own mini-disk for program storage. Cromemco software (BASIC, FORTRAN, Z-80 ASSEMBLER, DOS) is superior to anything we have seen. Also---the Cromemco machine can expand to eight BASIC users at a very modest cost for each additional user. Given the exceedingly low failure rate after installation, we expect to pursue this Cromemco timesharing approach in our high schools. (The argument against timesharing is that when the machine dies, so do all the terminals.) We have also had a good experience with our one and only PET. It performs as advertised and seems to be just as reliable as our other machines. Because of its attractive cost ($595 for 4K user RAM and $795 for 8K) we expect that we will purchase many more PETs over the next year or two. Since they are less versatile machines, they will probably find their greatest use in junior high where computer literacy is the primary goal.

There have been some problems, of course. We have found that a system should be thoroughly burned in and tested before installation. With microcomputers, the first few weeks of life are the hardest. We are also very much aware of the lack of good software interchange. BASIC has yet to be standardized and, just as important, there is no standard medium of exchange.

In conclusion, we would like to emphasize that what we are doing is very "plain Jane". We are not engaged in anything at all exotic. We are mostly teaching BASIC. The context of our programming is largely mathematics because mathematics teachers have preempted the field. We like BASIC because it is interactive and because it is so well suited to the range of abilities with which we have to deal. We know about CAI, but are not emphasizing its use at this time because of the almost total absence of microcomputer adaptable software and the high cost of preparing our own CAI software. More importantly, we are following the approach I have outlined above so that we can quickly adapt to the new demands being placed on the mathematics curriculum. Perhaps of most importance is our intense feeling that programming and computer science is a new curriculum area, related to, but somewhat separate from mathematics, science and electronics. The digital electronics has birthed a new secondary school subject and we want to be part of its upbringing.

MICRO COMPUTERS IN A HIGH SCHOOL - EXPANDING OUR AUDIENCE

William J. Wagner[1]
Mountain View High School
Mountain View, California

Introduction

The title of this talk is meant to be ambiguous, for I will argue that two audiences are being expanded: the introduction of micro computers into the high school will allow computer education to be offered to more students, and further, educators like me represent a very different set of users of the formerly hobbyist-oriented micro computer. So if you think my feet are shaking up here, it is merely because I am playing the role of the tip of the iceberg.

Computer programs in schools are not new. What is new is that micro computers open the possibility of bringing computer education to many more students within the strictures of modern school budgets.

This broadened market represents a real challenge for computer educators. In every school there exist some students (about 1%, I think) who will, if you let them, spend all their waking hours at the terminal, making the machine jump through hoops, and occasionally emitting those familiar gales of laughter. They are great people (they are us, right?). Every teacher needs these kids around - they keep the adrenalin circulating. We are proud of their programs and like to take credit for their progress. Indeed, we should take some credit, but not for teaching them much. Our contributions are in suggesting interesting new kinds of problems and in getting the equipment in their hands in the first place.

Important as it is to give these future professionals and/or hackers their first boost, our programs should not be judged or justified according to the progress of these superstars. Schools are full of lots of different people, and it is this other 99% which I intend to discuss here, and to which the program at Mountain View High School is directed. Also, although I am by no means a computer hobbyist and have only limited knowledge of the micro computer field, I will

----

[1] Home Address: 127 O'Connor St., Menlo Park, CA 94025

present my views on the special requirements of high school computer facilities, and why we selected micros.

Throughout this talk I will also ask for your help. Computer educators need better ways to find each other and share their ideas, plans, successes, and even failures. I hope this will be the beginning of a useful exchange among us.

The Students

Without risking more guesses about percent composition, here are five categories of students, for the purpose of discussion:

A. The computer hotshots. These have been discussed above, but I should add that they are not necessarily the best students in other classes, and may even have trouble getting computer assignments in on time. We can really help them by focusing their energy and enthusiasm toward activities which will continue their growth.

B. The good students in math and science. These are our natural audience, but sometimes it seems surprising that they do not flock to the computer. For one thing, they are very busy with course work, and school courses are not often set up to reveal how the computer might be of use. And the tightly packed college prep schedule has little room for electives.

C. Other good students. This may be the largest group in any school. They also are busy with various school activities, but unlike group B, computers are not on their list of things to pursue some day. Further, they often show a remarkable disinterest in what we like best - computer games and programs which crunch numbers.

D. Kids with lots of interests, but not particularly top students. Some of these might be interested in our hardware, but not so much in how to manipulate it. Others find programming to be their first interesting

"academic" class. Often their math level is
so low, however, that the number of problems
that can be suggested is limited.
L. Unmotivated and unsuccessful students.
This is another large group. They find their
way into computer classes because their
schedule is not full and because someone,
usually not the student himself, thinks that
something flashy and new may produce a change.
It turns out to be true in some instances.

        I will mention in passing another
group which has remained an enigma to me -
young women. At my school they make up at
least half of the enrollment in advanced math
classes, but seem impossible to attract to
the computer program. Those in the
programming classes are every bit as good as
the boys, but only one or two out of the
twenty or so students who spend extra time
programming or on games is female. I look
forward to discussing this situation with
persons interested in sex differences in math
and science.

The Importance of Expanding Our Audience

        I think that we designers of computer
curricula have tended to think in terms of
groups A and B only. It is very difficult to
find a programming book that does not assume
knowledge and interest in higher math. Also,
we have been content with the few students
from those groups who flocked around us
because we couldn't have served many more
with our limited facilities anyway.
        It is important for us to reach out to
this large majority of students who do not
naturally find their way to the computer room.
It is obvious that computers are actually a
part of their lives whether or not they like
or understand them. However, few
opportunities will exist for them to interact
actively with computers outside school.
        Furthermore, these students tend to be
alienated from the computer activities at
school - they think it is for smart people
only, and they think that demented laughter
and secret language means you have to be a
little crazy too. (A colleague at a nearby
school who brought her Algebra I class to the
computer room for a week of introductory BASIC
told me a wonderfully poignant story that fits
here. As a 10th grade girl sat down at the
terminal for the first time, she looked around
nervously and inquired, "Can anyone see me in
here?")

        How can we reach these individuals who
do not automatically think of computers as
their thing? It is a difficult teaching and
strategic problem, but we must continue to

look for ways, for several reasons. First,
this could be their last chance in a
relatively non-threatening environment to have
hands on experience with a computer. Once they
leave high school their education and
experiences begin to narrow toward a
vocational or academic or lifestyle choice
which will preclude taking a flyer into an
apparently unrelated field like programming.
Their contacts with computers will be passive,
inspiring awe, fear, or anger.
        Also, the opportunities that do exist
out there are more restricting. The college
computer classes I know about are not intended
for the liberal arts person, but for the
future professional or technician. Thus we
must think of some of our teaching as part of
a general education. And for those students
too busy, nervous, or skeptical to commit
themselves to a full course, we must find ways
to bring computers into the classes they do
select.

        Another reason to teach these students
about computers is that we can offer valuable
experience in problem solving and thinking, as
well as a new vocational direction. This area
matches so many of the goals expressed by
various departments of my school and of the
school itself, that I am sure that if the
slate were wiped clean and a new public school
designed, courses about programming and
computers would merit permanent status in the
school.
        In this utopian school computer
instruction of some kind would be a natural
part of most students' coursework: either a
regular course like Geometry, the first step
onto the college bound math track, or the
equivalent of an elective in English or Social
Studies, or a vocational offering like
Woodshop or Auto Mechanics. I have found that
experience with computers and programming can
help teach math more effectively, can help
students problem solve and organize their
thoughts, and can of course lead to jobs at
various levels of academic preparation.
        Revolutionary school reform is not
imminent, so it is the responsibility of us
educators to work for the gradual increase in
use of computers in schools so that these
benefits can be made available to more
students and apparent to more decision-makers.

        Finally, I think that almost anyone in
high school who is willing can be taught
programming in BASIC at some level, and we owe
it to them to provide this opportunity to be
in control of the computer for once. The
problem is pedagogical rather than one of
prerequisites. What we need is a set of
activities for various levels of student
ability and interests.

It would be glib for me to say that this is an easy task, but I am busy working on the problem, and believe it can be solved satisfactorily.

The Curricular Offerings

At Mountain View High School we have tried to broaden the audience of our computer offerings by using five different approaches. They are a two week course in BASIC offered as a unit in certain math classes, simulations and games which have applications in various subjects offered in the school, easy access to the computer at designated times for game playing, classes in programming which offer experiences at three distinct levels of math ability and interest in programming, and a well attended adult education class in BASIC at night.

First let's talk about what we do not do. There is no CAI per se because I have not figured out how to accomplish this in a class situation with our present facilities, and I await the software which can provide the organization, record-keeping, and pacing which are required in order for CAI to actually be a help to a teacher. Also, I await the teacher who wants to give it a try. Another thing we do not do is any kind of instruction about hardware or lower level languages. This is simply because I know almost nothing about these things. I am a teacher, a sometime programmer, and somewhat of a mathematician, but I have had a nervous awe of electronics ever since my younger brother became a ham operator in the sixth grade and began speaking in tongues. I hope that others will know some ways to approach these omissions, or will suggest other activities we could usefully pursue.

The Two Week Course. This has been the backbone of our program. It has brought programming to a large number of students and it has been effective in building enrollment in our programming classes. Furthermore, it was a major factor in gaining permanent status for the computer program.

This course is based upon two premises: (1) A remarkable amount of programming in BASIC can be taught in two weeks, and (2) if programming is introduced as a non-optional part of a course, then some of the barriers against participating are transcended (for example, none of the regulars are around to make people feel dumb, and most students tend to try harder when something is expected of them, compared to a situation in which the activity is optional).

The two week course was first offered in 9 math classes during the Spring of 1977. The classes were Geometry, Algebra II, Trigonometry, and Calculus. Modified versions were given in three other lower level classes. I taught each course with the assistance of the regular teacher. I am confident that the same proportion of students who pass these classes came to understand the essentials of the following statements: INPUT, LET, GOTO, PRINT, and IF..THEN, how to work with strings, and how to use the system commands like GET, SCR, LIST, NAME, SAVE, and KILL.

At the minimum each student wrote and ran 6 programs, some related to their coursework, one using strings to produce a conversation with the person at the terminal, and some of general interest like computing a batting average, or a grade point average. Let me emphasize that this was the minimum. In each class there also appeared advanced programs such as one similar to NUMBER, change calculators, prime generators, and, my favorite, a program which guesses the person's secret number. Each of the writers of these particular programs had never programmed before, and accomplished these results within two weeks.

The facilities during this first go at the two week course were four teletypes time-shared to an HP 2000F at the Santa Clara County Office of Education. Everything was rented, because this was to be a pilot project to prove the value of the program and to help us better evaluate what facilities would be needed. I will discuss facilities later in more detail, but let me add here that four stations seemed sufficient, although one of the few complaints from the students involved the need for more terminal time.

Games and Simulations. HANGMAN in French and Spanish (and also in Tagalog!), LUNAR in Physics, CIVIL and ELECT in History classes, story programs in Creative Writing, NEWTON (getting across the stream against the wind) in Trigonometry, demonstrating limits in Calculus. These are all activities which we have carried out with classes. We reach a lot of students this way, and a lot of teachers. There are more things we can do, and improvements we can make with these. We have not, for example, offered a coordinated unit using, say, the Huntington teaching materials. If others have, I would appreciate hearing from them.

The problem of interacting with other teachers and their curricula is an extremely delicate one. We computer teachers must tred carefully – our programs and equipment must be easy to use (or we supply student assistants), and our offerings should be more than just time fillers. The entire class should be able

to get involved, and this may stretch our imaginations, given our limited facilities or space.

Also, we must work to convince our colleagues that we can contribute to their instruction, because don't forget that they are probably not from either group A or B described above. I hope that other teachers trying the same things will get in touch with me, for this is an area of great importance to the increased success of our programs, and one in which sucessful strategies are very critical.

Game Playing. The computer facilities are open various times during the week for game playing by any student in the school. The HP system library is full of games, many with no educational value. We do not encourage these games, but they do attract kids. We try to show them the more interesting (from a teacher's point of view!) alternatives to FOOTBALL, BLACKJACK, etc, and there are many on that system.

When we have changed over completely to our own computers there will be more control over the available games, but I still think that game playing will continue to be an important part of our program. There is a lot of thinking and problem solving required by the right selection of games, and I think such things are a useful addition to a school's extra-curricular offerings. Game playing leads some kids to inquire about programming, gives others ideas for original programs, and at the minimum lets a kid interact with a computer in a non-passive way.

Programming Classes. There are now three sections of a one semester course in which the primary activity is teaching BASIC. In the course description I made Algebra "recommended but not required", which is consistent with my desire to reach more students and with by belief that programming can be taught to a wider range of people than is usually thought possible. Students from each of the groups A through E described previously enrolled in the class, and it was immediately apparent that the course could not follow a traditional format of lectures and assignments.

The course now operates on three different levels. Some students just work on programs which they and I agree would be productive, and ignore the regular schedule of assignments. This schedule involves completing four chapters of our text, but almost every book or program assignment is divided into two levels of difficulty (students choose for themselves which level to tackle on each assignment).

I would be glad to discuss the course

in more detail with anyone interested, but let me close by highly recommending our text - Computer Progamming in the Basic Language, by Neal Golden (Harcourt Brace Jovanovich) - it is cheap and full of good assignments at almost every level. It does, however, assume a minimum of Algebra I, which has caused me to generate a lot of supplementary problems.

Adult Education. A night class in BASIC using our facilities is being offered each semester. Right now the format is 6 weekly meetings of three hours each. I consider this an important part of our program, with potential for exciting growth. If you think about a public school as a community resource, many uses come to mind for a public computer facility downtown. We are interested to talking to others about these possibilities.

Getting the Program Funded

Of primary importance in the initial stages of our effort to establish a computer curriculum at Mountain View High School was the involvement of several dedicated parents, including one Board member. These parents insisted that the school should have a program comparable to the other schools in the District, and did not let the matter rest until it was a reality. Almost an entire school year was required to produce a proposal, and after another six months final approval was obtained to begin.

In the spring of 1977 we began with four rented teletypes and offered the two week course described above. Its success produced wide spread interest in the computer throughout the student body, and led to a decision by the Administration and Board to make the program permanent. Other computer programs that I am familiar with in schools gradually grew over the years - a teletype here, renting or scrounging time there, with the continual struggle for marginal improvement each year. At Mountain View High School we now have adequate permanent facilities in large part because of sustained parent support in the early stages, and because we were able to involve a lot of students in a short period of time during the early part of the project.

Computer Facilities

Since October, 1977 we have been using two time-shared teletypes and two purchased Processor Technology "Sol" micro computers with North Star disk operating system and software. Each of the units has 24K of usable memory, which means that BASIC programs may reach about 10K in size. We use an Okidata

printer which may be accessed by each of the computers through a centrally located switch.

We have been very pleased with the Sol-North Star combination, and with the delivery and support provided by the Byte Shop of Palo Alto. As I write we are selecting two more units, one of which will be a Horizon computer from North Star with a dual disk drive.

The primary reason for choosing micros was cost, for the total purchase price of the four systems will be roughly equal to two years of rental of four time-shared teletypes. I was attracted to the DEC system which time-shares four ports, because of DEC's reputation and all the software which comes with it. But for their price we could purchase six of the Sol systems, and also DEC's maintenance costs over $2000 per year.

Having become convinced that micro computers had ceased to be strictly a hobbyist domain, I made the plunge and have been delighted. Like the Holiday Inn commercial, the best surprise is no surprise, and for me the best system is one you don't have to think about. We load BASIC in the morning and can continue through the day with students programming, loading, printing, and saving programs. Often at lunch time, or as a reward for my General Math classes, however, the flashy Sol games are brought out.

Computers in schools must be very easy to use. Students should be able to concentrate on the programming and ignore the machine itself. Teachers should not need special skills or knowledge if the use of computers is to spread to more schools and to more departments within the school.

It is fine if one staff member can fathom the mysteries of machine language or hardware, but the overall program should not depend for its operation on this individual or his/her knowledge. Uncommitted teachers will not want to touch a system which seems inaccessible or arcane.

In addition to satisfiying these unusual requirements for a computer whose tradition is with the hobbyist, our micros offer nice extras. For those students who wish to pursue assembly language, this is possible. The four systems are totally portable, and can each be dispatched to separate classrooms (or to different homes during vacations!). The FILL and EXAM commands (PEEK and POKE in some languages) bring a fascinating extra dimension to BASIC. And recently a student discovered how to allow data entry without the carriage return ... and so it goes.

Our micros are symbolic of our entire program. Their cost, versatility, and ease of operation make it possible to bring computer education to many individuals who never would have had the experience, and yet they still can provide challenge and excitement for the brightest student. I am very pleased to have joined the ranks of the micro computer, and especially pleased to have been able to join on my own terms rather than yours. There are a lot of teachers and students like me out there, and I predict that next year this place will be too small to hold them all.

INTRODUCING THE COMPUTER TO THE SCHOOLROOM

DON BLACK

INTEGRATED COMPUTER SYSTEMS, INC.

3304 Pico Boulevard

Santa Monica, California 90405

(213) 450-2060

Formerly Director of Computer Activities

THE LEARNING FARM

2696 Valewood Ave

Carlsbad, California 92008

AT THE LEARNING FARM,

We have been playing with our computer for a year and a half now. All the students in our school use it in one form or another, as do the teachers. Less I mislead you, we have 3-5 teachers and 20-40 students in a 'alternative education' environment. Instruction is individualized, we try to maintain an 8 to 1 student-teacher ratio.

Needless to say, there was a great deal of resistance to the introduction of this new A/V technology to the school. I would get comments like "They scare me.", "I don't like computers.", and this was from the teachers! Distribution of a few biorhythm charts, computer generated poetry, and Weizenbaum's ELIZA helped dispel any qualms about the system replacing teachers or taking over the school.

Before I brought the computer into the school, I made arrangements with the teacher of a local community Junior College (Palomar College, San Marcos, California), Mr. Mike Michaelson to bring my students to the computer center a few hours a day for a computer programming class. Mr. Michaelson allowed access to the Univac 70/7 system and a classroom, much to the chagrin of the Computer Science Department.

My plan was to introduce the students to some computer games and gradually sneak in some educational courseware. Unfortunately, the college had no computer games, and frowned upon

their valuable computer being used for such frivolousness. So, back to the drawing board. While I was busy drawing up my new plan of attack - designing and programming the PILOT Interpreter/Editor System - unbeknownst to me, my students were busily typing in the code from 101 Basic Computer Games, DEC and What To Do After You Hit Return, People's Computer Company. They were doing this in the afternoon on their way home from school. Needless to say, this did not endear me to the Computer Science Department.

However, Community College policy is to serve the community. When our plight was brought to the attention of the administrators of the college via the parents of the students our fairy godmother appeared in the form of Dean Cootz, Dean of Science and Technology and Tom Dolan, Head of the Computer Center. These gentlemen, through the Mathematics Department, offered us unlimited computer time for the duration of the semester in order that my students might meet their high school requirements. We were required to allow the Business Administration students priority on the system.

Remember the plan? My daddy always said "planning is essential, but plans are no damn good". Much to my surprise I discovered I had five computer nuts, now programmers, and their friends - potential computer nuts. The friends were playing the games that the

programmers had implemented. The games that the audience had chosen were STARTREK (now a classic), POKER, MUGWUMP, HURKLE, WUMPUS, PIZZA, BIOSIN, 3DPLOT, MAZE, TIC TAC TOE and BEAT THE COMPUTER. This last game was of their own devising: take a game and reprogram it so that you have better odds, (there may be a lesson there).

My daddy also told me "If you can't beat'em join 'em". It wasn't too far from our schools educational philosophy to let the students play games (after all, the teachers play some less constructive ones), so I incorporated the games into my educational strategy. By this time PI/ES was completed so we started translating their selected games from BASIC into PILOT. The games that were not translatable, we used in some math context. The translation allowed the students to explore the fundamentals of computer languages, freeing them from a 'dialect' dependence. They learned personally how the implementation of a particular syntax forces a programming strategy.

At this point, EDUTECH Project had agreed to provide to the San Diego County Department of Education one copy of the PILOT Interpretor/Editor System in return for the loan of a Teletype and access to their Burroughs 6700 system for a semester.

We still have access to the system and are at the time of this writing, negotiating for an extension in exchange for courseware. My thanks to Dr. Jane Gawronski, Mr. Bob Doolittle, and Mr. Bill Cue, not forgetting the glib operator with the sagacious one-liners that appear on my terminal at 3:00 in the morning.

The final educational strategy was:

1) Introduction via playing games. This taught some basic skills:

   a) turning the system on & off, b) keyboard familiarity, c) loading and executing a program.

2) Elements of Computer Programming. Using the PILOT language I introduce students to the concepts Input and Output. I give them the 'T:', and 'A:' statements and have them write a simple program.

3) The Black Box. Between Input and Output a Black Box that performs mysterious operations:

   a) Move a string from Input to Output
   b) Remember something typed in.

4) Decisions. The computer may compare an answer with an expected answer, and make

a decision based upon this comparison. 'M:', 'Y:' and 'N:' instructions. (Match, Type if Yes, Type if No).

Some of our students lose interest following number 4, above. They have the option of leaving the course at this point with a functional degree of computer literacy.

5) The students are now ready for more esoteric concepts such as 'J:' - Jump, 'U:' - Use (call), 'E:' - End (return), 'C:' - Compute.

6) The students are now writing and debugging their own programs and translating from BASIC to PILOT. The translation teaches them some programming tricks that are used by more experienced programmers, and how language capabilities differ.

7) New tasks are introduced that are not possible with PILOT (such as array manipulation, File I/O, Function references) that occur in BASIC games. The students then dig into BASIC in earnest. In the same way, FORTRAN is introduced. Attempts to debug BASIC programs introduces the virtue of Structured Programming and Documentation. Our students discover programming on their own, with a little guidance from the staff.

When the other teachers witnessed the success I was having with the computer, their interest was aroused. The programming students introduced the teachers to the games that they had written, and even taught them the rudiments of programming. The computer terminal has now become the focus of school activities in the area of academics.

One teacher wrote an English lesson program using the MADLIBS program in PILOT. Students interactively supply a list of adjectives, nouns and verbs of the proper tense and the program uses Mr. Basener's story 'mask' to write a story. A student then wrote his own program to tell his own story.

Mr. Tim Dawson, our English teacher, uses the Stanford Writing Programs by Ellen Nold and Sally Cannom (as appear in People's Computer Co.) in his English classes.

Mr. Peter Brown the Director of the Learning Farm, uses some PILOT programs acquired from Maria Montessori of the Golden Gate school in San Francisco for the younger children (thanks to Ursula Thrush).

We have developed a few PILOT arithmetic and English Grammar programs. Our arithmetic programs get progressively more difficult and display the student's percent correct (grade) and average response time (time to answer a question) at each level change. The levels start with horizontal addition of two random-ly selected numbers less than 5 (5 + 4 = ?) and proceed in individualized increments to multiplication of two 3 digit numbers and columnar addition of three 3 digit numbers.

The gimmick here is the response-time. As the students master the material (grade = 100%), they begin competing among themselves for quick-er response times. These routines provide self-motivated drill and practice.

The grammar programs are also very simple so that there is a lot of positive action on the part of the student. One series of programs requires the student to identify a particular grammatical element in a given sentence, such as the verb. A second series requires the student to identify the part of speech of an underlined word in a sentence. A third requests a word of a particular part of speech, and then weaves the words into a story. In a fourth series, the student constructs a sentence from a list of words provided by the program. The program then checks the sentence grammar. These routines give the child experience manipulating symbols. Words become things - tools - to be used.

In order to make this material available to users beyond the Learning Farm we are working on documentation, the hardest but most necessary part of the system. This paper is perhaps an element of that task.

If you are interested in writing course-ware or lessonware, contact the PILOT Information Exchange at Box 354, Palo Alto, California, or the EDUTECH Project, Box 1023, Encinitas, California 92024, and you will receive some author's guidelines.

Let me leave you with this advice - write the documentation first.

EDUCATION OR RECREATION: DRAWING THE LINE

William P. Fornaciari, Jr.
Math Dept., Polytechnic School, Pasadena CA 91106

Abstract

There is, in any discipline, a fine line dividing serious work and play. So it is with computing in schools, and this problem (of defining serious work) is magnified by the large role that game playing and game development assumes in curricula with computers. It is inherent in interactive computing that CRT's tend to become recreational devices -- a visit to a university computer where students have virtually unlimited amounts of computer time on interactive systems (Caltech's PDP-10 or Dartmouth's facility, to name only two) will verify this. Games appear to be the sole activity on terminals (at least upon a general inspection) with little "serious programming" done at all (serious programming belongs exclusively to the batch processor, where there is usually a cash ante). With microcomputers being virtually universally interactive and video-based and the economical approach for use in secondary schools, the impulse to play games can be addicting beyond all reasonable value. After a student starts the nth run or version of Star Trek, how can you hold the line?

Introduction

Perhaps we might ask how many Klingons must be destroyed before a student ceases learning and is merely passing time, avoiding his or her academic assignments? The same question may be asked of an educational game, say the Huntington Project's "Pest Management"[1]: how many flies must the student wipe out before the game ceases to be educational and becomes solely recreational? For this program "as soon as s/he kills the flies cheaply" or "when s/he passes biology" might do; in general, "how much is enough?" might be answered by noting when 1) the student starts to modify the game; or 2) the student starts to modify the game, by resetting variables, to cheat; or 3) the student becomes bored with the game and either asks "What else have you got?" or gives up computer games and wanders off toward other ventures.

The game is a means by which we attract people to computers -- and perhaps hand out a little ethical education. In any of the situations of extreme behavior with respect to game playing or, sometimes, game development (at least of those games which we might consider with little educational value or simply variations of the same theme), the educator must reevaluate the role games play. Surely there must be other activities. I wish to present some experiences and observations as a preamble to a short forum on alternatives to games which will follow this talk at the Faire. Last year, Liza Loop [2] observed: "Don't introduce Startrek or gambling games in class. You'll never get the kids' attention back." Some of my observations are as profound, but designed for those who did not heed Ms. Loop's advice. They generally apply to older students (most students where I teach math and supervise computer activities have virtually unlimited time fro grade seven through twelve). For standardization, computer statements are in BASIC.

Getting Started: Problems

Without categorically rejecting games and their development in a curriculum, there must be some other activities the teacher can suggest as an alternative to trekking. Perhaps we need only look at the batch processor for the answer. Most will agree that time spent programming a 370 (or whatever) is time well-spent (never mind that the computer money might not be better spent in other areas of research or other approaches). We batch process because we have a problem to solve. Why can't this use be fostered in high schools? It is important to preserve the integrity of of the computer education department as a competent entity of problem solvers and not game players. Then, and only then, will we be asked by our colleagues to solve problems, and maybe develop a few. It is, of course, not as much fun -- at least at first. And, upon running short of problems, other approaches can be suggested. If work seems justifiable, then the time spent on games seem to be a little more justifiable -- some time off for a brief period of recreation. I'm not saying no one should play games or write new ones for computers; I've indulged quite a bit in this pastime and it's great. But other things can be achieved with as much pleasure, and it is important to develop a working spirit.

Game development reveals a lot: one of the most illuminating and frustrating afternoons was spent with a particularly bright and very imaginative ninth-grader whose descriptive abilities had not quite matured.

He was interested in programming a hide-n-seek game of the twenty-second century, where players took turns peeking from behind a barrier, semi-permeable to laser (or was it phaser?) pulses; you got so many chances to move or lase, or whatever, until you blew your opponent (he hoped would soon be the computer) off the face of the screen. "Whew!" I declared, "that's some game, but let's play it together without the lasers, please." With difficulty avoiding reference to the futuristic ordnance, this eager fellow (who happens to be a good chess player, good at math and not a computer novice) to agreed to show me his game with a chessboard, a couple of pawns, an invisible barrier which we both agreed to respect, and notepaper to record the results and rules. As it turned out, the game was totally undefined. von Neumann would have been both pleased (it's clearly a zero-sum game) and disappointed (it has no rules). I keep a paperback on game theory near the computers, and most are, needless to say, dissappointed at its definition. Corollary to this, it's a good idea to have a game development system: checkerboard, dice, cards, a copy of Hoyle, etc. near the computers.

The above illustrates one of the fundamentals that holds computing, and the methodology of clearly defined and logical procedures we subscribe accompanies it, together: if a person cannot verbally describe an operation, s/he surely cannot write a program to perform it. Problem definition is a constant problem (but the effects of its use are strongly felt in computing and elsewhere) -- when is requesting a problem definition asking too much? At one point, requiring students to request time with a clearly-stated paragraph was tried for a couple of days, but was not too warmly received; it was hoped this would weed out the monopolizers and those who were constantly writing trivial programs (10 PRINT "HELLO";:GOTO 10) as jokes. At best, you can pester students to do this, and to be self-consistent, this would require the person who might be curious enough to begin computing to request, in writing, to play MUGWUMP [3]. At worse, the whole operation might succeed, and the teacher becomes a miniature bureaucracy awarding time for successful proposals (this might be educational, but let's defer this lesson for as long as possible).

Working with the Students

Let your students define with you the operating procedure for allocating computer time (I've tried limited hours per week by sign-up sheet and it's really hard to maintain). Agree to practically anything, including unlimited time, but insist on each student's having at most three programs or projects (two is better) that s/he might work

on. Then when the project becomes bogged down, insist on clearly-defined questions and problem definition before you help them out (you won't be able to do so any other way). The only escape from this, for the undocumenting student, is help from others. Reserve the right to restrict this, and you, and your trained students, will straighten out the corner-cutters.

For those that begin by typing in games from the various sources, suggest that they limit the associated story to skeletal lines and go for the computation statements (note: it is imperative that linenumbers be preserved in order to assist in proofreading and debugging. Also, remember to acknowledge to source, author and typist/translator in a comment; early adoption of this practice will insure later respect of authorship and documentation -- comments may be minimized, but not removed. In later practice an original program should begin with comments of declarations, entry points, subroutines. The program will then seem to write itself.) Once a student has completed the guts and is satisfied that the game is meaningful, it is appropriate to append instructions, fancy I/O, etc. Maybe s/he will find the game was not all it was advertised to be; s/he will have saved some time (lots if s/he doesn't know how to type) or possibly have found something more challenging.

On the other hand, a BASIC INPUT statement without a prompt message is worthless. When posed with a solitary "?" during the execution of a program, I am tempted to respond to this ERROR 477 [4] with "WHO, ME?" and await the usual "REDO FROM START" (unless someone was just named "WHO, ME?"). While the computer misses the message, the programmer seldom does. Error recovery is a very important habit to develop. It is not to much ask for idiot-proof programs, with software designed to work with humans. Programs are complete when some proof of correctness has been demonstrated; we must instill the importance of documentation in all students.

Assembly Language

Z   Assembly language coding is surely one of the best activities for those who really understand BASIC to the point where they're almost bored or keyboard in short programs but fail to develop them. To be sure, programming in assembler requires time and dedication on the part of the teacher, and usually some classroom exposure to the instruction set. In some circumstances, this may be impossible to provide directly, but for those who a true prodigies, the manufacturer's monitor listing and the chip's manual may be all that is necessary, with a few video games from Dr. Dobbs' (here the means justify the ends) or

other source [5]. Programs, utilizing memory-mapped output via a VDM or VTI, are especially good, because the results can be quickly realized. Even if the program blows up, some video output has probably evolved, giving hope, necessary according to Joyce.[4]

Start students assembling by having them copy programs and patch I/O to your own system; pretty soon they'll come up with their own ideas, or possibly assist in getting some major systems software implemented. Mostly, by the nature of assembler, they'll be diagnosing errors away from the computer, defining their problems more carefully.

At present, I have a half-dozen students working on assembler programs. Excluding the student who speaks macros, two are working on programs in assembler because it's the only way to achieve the speed (ever try to write PONG in BASIC using POKE statements?), one is interested in robotics, and several others see it as an opportunity to increase their machine time, though they don't realize it is actually freeing the machines, while puting greater, but tolerable, demands on my time. In the end, I think, it should pay off as I get six reliable programmers to implemet software, debug, document and maintain the hardware. It's a chance to work more closely with the students, and sometimes you wonder who's the teacher. They're full of ideas.

The Unexpected from the Youngest

In the summer of '77 (which will be best remembered for "Star Wars" and a demand for more spectacular space games), I assisted Dave Kressen, a veteran educator, math and computer science teacher of the junior high (and younger) levels, in a "summer school" offering of programming micros for students post fourth through eighth grade. Dave, with several years' experience using Caltech's PDP-10 as the principal computer, had taken task to write his own BASIC primer, "The Mathematics of Programming in BASIC" [6], the previous year, compiling a dozen arithmetic problems to be solved, each adding successively complex programming techniques (I generally refer my senior high students to this text and suggest they try the programs to see how counting methods are used, deciding the best conditions on IF statements with the multiple statement per line BASIC used by micros; Kressen's booklet uses ANSI BASIC and hence, is not code optimized). For the most part, the older students were interested in games, Star Trek, and cornering the market on source tapes which they might trade at a cash profit, which is a problem to be solved. The new fifth and sixth graders, having a first "hands-on" experience, were going through the mathematical exercises and, with only a slight discpplinary hand, tending to business.

When the problem of generating primes by Eratosthenes' seive was assigned, quite an amazing transformation took place -- the younger students were absolutely spellbound by primes (it should be noted they understood what a prime was), and especially by the rate at which the gaps between primes grows. There was considerable speculation about when a gap of 100 might be found. It became necessary to dedicate one computer to generating primes overnight; this required a premature introduction to the one-dimensional array (to hold the primes between print-outs) and the two-dimensional array (to hold the lower of two prime and its associated contending gap).

The kid's ideas and questions on this particular of how to generate primes faster (Kressen's original program used trial division by all odds less than the square root of the odd integer in question) were unbelievable. Ten-year-olds suggested that it would be faster to try only known primes as trial divisors, and thus a bootstrap routine to generate the first primes to 1000 was written to produce a basis set. The problem, designed to introduce the GOSUB statement, captivated the class for over a week, started programs in prime-factorization, tests for perfect numbers and eager efforts on Goldbach's Conjecture (that all evens can be expressed as the sum of, at most, two primes). When asked about the apparent slow speed of the computers (a weekend to generate all primes between 200,000 and 900,000) most were able to identify the method of programming as the ultimate limitation and several suggested that the numbers could be represented as whole numbers rather than reals to speed things up.

Motivation and Incentives

Within most fields, incentives to produce a superior product are commonplace, arising out of competition rewarded by recognition. Practically every aspect of education offers incentives, competition and recognition, and so it should be within a computer curriculum. Certainly writing and debugging a particular program are milestones themselves, affording the student the satisfaction of mastering a demanding servant. Perhaps, also, the numeric result achieved will find its place among others in a completed piece of research or the cataloging of records or similar data processing effort.

Building a library of good software requies that we provide these three ingredients to the student programmers. An immediate recognition of a minimum level of quality occurs when others ask for copies of programs, and for works of particular value, requests that they be made universally available (with appropriate documentation)

introduce new incentives to the student to improve the next effort. We must always challege his or her capabilities in new areas.

A particularly effective recognition is to ask one competent programmer to write a particular program. In this instance it is particularly important to approach the student as a professional. S/he may or may not expect clearly specified I/O routines, but after having received them, s/he will appreciate their value. Commission your better programmers with good problem definition, some suggested reference materials or algorithms, and be surprised by the results -- it can even be done with novices, if you offer a little help. Two students wrote an effective class grading program as their third BASIC project in only several hours after some preliminary meetings about sorting methods, averages, medians, data allocation and trying two programs out of Kressen's book (they were able to translate mathematical algorithms into character algorithms after agreeing that "A" is no more equal to "B" than 1 = 2).

The concept is no different than giving a programming assignment to a classroom of students and assigning grades; in this instance, however, you deal with one or two (that's the "gradelike" incentive) and follow up by using the commission work, saving it as an example of good, complete work which might be the basis for the next program

A modification occurs when the school is part of a user's group. We are part of a group which has an excellent barter economy established. For each program accepted, we get box tops worth $15 in exchange programs (which usually sell for $3 to $5, which seems like a fortune to some kids.) It means we get recognition for contributing worthy programs, we can save some effort keying in programs we don't already have and we get to see what others schools are doing with similar systems. The biggest problem is a common medium of exchange. For users of identical equipment, there is little problem, but Grimes'[7] observation has been prophetic. Schools should be at the head of users groups -- there are too many opportunities and incentives lost without them. The programs should not be limited to BASIC, as assembler and CAI programs need encouragement and resources.

Finally, end-of-year recognition is no more out-of-place in computing than in basketball or scholastic achievment. Certificates of Excellence should be awarded by a knowledgable faculty/professional committee and should be vertically distributed. The incentives, direct and indirect, with adequate recognition will produce good competition. The combination result in a productive atmosphere where the students feel they are truly working, insist

on conditions suitable for productive work and become discriminating and efficient in their use of time and resources. They will also see the computer as a tool to solve real problems, because, in the atmosphere described, that is what they hve been doing throughout.

A Summary of Suggestions

The following might be useful in developing and maintaining a computing center where little question of the justification of its existence will ever arise. Some suggestions have been developed above, and others are offered without comment:

1) Computer educators must encourage colleagues in other academic fields to assign problems which require computation to achieve a satisfactory solution. Numerical solutions to long algebraic problems, simultaneous equations, even atomic structure and solid-state problems become reasonable but challenging for high school level science and math, as examples; social sciences can use simulations and statistics which are really just games with a lot of data.

2) Get and use an assembler. The problem solving techniques required to use assembly language are applicable in any field; furthermore, you'll be able to develop and implement a very extensive library of utility programs.

3) Start and maintain a library of programs, a use your students to get the job done. Insist on documentation before and after actual programming. Encourage competition and show recognition when earned. Participate in an exchange with other schools.

4) Try to promote the idea of data processing at the student level, i.e. maintain club rosters, newspaper advertising accounting on the educational computer. Have students write the programs.

5) Maintain a workroom atmosphere relative to the student age level. There will still be some frivilous game playing one, but usually as a temporary diversion.

If you must Trek, by all means insist on a few basic modifications to the probably most widely-found version by Lynn Cochrane [8]. The course bearing notation (1=east or right to 8.9) is found nowhere on (even astronomical) maps. Use the standard polar coordinate system where 0 degrees is to the right; try accepting angles greater than 360 and negative bearings (figure another way to abort the torpedo or engine command). For advanced students, have them indulge in radian measure (in units of pi), and, memory permitting, replace Spock with an "on-board" computer which returns principal angle arc-tangents given rises and runs (remember

the division by zero); the students can then think to figure out the proper quadrant adjustment.

Also, I'm a little incredulous about a photon torpedo's (even one from the Enterprise) capability to destroy a star — disable this awsome feature with a hard fix. To do all of this, you will have to look at the program. "What to Do After You Hit Return [9] makes its biggest point in the value of game playing when it suggests the game is not so important, it is how it is play and how it is programmed. You'll have to figure out how Cochrane makes the stars go away before you can save them. So for all the problems Star Trek has created, here's the praise it deserves as being a model program of efficiency. A good starting assignment for your students: explain where Klingons come from! And if they waste those torps on stars, or spend afternoons stalking Darth Vader, insist they be quiet about it!! Others are working!!!!

Conclusions:

I have attempted to define several areas that can be considered productive activity and those which are recreational. To justify a computer activity as productive is very subjective, but a reasonable definition might be an activity requiring that an active mental process employed to achieve a defined goal -- work, if you will. And if your goal is to destroy 23 Klingons in 30 years, to do so might require an active mental process, though usually not for long. If given challenging problems to solve, encouragement and a little incentive, students will opt to solve problems. And "problems" are relative, also. To a ten-year-old, his or her curiosity might be the biggest problem. To an older student, using a different language to represent the same data and procedures is truly a challenge.

One person's play is another's work — how or where do you draw the line? It is an important question to ask yourself, and to reflect thought and development in your computer curriculum.

References

1. Huntington High Scool Project

2. Liza Loop, "Sharing Your Computer Hobby with the Kids." First Computer Faire Proceedings, p. 156.

3. MUGWUMP is a number guessing game. see ref. 9.

4. James Joyce, "Human Factors in Software Engineering." Proceedings, p. 56. ERROR 477

is as vague as it appears here.

5. There are too many references to list, but Marvin Wizenread's video games, in several issues of Dr. Dobbs Journal, are particularly good, as well as reasonably short.

6. David P. Kressen, "Mathematics of Programming in BASIC." unpublished.

7. Peter S. Grimes, "Classroom Microcomputers." Proceedings, p. 165.

8. Lynn Cochrane, INTERFACE magazine, July, 1976.

9. What to Do After You Hit Return People's Computer Co., Box E, Menlo Park CA 94025.

LEARNING WITH MICRO COMPUTERS

Richard Harms, Santa Ana College
Santa Ana, CA 92706

Introduction

It is interesting that we should have in our language the idiom "A person learns a bit at a time". This paper describes an instructional approach based upon the micro computer equipped with tape cassette. The approach is structured around the premise that "A person learns a bit at a time". Small segments of any topic are presented to the student. The student works with the material at his own rate. Based on his responses, individual learning pathes are developed. Non-computer disciplines including accounting, journalism and psychology have been successfully implimented.

The Rationale

The problem of designing learning packages for computer is not a new one. Work in CAI (Computer Aided Instruction) goes back over the past ten years. The cost performance characteristics have traditionally been achieved by spreading an expensive resource (the computer) over many users and long periods of time. It is then generally possible to show favorable cost performance.

With the microcomputer the primary cost constraint in CAI is eliminated, namely the expensive resource. We can then concentrate on what should be our primary objective of computer learning - effective learning. It is this objective with which we have attempted to work at Santa Ana College, and this is the nature of my presentation.

Within the confines of the latest widely marketed microcomputer, a system has been developed for authoring and presenting CAI which requires very little computer ability of either instructor (author) or student. This system is an outgrowth of a successful system called GULP (General Utility Language Processor) which was implimented several years earlier on an educational mini-computer.

As is true of other CAI packages, two distinct processes are required. The first is the author's creation of the material and the second is the delivery of the material to the student. Both processes are discussed below.

Creating the Lesson

As a general level of sophistication within the microcomputer state-of-the-art, learning programs can, at most, be tutorial in nature, including sufficient opportunity for student drill-and-practice. The microcomputer of today by its very intent cannot be used for extensive simulation and gaming learning models. To prepare the tutorial lessons, then, the instructor must identify salient points which are student important. Prior material assumed known or unknown must be clearly defined. The instructor prepares briefs of each background point. Physical constraints such as screen size are considered in writing the briefs. Briefs of student important points are also prepared. Meaningful interactive questions are developed for each brief. Consistent with the initial assumption that learning takes place "a bit at a time", each fragment is kept discrete.

After the instructor has created the text and questions and answers for the student learning session, an interdependent relationship is established manually between the lesson fragments. This is, in fact, similar to the PERT process. From these dependencies a linear string of fragments must be produced. If ties exist, an arbitrary ordering must be made because the next step in the process is the transfer of all fragments to the tape cartridge device. The fragments are serially chained. Subsequent fragments are

response dependent. A correct student response may cause the program to route to one fragment, an incorrect student response may cause the program to route to a different fragment. This allows the most flexibility for developing materials geared toward individual differences.

The approach is not without flaw. If it is necessary to deliver the same material twice (i.e. the student did not "get it" the first time), the material must be stored twice on the tape. Good lesson design will keep textual material brief enough so that the material is still on the screen if the question immediately following the material is missed. Using this technique, the instructor can tell the student of the error and refer the student to the material still on the screen.

The program for storing the learning fragments, prompts the instructor, accepts his input, and stores the lesson on the magnetic tape. All the instructor materials are stored as data on the tape. In delivering the data to the student, the structure of each lesson is such that the delivery program is identical for all lessons.

Presenting the Lesson

After the instructor has prepared the lesson and stored it as data, multiple copies of the tape are made. Tape costs are low and the intent is to have available in the classroom a copy of any lesson for any student at any time. We are incoroporating the microcomputer into the classroom in much the same way that typing classes have traditionally provided a work station for each student. The class is conducted (in the traditional way), using the lecture method. Students may concurrently follow similar presentations on the microcomputer. Alternately, if the classroom presentation is inappropriate for the student (too easy, too hard, redundant, boring, etc.), the student is free to pursue learning at the terminal. Some time is allocated for learning only from the microcomputer. During this time the instructor is free to circulate about the class answering questions and helping on an individual basis. Finally, outside of class time, the terminals serve as a great help in providing interactive information. Not all students attend every class every day. We have found the students much more willing to stay with us to the end of the semester when they have a place where they may pick up what they have missed. We are very optimistic of the future. We look forward to larger micros, more capabilities, less expensive quiet printers and numerous technological advances which will continue to make the microcomputer the most exciting educational innovation in ten years.

BACK TO BASIC (BASICS)

David M. Stone, Teacher, ESS D
Box 932, Pacifica, CA 94044, 415-589-5900

The current press to get back to basics of education (The 3 R's) comes at a time when I, an elementary teacher, can get funded to assemble a computer for use in the classroom. With the help of students and other teachers, we have developed a program using BASIC (Beginners All-Purpose Symbolic Instruction Code), one of many computer languages.

Three "Computer Program Operators", who are students from my fifth grade class, set up the computer daily for the seven classrooms of 2nd, 3rd, 4th, and 5th graders who are using it this year. My suggestion to name them three CPO's seemed quite logical, I thought. My "Star Wars" conscious students named themselves C3PO's, politely, seeming to have agreed with me and correcting my misordered science fiction name all in the same breath. (It was days before I realized that it was not my suggestion to which they had agreed.) Their diplomacy and understanding has been important to all of us involved in the project.

For those of you who are developing computer projects or are warming up to the idea of bringing your local students some familiarity with this marvelous tool, I hope I will pass by closely enough to what you need to jog the right ideas to mind as it seems to have happened with the C3PO's.

The Need

First in importance, perhaps, in developing a computer project is motivation. When working with a class of thirty in fifth grade math, the mathematical ability of the students may range in an approximate bell shaped curve from second grade to high ninth grade with several of them grouped about fifth grade in ability. Chances are no two students would have the same set of strengths and weaknesses.

As a beginning teacher in fifth grade, I learned that the text book the students were using was excellent for developing concepts but short on developing computation ability. In the following years the only thing I

added to the computational part of the program was a drill in the math facts. Class averages in the ensuing years came up substantially. The importance of the drill in rote memorization of the math facts came home to me one day when checking how one student was doing, who always took about twice as long as anyone else in completing his drill but almost always got a perfect paper. He was in the process of counting five times five on his fingers under his desk. It took him 20 minutes at first to do the 40 most difficult multiplication facts. During the year his time gradually came down on the weekly drills to one fourth of his original time at the beginning of the year. He came back recently and said that he had continued to do well in math in junior high.

It took 20 to 30 minutes to grade and record those tests on the math facts in multiplication and division. Not everyone had the tenacity to stick to the drill and attempt perfection as that one student did. A fair number of students had reoccurring difficulty only with certain math facts while knowing others quite well. As you can see it all boils down to a situation that the computer can handle outstandingly well.

Implementing the Use of Hardware

Some students just need enough time to finish in order to improve. Some just need to have more drill on certain facts. Others could use the time better by occassionally passing up the drill for developing other skills. The setting where each student works at his own rate and level of difficulty is called individualized learning and is getting considerable attention in the field of education today.

Enter microcomputer nearing the end of its second year of being bumped over deep cracks in our sidewalks as it is rolled from room to room on its little AV cart. The children can see the computer through the clear plastic on the lower shelf which protects it from accidental injury. As the computer rolls into the classroom the teacher continues to teach. The computer operator plugs in the computer and the TV

monitor,gets the disc from wherever the teacher keeps it in her classroom, inserts it in the floppy drive, and starts up the first program which establishes the date for that day. As the monitor leaves, the first student in a predetermined order moves to log on the computer and the computer has chained from the date program to the math program. If the student is an intermediate grade student the computer will ask for his last name and an identification number. If the student is a primary grade student the computer has asked whether that student was there. If not the computer asks if the next student is there. If the next student types in "y" or "yes" the computer asks for his identification number. When the last student is finished the computer asks once more for each student who was absent the first time around. If time permits the computer also chains to an educational game which develops some concept such as "is greater than" ($>$) or "is less than" ($<$).

At the end of the morning or afternoon the computer operator comes in to remove the computer and returns the disc to where he found it. Meanwhile the teacher has continued to teach uninterrupted. Even in the lowest grades a computer operator aide can be found to correct small problems like "input error" or call the upper grade operator if it's something that can't be corrected by the keyboard.

One second grade substitute teacher I talked to one morning was very fearful of taking on the computer especially since a new logging on procedure was to be used that morning. I explained the procedure to one of my computer monitors in a few seconds. He explained it to his assistant and was not called back the rest of the morning. At noon I asked the substitute how it went. She was all smiles and relief. She said, "Great! It was simple!"

At the end of the day or whenever she has time the teacher takes her disc down to the teletypewriter to get a printout that looks something like A-1. Some teachers just send the disc down for a print out.

The Use

The primary concerns of the program are that the students always achieve an adequate rate of success and that the administration of the program is acceptable to the individual teacher. To the low achiever

success seems to be a strong motivator. To the high achiever success _and_ an adequate challenge seem to be the key.

At this writing it seems we are able to meet the needs of _all_ students. This is done in two ways. First, through variables in the program that automatically move the student from one level of difficulty to the next which are chosen by the teacher before the beginning of the year. Second, other variables may be changed for an individual after he has shown a need for more time or a different kind of problem. For example, the teacher may change this variable by herself by loading and running a program which gives her control over any variables which she may want to adjust for an individual student or perhaps the whole class.

You can see from the preceding report that the teacher is relieved of some time in correcting papers, making dittoes, and running them off (which also saves paper) by using a computer. The teacher's valuable time and experience is moved away from clerical work to focus on evaluation and even encouragement. (At the end of the student's drill, one of perhaps several comments tailored to his or her personality, could be printed out.) Control of the students' educational welfare is, therefore, comfortably in the individual teacher's control.

If what you would like to do is somewhat like what I have just described, some of the following hardware thoughts may be important to you.

Hardware Considerations

If funds are limited, get a computer which will be able to handle time sharing. When you get the money to expand to more terminals you can do it. Almost 200 students use our computer each week at my school. This gives them about two minutes each. (A _fifth_ grader can do forty multiplication facts in that time.) Two terminals would give each student twice as much time. Three terminals would ...... well, you can see my point.

The importance of some sort of timesharing hardware comes home when you only have to sit down at _one_ terminal to ask only _one_ disc to give you a listing of the events of the day from your students. Remember, computers are here to save you time _and_ work.

A timing routine or hardware device is important. Students who don't get on during the day because someone else is very slow get upset and they upset the slow student. Lincoln Semi-

conductor of Sunnyvale, CA has a rea-
sonably priced and, from my experience,
reliable timing board. What I'd like
to find though, is an affordable board
that doesn't need to be told the month,
day, hour, minute, and second when the
computer is first turned on. Something
powered by a small battery might do the
job.

The Future

        One hardware note for the future:
keep an eye out for interactive video
discs. Phillips and MCA are working
independently on players for the ed-
ucational and industrial (EIT)users
(1). Conducting a discussion while
using a computer controlled video
display could be just a year or two
away. Imagine controlling the ani-
mation, as it happens, to illustrate
your point. It may be the most en-
gagingly happy experience you or your
students have enjoyed in school.

                REFERENCES

(1)   Braun, Ludwig, Video Discs:
      Magic Lamps for Educators?,
      p. 14, People's Computer, Vol.
      6 No. 4, Jan.-Feb. 1978, 1263
      El Camino Real, Box E, Menlo
      Park, CA 94025.

                APPENDIX

        A-1
        Teacher   ?Stone
        Is this a parent report.?N

          January 14
        ID   NAME   COR   TOT   %   TIME   SGN   AV%   AVT   RND

         1   Randall S.   24   25   96   1: 29   +   98   1: 10   2
         2   Dave S.   35   37   94   2: 1   /*   94   1: 34   2
         3   Don D.   40   40   100   1: 20   -   100   1: 20   1

            COR = Number of Correct Problems
            TOT = Total Problems Attempted
            %   = Per Cent Correct
            TIME = Time on Drill Exercise
            SGN = Sign of operation to be done
            AV% = Average percentage of all attempts
            AVT = Average time of all attempts
            RND = Round or number of turns at the
                  computer

A COMPREHENSIVE COMPUTER SCIENCE PROGRAM FOR THE SECONDARY SCHOOL UTILIZING
PERSONAL COMPUTING SYSTEMS

Melvin L. Zeddies
1854 Pacific Beach Drive
San Diego, CA 92109

ABSTRACT

It is well known within the computer science community that computers are trans-disciplinary. They provide, in many ways, a unifying influence for educational experiences. Now that the shock of personal computer system availability has subsided, we must develop and implement programs in the schools that provide students with experiences enabling them to use personal computer systems.

This paper presents a computer science program for the secondary school. The program covers both hardware and software areas. Course titles, descriptions, outlines, and suggested references are included.

TEXT

Today we have available inexpensive highly reliable computer systems; personal computer systems. The impact of these systems has not been strongly felt yet, but it will be of ever increasing signif-icance as these systems become widely used. Right now we need to utilize these fantastic little machines in our educational institutions. They are not only capable of performing great amounts of mundane work at a fraction of the human cost in time and energy, they are a marvelous teaching resource, allowing for wide application in every subject area. They have enormous potential for providing students with the oppor-tunity for creative thinking, problem solving, and expanding awareness, also synthesizing knowledge.

It has been several years since personal computer systems first appeared upon the scene, students in some classes have had the oppor-tunity to develop a basic system. But what now? Now that we have the computer system, what do we do with it? Unless we address ourselves to this question, schools will become even more out of date, and the computer will become an idle piece of furniture in the same manner as the overhead projector, and the tape recorder.

We must take advantage of the myriad possibilities computers offer. We need to develop and implement a scheme which will utilize computer systems in the schools. We need a computer curriculum that can be incorporated into the offerings of any school with a minimum of dis-ruption, and which will provide students with the background and knowledge needed in this area. The following courses, several of which have been implemented, are proposed to enable the schools to begin up-dating and incorporating the computer into the general curriculum. Com-puters are already in the mainstream of society, and should have that position and acceptance in the schools as well. These courses are extremely relevant and are vitually needed by today's technologically extended student. The proposed curriculum would form a strong, viable program which would provide students with the knowledge and abilities needed at this time, to function in an increasingly computer oriented society.

Each of the following courses could earn the student 1 semester of credit and have a maximum duration of one semester. Students might move through the sequence in a variety of ways, depending upon their interests and abilities. However, the first two courses are prerequisite to all others. Students may elect to challenge the contents of a course by examination, which is oral and written, and is administered by a computer science teacher. Successful challenges would move onto the next course in the sequence. It would be at the descretion of the principal whether to grant credit toward

graduation for successful challenges.

Course descriptions are given in outline form only, space does not permit any detail.

Computer Science I

An introduction to the area of computer science which would include programming in BASIC, a study of computers in society, and an examination of the characteristics of computers.

Objective: At the termination of the course each student will have developed, debugged, and run 10 programs that utilize the language BASIC, and the techniques of structured programming. Students will also have developed one project or program on a topic of their own choosing. All programs and projects will be presented to the instructor with a run and a listing of each program.

Topic sequence:
What are computers?
   limitations
   advantages
   human problems
Communicating with your computer
   terminals
   cards
   other
Programming
   structured flowcharts
   programming in BASIC
   debugging techniques
Developing your own programs
   determining if you can do it
   limiting the problem
   using your resources
   testing your hunch
   final production and publication
Course project.

Selected references:
Albrecht, R., et. al. BASIC.
   New York: John Wiley and Sons,
   1973.
Hamming, C.L. Computers and
   Society. New York: McGraw-Hill
   Book Co., 1972.
Kemeny, J. Man and the Computer.
   New York: Charles Scribner's
   Sons, 1972.
McGowan, C., et. al. Top-Down
   Structured Programming Techniques.
   New York: Petrocelli/Charter,
   1973.

Computer Technology I and II

This course presents the student with the opportunity of studying the computer and how it functions, maintenance, and troubleshooting techniques. The use of test equipment and component replacement are also covered.

Objective:At the end of this course each student will be able to determine if a system is functioning properly, if not to troubleshoot at the block level using appropriate test equipment and techniques to bring the system up once again. The student will also be able to identify and replace malfunctioning components at the board level.

Topic sequence:
What do computers do and how
   do they do it?
Block diagram level of operation
Electronic circuits
Component level of operation
Test equipment and it's use
Troubleshooting strategy and
   techniques
Comprehensive examinations
   written
   practical.

References:
System manuals for the specific
   systems being used
Test equipment manuals
Handouts prepared by instructor

Computer Technology III

This course provides the student with the opportunity of exploring the construction of computers and related equipment. Students will be encouraged to construct from kits or components, items of use in a computer system.

Objective: By the end of the course each student will have developed a computer device (anything related to computing) and demonstrated its proper functioning to the instructor and/or class. The student may use any resources within the community.

Topic sequence:
   TTL

Integrated circuits
Theory and application of
   experimentation
Techniques in electronics
Design considerations in
   electronics
Use of design information
   (manufacturer's data)
Project.

References:
   Articles from issues of:
   BYTE, Interface Age,
   Personal Computing,
   Kilobaud, and Popular
   Electronics.

Computer Science II

A study of the application of
decision tables, modularity,
data structures, sorting, and
simulations.

Objective: Each student will
be able to produce at least one
computerized simulation by the
end of the course. The topic
or area of the simulation will
be agreed upon by the instructor
and student. A listing, and run
of the simulation will be present-
ed to the instructor as the course
project.

Topic sequence:
Modularity in programming
Decision tables
Applications of decision tables
Files and their uses
Matrices
Cassettes and discs
Sorting
Developing simulations
   what are they?
   characteristics of simulations
   designing simulations.
   programming
   implementing
Project.

References:
Flores, I.(ed). Computer Sorting.
   Englewood Cliffs, N.J.:
   Prentice-Hall, 1969.
McDaniel, H. (ed). Applications
   of Decision Tables. New York:
   Bradon/Systems Press, Inc.,
   1970.
Maidment, R. and R.H. Bronstein.
   Simulation Games. Columbus,
   Ohio: Charles E. Merril Pub-
   lishing Co., 1973.
Selected articles from: BYTE,
   Creative Computing,

Interface Age, Dr. Dobb's
Journal, and Kilobaud.

Computer Applications in Business

A survey of the applications of
computers to the field of
business. Students will develop
applications in any area of
business: marketing, management,
accounting, etc.

The actual content of this course
will vary from student to student.
Specifications of the course
content will be stated upon
a student-teacher contract.
(see Appendix B)

Sample Objective: The student
will develop and run a computer
program that will process
data in the area of accounting.
The program will produce a
full inventory, with sales
trends on a month by month basis,
and include an automatic reorder
feature. This project will
be completed and submitted to
the instructor before the
termination of the course.

References: It is suggested that
the student be encouraged to
survey the publications: BYTE,
Interface Age, Creative Computing,
etc. Also contact business persons
in the community for information
as to what is actually needed.

Computer Applications in Mathematics

This course provides the student
with the opportunity of exploring
computer applications in the area
of mathematics, including: number
theory, geometry, numerical
analysis, and statistics.

Again, the actual content of this
course will vary from student to
student. A student-teacher
contract will explicitly state
what is to be accomplished.

Sample Objective: The student
will develop and produce a
computerized study of the residues
in mod 7 systems. The developed
materials will be presented to the
instructor at the end of the course.

References: Mathematics texts,
and programming manuals. Again
the student should be encouraged
to review the periodicals.

Computer Applications in Natural Science

This course covers applications of computer technology to the areas of physics, chemistry, biology, earth sciences, astronomy, biomedicine, etc.

The actual content of this course is determined by student-teacher contract. (Appendix B)

Sample Objective: The student will produce a computer simulation in chemistry, specifically dealing with the determination of safe combinations of elements. The student will present to the instructor by the end of the course, all materials developed and tested, with a run and listing of each program.

Sample Objective: The student will produce a simulation in the area of physics that demonstrates the behavior of projectiles in any atmosphere (Earth, Mars, Pluto, etc.). The student will present the simulation to the instructor with a run and listing before the end of the course.

References: The references again will vary with the topic and the student. Standard texts in the subject areas, programming texts, and periodicals should be used as references.

Computer Applications in Social Science Science

This course provides the student with the opportunity of exploring applications in the social sciences including: psychology, history, economics, anthropology, sociology, and political science.

The content of this course will be determined by the student and teacher through the use of a contract.(Appendix B)

Sample Objective: In the area of political science the student will develop a computerized voting preference projection for a local election. The finished program should have been tried in an actual election situation. The materials, run and listing, with results of application, will be presented to the instructor

at the end of the course.

Sample Objective: In the area of economics, the student will produce a computerized stock market analysis that will include: highs, and lows for the year, and projections, based upon passed performance. The student will submit to the instructor a run and listing of the material developed during the course. This material must be submitted before the end of class.

References: The materials needed to produce computer applications in this area or areas will vary. It is suggested that the student survey the regular texts, and periodicals in the area of his/her study.

Independent Study in Computer Applications

This course will allow the student the opportunity of exploring applications in the areas of: music, art, dance, literature, counseling, etc. Areas not covered in any other course in the field of computer science.

The content of this course is so variable that contracts between student and teacher are needed. (Appendix B)

Sample Objective: The student will develop a computer program that will generate poetry, in the proper meter. The program will be presented to the instructor this includes a run and listing.

Sample Objective: The student will produce a computer generated musical composition, and play the composition for the instructor. This will be accomplished before the end of the course. A run and listing of the program must also be presented to the teacher.

References: In addition to the periodicals mentioned earlier, all available books, and persons familiar with the area of study should be used as references.

Note: Additional ideas and objectives can be found in Zeddies pages 61-67.

Computer Science III

This course provides the student with the opportunity of studying assembly language, and machine coding.

Objective: The student will be able to produce at least five programs in any area, in assembly and/or machine code. The student will also develop an approved project and present it to the instructor with appropriate verification of correctness and execution.

Topic sequence:
    Assembly language
        composition
        usage
        applications
    Machine coding
        composition
        usuage
        applications
    Programming in assembly language
    Programming in machine code
    Project.

References:
Appropriate manuals from manufacturers of cpu chip.
Instructor developed hand-outs.

Independent Study in Computer Languages

The course allows the student to study computer languages such as: FORTRAN, etc.

The content of this course will have to be defined through the student-teacher contract. (Appendix B)

Sample Objective: The student will develop, run and debug 10 programs. The student will present to the instructor a listing and run for each of the 10 programs. All programs are to be written in FORTRAN.

References: The references needed in this course will vary and therefore need to be stated on the contract. Include texts, and periodical articles.

Independent Explorations in Computer Science

This course allows the student to explore the areas of trees, searching, compiler writing, and other advanced topics of interest to the student.

Again the contents of this course will vary with the student. A contract should be used to explicitly state the material to be covered. (Appendix B)

Sample Objective: The student will develop an interpretor that will function within the limits of the available system. The completed project must be demonstrated for the instructor, and a written listing submitted at the time of demonstration.

References: References will vary depending upon the topic of study, however the periodicals should be surveyed and it is suggested that you consider:
Aho, A.V., et. al. The Design and Analysis of Computer Algorithms. Reading, MA: Addison-Wesley Publishing Co., 1976.
Gries, D. Compiler Construction for Digital Computers. New York: John Wiley and Sons, 1971.
Knuth, D.E. Fundamental Algorithms. vol 1 of THE ART OF COMPUTER PROGRAMMING. Reading, MA: Addison-Wesley Publishing, 1973.

The proposed courses are not to be traditional in nature. They are quite individualized and are probably best incorporated into the present curriculum in a multi-level configuration in which a classroom of students will be made up of several smaller groups of students, with each group pursuing a different course of study. This form of instruction allows a great deal of learning to occur between students, and enables the teacher to assume the role of a consultant, rather than the fountain of all knowledge. This also allows the students to observe the teacher in action as a learning being,

similar to the situations that would be found in a university or research center. In keeping with this format of organization, the teacher might want to consider the following: students are divided into groups of approximately 10, with each group having some beginning students, some advanced and some intermediate students working on projects of varying sophistication. The function of the groups is two fold: First to provide working groups with the opportunity to expand each student's understanding of the computer science area, and second, to allow groups to meet in seminars where students can report to the group concerning their past two weeks work, their problems, and their accomplishments. In these seminars much information is presented and helpful suggestions are freely offered, the teacher also participates but does not dominate.

The amount of equipment needed to implement this program would be minimal and could even be phased in over a period of two years. It is suggested that a minimum configuration for the entire program would include the following items.
1. A multi-user system for 4-6 terminals and at least BASIC, with discs, and cassette I/Os.
2. A small system one user for assembly and machine coding courses. This system could also have a compiler/interpretor for wider use.
3. A system with the capability of several languages. To be used in the more advanced courses and for heuristic explorations.
It should be possible for all the systems to be configured for use as a computer network when this would be desired.
4. A reference library that would contain books and materials covering:
    computers in society
    computer security
    computer hardware design
    manfacturer's data on CPUs
      and ICs
    structured programming
    algorithms
    computer applications in
      various disciplines
    compiler construction
    system development
    data handling.

Also students would be in need of materials covering:
    numerical analysis
    simulation construction
    Boolean algebra.

In addition to the above, subscriptions to the following periodicals should be maintained: Byte, Kilobaud, Creative Computing, People's Computer, Dr. Dobb's Journal, Personal Computing, and Popular Electronics.

A diagram of the course sequence is included in Appendix A. This diagram should clarify the sequence of courses as suggested in this paper.
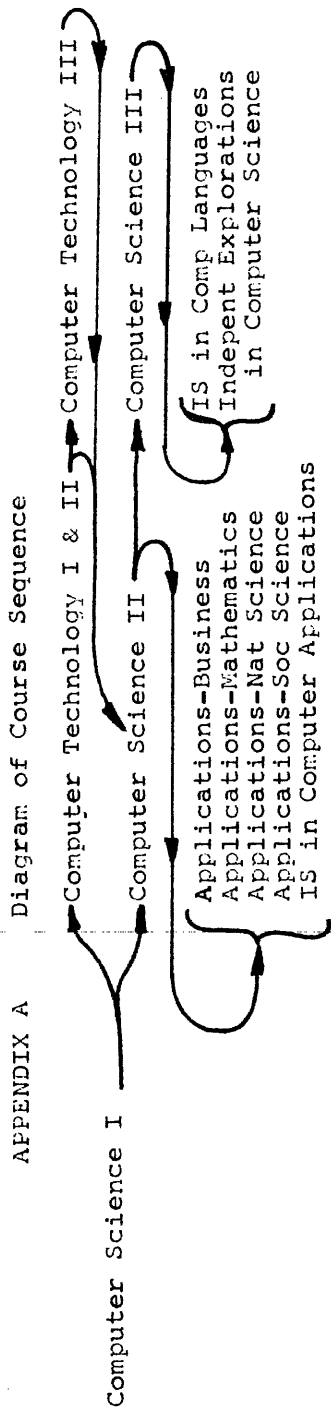
CONCLUSION

The computer science curriculum proposed in this paper is but one possible configuration. This configuration presents the student with the opportunity of pursuing studies in the area of computer science at a depth beyond what many educators would believe possible. However, the work being done in the field of computer science at this time is basically the work of young people, and one must not forget that young students are less hampered by the cultural inhibitions which limit and stultify older generations. The experiences of the author have convinced him that the proposed curriculum is not only possible, it is a necessary change.

The school system is guilty of severly limiting it's students, of failing to adequately prepare them with the kinds of knowledge and experiences needed in the face of the present day technology. We must do a better job of meeting the challenges presented by an ever expanding technology. We must utilize all the resources available and provide our students with the kind of educational opportunities so vital to them in this day and age.

BIBLIOGRAPHY

Zeddies, M.L., et. al. <u>Individualized Instruction for Gifted Students Using Computer Time-Share Systems</u>. San Diego: San Diego Unified School District, 1974.

Computer Science I

Computer Technology I & II → Computer Technology III

Computer Science II

Computer Science III

Applications-Business
Applications-Mathematics
Applications-Nat Science
Applications-Soc Science
IS in Computer Applications

IS in Comp Languages
Indepent Explorations in Computer Science

APPENDIX B

COMPUTER SCIENCE CONTRACT

Name:_____

Course:_____

Dates: begin_____,end_____

Faculty:_____

Course Objectives: _____

_____

_____

_____

Resources Required:_____

_____

_____

_____

Projects to be Produced:__

_____

_____

_____

Evaluation Criteria:_____

_____

_____

Contract Approved by:

   Student:_____

   Faculty:_____

   Date:_____

Contract Completed:

   Faculty Signature:_____

   Date:_____

   Grade:_____

Contracts to be made out
in duplicate with each party
receiving a copy.

MICROPROCESSOR COMPUTER SYSTEM USES IN EDUCATION
(or, You Can Do It If You Try)

Robert S. Jaquiss, Sr., North Salem High School, Salem, Oregon 97301

Abstract

Now is the time for the (micro)computer to become widely accepted as a teaching tool in the subject area classrooms of mathematics, biology, chemistry, physics, business and social studies.

This can be accomplished by the use of micro processors and the establishment on a high school level of a Computer Science Department which will implement a series of computer courses. These computer classes will form the basis to carry computer usage to the subject area classrooms.

Teachers in the subject areas will need to be re-programmed to accept computer usage and to include computer usage in their lesson plans.

There is need to make a 5-year plan for the acquisition of computer facilities, classes to be taught and specific programs to be implemented in specific classes.

Statement of the Problem

The computer has not been used to its full potential in education.

The problem will be restated in various positive statements.

The high cost is one reason computers have been used so little in education. Modern technology in the form of microcomputer systems is providing the capabilities to use the (micro)computer as a tool in the subject-area classroom at a more reasonable cost. The computer may be used as a teaching tool in many clever ways.

Teacher Training. Because this technology has come on the scene so rapidly, many teachers have received no instruction in the use of the computer as a teaching tool. In addition to not knowing how to use the computer as a teaching tool, these same teachers may fear the computer.

The solution to these various statements of the problem seems simple. Just make use of available technology in education.
This may be easier to say than to do.

I have a plan

Much software in BASIC already exists as the product of the last several years of research and computer use.

The hardware to make use of that software is now available in the microprocessor based computer system.

All we have to do is put the hardware and the software together in our school system.

What A Micro Can Do

In preparing this talk I wrote several pages about the capabilities of the microcomputer system. Perhaps those of us here already have a pretty good idea of what a microcomputer can do. On the other hand, perhaps some of you came here to the San Jose Computer Faire to find out what a micro can do. I guess that is why I came -- to find out more about what is, and what will be available in the micro world.

What did the Mini say to the Maxi Monster?
  "Anything you can do I can do better."
And what does the Micro say to the Mini?
  "Anything you can do I can do better."

If you haven't heard it before, then hear it from me. Practically anything you ever heard of a computer doing is being done by someone's microcomputer. In the exhibit area there are micros that talk, play music, draw pictures, make graphs and charts in color; some even compute.
Did you see the walking talking computer?

A micro can turn your lights on and off, lock your doors, call the fire department, keep your financial records and teach your children math or play games with them.

A microcomputer system can be used effectively in the educational process in your school in math and science, English and social studies. All that is needed is to get the hardware and the software together---- and one more thing--- you. You will have to do the getting and putting because you are obviously the person in your school that is interest in computer use in education.

How to get hardware

Establish a need. As you have noticed, I assume that some of you are teachers. OK Teach, try this. Walk into your principal's office at 7:30 Monday morning, lean dramatically on his desk, look deep into his eyes and say:

"The school district is failing in its obligation to teachers and students to provide modern equipment to use as tools in the teaching of academic classes and is therefore lessoning the value of the education provided to the students in our school.

"The district may be considered negligent in not obligating teachers to seek additional training in the use of computer-based instructional methods.

It is my guess that my now you have the principal's full attention. While the principal is trying to decide whether to fire you on the spot or give you a week's notice, you must quickly take a deep breath and continue-

"Students are being denied the opportunity of experiencing hands-on use of modern equipment made available by the development of the microcomputer.

The principal has probably gotten to his feet. You must say something soothing before he can get a word in edgewise. "Come let us reason together." And continue,

"Our citizens of tomorrow are being deprived of the opportunity to use, see, and manipulate computer based simulations, tutorials, and problem-solving techniques in the curriculum areas of biology, chemistry, physics, social studies, mathematics, business and even foreign language, because the equipment is not available for teacher and student use.

At this point, the script calls for the principal to be seated again. With a smile, because he has decided you are not really dangerous, and because he has remembered plan X, he has an answer ready for you.

The principal will say to you, "There just might be something to what you are saying Mr. Smith. Write up a proposal to take care of these problems. Justify all the parts of the proposal. Make out a list of equipment you will need, and the cost, and project it on a 5-year plan."

Of course he will have to approve the proposal and pass the proposal up the line to program planning and evaluation.

Justify the Need

I suggest that you subscribe to a number of computer journals and read some books. One article I like is "The Rhetoric of the Computer" by Barbara Marsh published in the Jan-Feb, 1978 issrue of Creative Computing. On page 131 she says, "I believe people ought not emerge from schools at the mercy of what they see on television, what they read in the newspapers (if, infact, they read), or what a computer analysis tells them is the case. The television may define what are "the issues", and computer analyses may provide "the answers," but we must try to provide an education which leaves room for students to make their own evaluations and decisions. They should be able to assess the appropriateness of the computer mediation of the information dealt with, and have some idea of where to look to find what computers leave out. I think one way to make this more likely to happen is to have teachers who are able to think about computers as persuasive media whose output must be evaluated."

This article is one of a growing number of articles about computer literacy.

Another article in the same issue, written by Cashman and Shelly with a very long title beginning, ""Hands-On" And Fast Turnaround..." has some good thoughts. Cashman and Shelly say, "Teaching programming without access to the machine is like teaching chemistry without access to a chemistry lab or teaching literature without reading a book."

Cashman and Shelly are writing about using computers only to teach computer programming The reader should not be so narrow-minded. Computers should be used as a tool in the educational process. The points made in the article are very valid, but the scope should be enlarged to all users and not just computer programmers.

ANY student should be able to use the computer in an interactive mode to the limits of his general ability.

Cashman and Shelly provide a quote from a book by John Kemeny, Man And The Computer, published way back in 1972 by Charles Scribner's Sons, pages 80-81. Allow me to read this quotation to you.

"I consider it imperative for the benefit of mankind that during the next decade computers become freely available to all colleges and universities in the United

States and that most students before
graduating acquire a good understanding of
their use. Only if we manage to bring up a
computer-educated generation will society
have modern computers fully available to
solve its serious problems. While computers
alone cannot solve the problems of society,
these problems are too complex to be solved
without highly sophisticated use of comput-
ers. I see three major bottlenecks that
must be removed if this goal is to be
achieved.

"First, most university computation centers
are still research-oriented. They are
typically operated in a batch-processing
mode with priorities given to a very small
number of users who need a great deal of
time. The philosophy of the university
computation centers must be changed.

"Second, college administrations do not yet
appreciate the immense favorable impact
that a good educational computation center
can have on their institution. I would like
to propose that by 1980 no college or
university should be given full accredita-
tion unless computer services are freely
available to all students. Use of the
computation center must be considered the
exact analogue of the use of the library.

"Finally, the implementation of this pro-
gram for millions of students will take a
great deal of money..."

It is to quotations such as this that we
may turn to for justification of the
existence of a computer use program.
Kemeny, and others, looked to the future.
It is because of their efforts that the
university computer centers have greatly
improved since these words were written.

Some colleges and universities now require
a class in computer programming or
statistics to graduate. At some institutions
consideration is being made to make a class
in computer programming an entrance
requirement.

We might amend Kemeny's words to include
high schools: No high school should be
given full accreditation unless computer
services are freely available to all
students.

Kemeny had no way of knowing that the
computer on a chip would become a reality
so soon. While the implementation of a
computer usage program in high school will
take some money, the amount is many times
less than what would have been required a
few years ago, and the expectation is much
greater.

The solution

Simply get the hardware, the software, and
you all together. You see, it all depends
on you.

The school district should resolve to act
with all possible intelligent purposeful
planning to implement the use of the
computer as a teaching tool.

Computer Classes In High School

Your school should have a Computer Literacy
class to teach about computers and how to
use them, how to use canned programs and
how to do a little programming. Then the
school should provide a Beginning Program-
ming class for students who find they like
computer programming. Finally, there should
be an Advanced Computer Programming class.
It is the advanced class that will run the
systems, encourage others, write original
programs, modify the library programs to
work a little better and take the computer
to subject area classrooms for
demonstrations.

Eventually one of the teachers will take the
portable terminal to his classroom for a
week to run simulations in biology class.

This system is working for me and it will
work for you. I also have students on an
independent study program.

Costs

Someone is going to ask about the money
necessary to implement a new program. I am
sure that the purchase of the first 16-mm
projector for your school was a real event.
Someone had to decide to purchase the first
overhead projector. Believe me the money
is there. If you can show the need for the
program the money will come. The computer-
use program cuts across the whole spectrum
of studies in the subject area classrooms.

The school district can afford to spend
one-fourth of one per cent of its total
operating budget for computer education
and the program to use the computer as a
tool in the subject area classroom.

What a Microprocessor system can do.

I was watching a program called NOODLE on
a PET computer today. Fascinating. The
little dot goes chasing around on the
screen leaving a line and little rectangles
behind it. With the proper persuasion I
suppose the little dot could be taught to
make graphs or maybe write one's name.

The computer, keyboard, CRT and a box to hold it all together costs only $800 with the additional memory option. A terminal usually cost more than that.

I was going to try to define a micro-computer system and then I could talk about available software. But the system cannot do anything without

Available software. For software I was thinking first about the Huntington simulations. The Huntington I and Huntington II simulations were written in primative BASIC by groups of knowledgable teachers for use in subject area studies.

Each program is not very large. Most of these simulations run in very little memory. The Huntington Programs are available from Digital Equipment Corporation. The Huntington I programs are available in listing form. The Huntington II programs are available on punched paper tape and come with three booklets, one each for the teacher, the student, and for additional resources.

The various computer journals have published a vast amount of computer soft-ware. Several computer companies have established user groups which maintain a library of programs available to members for the cost of reproducing the program. There is at least one private venture that is attempting to provide software to the user for a price, pay royalities to authors and make a profit. The hobby community exchanges programs so freely it may come as a surprise that someone would sell programs. The big companies have been selling software for quite a while. If you purchase a mini-computer you will find it won't do much until you feed it several thousand dollars worth of software.

Many books are available with program listings from simple games to General Ledger.

There exists a vast reservoir of Computer software and computer knowledge relating to computer education in printed form as a result of several projects in computer education that have been undertaken on large time-sharing computers.

Many of the computer programs thus devel-oped are available and can be executed on a microcomputer system. There is very little that a huge time-sharing computer can do that cannot be done on a microcomputer system. Those things that cannot be done on a micro I didn't want to do anyway.

Expectation Rises With Product Improvement

There was a time when I was satisfied with a teleprinter throbbing away at 10 char-acters per second to slowly and noisily print out the results of my program. No more. I expect my printer terminal to print at least 30 characters per second and do it quietly.

At one time I questioned the value of a CRT which leaves no written record but which is very quiet and very fast. I liked the security of that printed record to carry away with me.

My students do not have this hangup. They make a printed record when handing in assignments. More and more program output goes to the CRT in a form not amenable to character oriented printed output. So far I have insisted on a printer with each system so that output can be assigned to either the CRT or the printer.

What are the pieces that make up a minimum micro computer system? I suppose it would be good to have a check list of things we expect to have in a minimum system. When we purchase a system we will at least know if we do not have some of the items on the list.

Everyone is not going to come up with the same list of expectations. It is easy to find microcomputer systems that do not have everything on the list.

Minimum microcomputer system for Computer Use in Education

CPU board, name your favorite processor
32K memory
25-30 amp power supply
extra slots for expansion
CRT with ability to display memory
  locations, upper/lower case characters
Matric printer terminal (keyboard, 30 cps)
  upper/lower case, adjustable paper size
Extended BASIC 12-16K
Operating system
Assembler
Output may be directed to CRT or printer
  from within the BASIC language program
Cassette storage system

Such a system is available for a few thousand dollars.

Perhaps your list is different. No matter. The important thing is that you understand the limitations of the system you are thinking about purchasing.

Additional Options to expand the Minimum Microcomputer system for Computer Use in Education

CRT with graphics capability
24 or 48 lines of 80 characters
Color CRT with graphics and plotting
Matric printer with graphics and plotting
  bidirectional printing
  move paper forward and backward
  lower case descenders
Letter quality printer/terminal
Dual 8" floppy disk system with
  Extended disk BASIC with read/write
  files, chaining
Paper tape I/O for use in transferring
  programs from one system to another and
  to read punched paper tape prepared off-
  line.
Matrix printer/terminal with programmable
  character set for use in applications such
  as foreign language classes.

More than one system, compatible with each
other. Computer programs are written and
developed on the development system and then
transferred by cassette to the portable
terminals that can be taken to any class-
room.

First Things First

There are several thing you will have to do
first before your school can make use of
microcomputer systems in education.

First you must have a computer to run.
First you must be a programmer.
First you must want to be a programmer.
First you must be willing to put in lots and
lots of extra hours at school and at home.
First you must decide what kind of programs
you want to run.
First you must decide if you want an
expandable or a closed system.
First you must consider paper tape systems
cassette systems, floppy disk systems,
hard copy or CRT.

First you must convince the administration
of the need to use computers in education.
That is where we were a while ago. The
principal had just asked you to write a
proposal to use microcomputers in the
educational process. So first you write
the proposal.

One outline commonly used in proposals is:
-Statement of the problem
-The Proposal
  Curriculum changes
  Effects on students
  goals of the proposal
  teacher training
-Justification
- costs, and timeline for implementation

Goals

Lets take some of the problem statements
and change them into goals to write into
the proposal.

The school district will provide
one computer terminal for each-
- 200 elementary students grades 1-6
- 100 students in junior high grades 7-8
- 50 students in senior high grades 9-12.

The school district will-

-provide the opportunity for teachers to
take classes in computer uses in education
for college credit.

-encourage teachers to take these classes.

-provide additional support in the form of
books and magazines for the library.

-direct the principals to procede forthwith
to implement computer use in educationally
sound ways.

-recognize that every student has the right
to be educated in Computer Literacy.

-recognize that it is the obligation of the
school district to provide the opportunity
to the student to become computer literate
at a level commensurate with his/her
overall level of education.

-thoughtfully consider that the school may
be cheating those students who need
computer literacy/programming to become
effective citizens or college students.

-admit that teaching using computers should
occur whenever computers are an appropriate
and educationally sound aid in the overall
instructional process.

Value to the Student

Let us consider some of these same concepts
in terms of those values and advantages
that accrue to the student.

The student is able to-

-use applicable and available technology
in acquiring an education.

-investigate topics in his classes by using
simulations that would not otherwise be
possible because of
  time (heredity experiments in biology)
  danger to the student (heat, nuclear)
  mathematical drudgery (decimals).

-become computer literate to suit the
student's own prupose whether it be taking
an elementary class about computers, or
an advanced computer programming class.

-use problem solving techniques by
using prepared programs, or by
writing original programs.

Value to the Student, cont.

The student is able to-

-learn how to program a computer in several languages. (Some of my students speak four dialects of BASIC, FOCAL, FORTRAN and two assembler languages.)

-consider computer programming
  as a vocation
  as a recreational interest.

-satisfy existing and/or proposed college entrance requirements.

-have the experience of actually writing useful programs to be used in the education of other students.

-do system programming. (My students have revised the operating system for several reasons by using machine language patches. They have written their own device handlers and disassemblers and are working on an operating system.

The teachers of science, social studies, math and business will have the opportunity to use the computer to run prepared simulations in their classrooms.

As a result

-students will have the opportunity to run simulations, tutorials, and drill and practice programs relating to classwork, either as an outside class assignment or just because they want to.

-students will have the opportunity to play games on the computer. It is painful to some teachers to see students playing games on the computer but games represent problem solving and strategy in perhaps a different context. Many games reflect a learning situation. Sometimes there is an effort to disguise drill and practice as a game.

Playing games very often leads the student into computer programming because the student wants to design his own game. Programming a computer to play a game is in itself, problem solving.

Make a 5-year plan

Realize at the beginning that next year you will have to make a new 5-year plan.

Still, it is essential to plan where the program is going, or at least the desirable direction for it to go.

Realistic goals change a great deal depending on available equipment, both hardware and software. I hope you will pardon some examples from my own situation to illustrate the point.

In 1974 the computer at North was a PDP8I with 4K of memory. We had BASIC with 10 or 12 statements and a user space of about 1500 characters. We also had FOCAL which is a much better language. I have some great student programs written in FOCAL. There are a number of games. The most ambitious program is one that will solve 9 simultaneous linear equations in 9 unknowns. We did not do much in the way of program documentation.

The first computer I programmed was an early IBM monster that required a suite of airconditioned rooms. In 1957 we p rogrammed in machine language and optimized the program by strategic placement of variables in storage locations on the 4K drum memory. I was proud of my program that successfully added two numbers.

With this kind of equipment goals may be limited.

I still teach my Computer Literacy classes to use machine language to add and multiply two numbers, but only to demonstrate how the computer works.

In September, 1974, my 5-year plan was to teach computer programming on the PDP8I in machine language and FOCAL, and to teach time-sharing BASIC using a teletype connected to Oregon State University. The addition of a highspeed paper tape reader made it reasonable to use the assembler on the PDP8I.

In February, 1975, I ordered an Altair computer from MITS for my personal use. The world was about to come unglued.

In September, 1974, I had no reason to hope or even expect to hope that I would be able to do anything useful in computer programming classes.

The time-sharing Teletype was connected by leased long-distance telephone line and was costing a large amount of money. There was no educational program library on the time-sharing computer, although there was a large technical mathematics library. This was the state of the art at that time.

In my summer school classes I heard about marvelous concepts like PLATO and HUNTINGTON simulations. I think I remember saying, with some bitterness, "What good is all this going to do me?"

I ran many decks of cards through the IBM 360 on Free WATFIV. FORTRAN on cards and the slow turn-around-time really did not excite me very much.

Expectation rises with better equipment

So in September, 1974, I really had low expectations. I did have a $5000 per year budget for the time-sharing terminal. When Digital Equipment Corporation announced the CLASSIC in December, 1974, I was ready. The time-sharing was cancelled and the money thus saved was used to purchase the CLASSIC on a 5-year lease-purchase plan.

The Digital Equipment Corporation's CLASSIC computer has a PDP8A CPU, 16K of 12-bit core memory (effectively 32K characters), dual floppy disk (256,000 characters per disk) and a keyboard CRT (12 lines of 72 characters). We have since added a print-only DECwriter for hard copy output, and a Teletype ASR33 for punched paper tape I/O to the system. We also purchased software including OS/8, CLASSIC BASIC, FORTRAN IV, and PAL8 assembler. One of the students wrote an 8K FOCAL language processor for the CLASSIC.

.e have all of the Huntington II simulations and "101 BASIC Games" on disk. In inkprint we have all of the Huntington I programs and several publications of application software.

All this is costing us less than $250 per month.

Now the five-year plan has to be completely re-written. Expectation has gone up because of available hardware. In the meantime we have added 2 ASR33 Teletypes for off-line punched paper tape preparation of programs.

Our situation has changed completely. We now have a large software resource but only 2 terminals. Students are using the CLASSIC and the 8I all day long. After a brief time of playing games and running the Huntington simulations these students are interested only in creating their own programming masterpieces. The goal of using the computer in subject area classes is not being implemented because the computers are already being used fulltime for a worthy purpose.

For part of the spring term we were able to rent a timesharing terminal connected to the HP2000F at Willamette University in Salem. This was used briefly in junior high classes and in biology classes. Money came from a grant from Oregon Mathematics Education Council (OMEC) which was in turn funded by the National Science Foundation. This experience proved to me the validity of the concept of computer use in subject area classrooms.

In November 1976 we ordered a SOL-20 with dual floppy disks. This purchase was made possible by a matching grant from OMEC. When the unit had not been delivered by April, 1977, the order was cancelled and a microcomputer system was designed from on-the-shelf items at the local Byte Shop. This system was delivered in two weeks and was used the last 5 weeks of school. Several subject-area demonstrations were made during the last weeks of school.

I am telling you about the history of our local situation to provide an example of how expectations change with the kind of equipment available to be used.

In September, 1977, our computer #3 consisted of an IMSAI mainframe, Cromenco Z-80 CPU board, 24K memory, VDM-1, Cutts Cassette interface, Byte Saver board with modified TDL system Monitor, TDL BASIC, CRT Monitor, DECwriter LA36, ASR33 Teletype and a cassette player. This is a powerful and versatile system. Either keyboard can be used as the console (not both). Output may be directed to either the CRT monitor or the printer associated with the keyboard being used, under software control in several ways: as a monitor assignment, by the 'switch' command in BASIC, or by BASIC statements of PRINT and LPRINT. Programs may be input or saved on paper tape or cassette. This system cost $3800, plus the Teletype purchased earlier, plus the DECwriter. Many computer stores now have financing available. You should be able to purchase such a system for $186 per month for 5 years.

Let's review:
In September, 1974, available equipment included Computer #1, a PDP8I (4K) with an ASR33 Teletype, and a timesharing terminal connected by telephone to a university 40 miles away.

By September, 1975, the timesharing had been dropped and a higspeed paper tape reader added to the PDP8I. A CLASSIC had been ordered but was not delivered until Christmas.

By September, 1976, additions included Computer #2, the CLASSIC with DECwriter, CRT, ASR33, BASIC, FORTRAN IV, FOCAL, OS/8 and large amounts of program material on disk or printed.

By September, 1977, we had added computer #3, the IMSAI-TDL system described above.

I hope you don't think that is the end of the story. It isn't. We need more terminals and more options for a growing program.

If You Want Equipment, ask for it

I am happy to report that my school has just approved the immediate purchase of 2 new computer systems with 19-inch color CRTs, color graphics and plotting capability. One of these systems will be available at all times for subject-area classroom use. The other system will be used for program development most of the time.

I think I have been able to get this equipment because I have kept up a continual barrage of computer proposals, letters, and copies of articles from computer magazines to teachers, the administration and to the school board.

I have had a lot of help from OMEC, from people I met at the University of Oregon, from Willamette University in Salem, and from students and faculty at North. I subscribe to many of the computer magazines and I put in a lot of hours. I would like to think we have the beginnings of a good computer education program at North Salem High School.

I am not trying to get my own back patted. My message is just this:
-the software is available
-the hardware is available,
-you can do it, too.
Why are you just sitting there?

If you want equipment, ask for it.
Ask again, and again.

Design a program for your school that looks to the future. What is desirable now? --next year? --in five years?

Plan classes in Computer Literacy, Beginning Computer Programming, Advanced Computer Programming. Use BASIC, Assembler, FORTRAN, PASCAL......

Plan for computer usage in the subject area classroom. Read those magazines, write another proposal. Attend that conference, write another proposal. Visit another school, write another proposal, and keep hammering away.

North Salem High School cannot compete with Lawrence Hall of Science the first year, but if we go back into history far enough we will find there was a day when Lawrence Hall got its first computer terminal. Lawrence Hall just got a headstart.

I believe that the computer should be used as a teaching tool in subject area classrooms. A class of advanced programming students can be a great help in implementing this program. We are starting with the Huntington simulation programs and re-writing them with improvements that are possible with a better BASIC. We will answer 'yes' or 'no' instead of '1' or '2'. Now we will modify the programs to take advantage of the possibilities of a CRT that can display 48 lines of characters in any color and that has graphics and plotting capabilities.

We may get programs from books and/or magazines but most of the time we just write our own.

Write another proposal, Ask the librarian to order computer books and magazines. Answer the ads in the computer magazines by using the bingo cards.

You do not have to have a PDP-10 or an HP 2000 F or an IBM 360 to run worthwhile programs. Many BASIC programs will run on a stand-alone microcomputer that can be taken to any classroom.

You can do it if you try.

AN IDEAL COMPUTER SYSTEM FROM THE USER STANDPOINT

1. Is easy for the user to use; teacher, novice or programmer.

2. Provides for a number of programming languages.

3. Permits the user to access a large variety of on-line canned programs for simulation, problem solving or computer literacy.

4. Gives the user a choice of using CRT or teleprinter terminal. Output can be directed to either the CRT or printer from within the (BASIC) program while it is being RUN.

5. Is flexible so that terminals may be plugged in in any classroom for use in the subject area classroom. Allows the use of a large 25-inch CRT monitor, printing terminal, plotter or graphics terminal, or is a portable system that can be moved to the desired classroom.

6. Offers the user the options of graphics, a plotter, color CRT, line printer, CRT, or printing terminal either matrix or letter quality.

7. Allows the user to have his own user number and secret password, and to have protected files in his own private library, or else his own floppy disks.

8. Has sufficient memory so that each user has adequate user space.

9. Provides for easy transportation of programs to and from the system. Locally produced programs can be provided to other computer centers. Software secured from outside sources can be put onto the system via magtape, punched paper tape, cassette tape, floppy disk. In the case of several micro systems programs should be easily moved from one microcomputer to another.

10. Is expandable. Either upward compatible as in the PDP-11 series or the microcomputer system has extra slots available installed.

11. Provides the user a method to save his programs outside the computer (punched paper tape, tape cassette, floppy disk).

12. Provides system backup in case the system crashes. Systems do crash. In some systems the entire disk is copied onto mag tape at regular intervals. With a floppy disk system the user can make his own back-up disks.

I forsee intelligent terminal networks coming to the aid of microcomputer systems with each intelligent terminal being able to access a large intelligent disk data base. The intelligent terminal will go to the disk to get whichever language processor is desired and copy it into memory. Then the intelligent terminal will access the master disk again to load (or save) programs. The intelligent terminal will be able to execute the desired program in the selected language. A user will be able to select a CRT terminal or a printing terminal from which he can control a high speed line printer, reader punch, cassette tape or floppy disk. Schools will have a jack in every room so that any terminal may be plugged in. The system will provide color CRTs with graphics and plotting and a color camera copier. There will be a system plotter and a line printer capable of 200 dots per inch resolution.

I would like to teach my microcomputer to groan when the student makes a mistake and maybe play "On North High" when the program runs.

THE COMPUTER IN THE SCHOOLROOM

DON BLACK, Formerly Director of Computer Activities, the Learning Farm

now with

INTEGRATED COMPUTER SYSTEMS, INC.

3304 Pico Boulevard

Santa Monica, California 90405

(213) 450-2060

The Teachers Requirements for an Educational System

This presentation may be functionally divided into two sections: Hardware and Software.

Hardware

Packaging. The most important part of the system is the student, so I will start with what the student sees first, the Package.

It is well to have the system in one or two self contained modules. A jumble of wires and boxes can be somewhat intimidating. A clean, simple unimposing package, without too many lights and buttons on the other hand, invites exploration. Sophisticated Package.

CRT vs Hardcopy. CRT's are nice, quiet, fast, but they cannot provide Hardcopy. After I have spent an hour being creative, I would like to have something to show for it. If the presentation device provides Hardcopy,

-The student can take it home.

-The student and teacher have a record of the student's progress (we compile a portfolio of student progress in the form of this annotated output).

-The author has some feedback on courseware effectiveness.

If I have a choice between a CRT and a TTY, I choose the TTY for the reasons outlined above. Hardcopy.

Graphics. The only reason I would choose a CRT over a Hardcopy device is for a graphic capability coupled with a medium speed data rate. The graphics need not be elaborate, just easy to use. The graphic capability need be no more sophisticated than drawing a point (or 1/4" square) at x,y. For the author to

to be able to specify a vertical or horizontal line (as with The Apple II system) would be better, but we are venturing into software, or at least firmware.

Yet there is still very little that can be done with a TTY. This final decision may very well be a matter of taste. I still prefer the lowest common denominator, the TTY. Graphics is Fun, But Unnecessary.

Memory. All the courseware I have written has resided happily in 6K bytes. To this must be added the software and firmware. Lets give software 4K (the author language itself) and firmware 2K (monitor, I/O, integer multiply/divide, and utilities) so our system requires 10K RAM and 2K ROM. If we went to a nice round number like 12K, we would have room for some fun added capabilities. 10 - 12K RAM and 2K ROM.

Storage. Our system will require a form of storage that will allow the student to painlessly load a course without any training. I imagine a cassette tape with a big green 'LOAD' button. Punched Paper Tape is too slow and troublesome. It requires both patience and understanding, somewhat limited commodities. A floppy disk might meet our requirements, but a cassette tape could do so also, and it is still cheaper and more familiar. We all look forward to the day of the bubble memory cartridge, which could also allow virtual storage capabilities, yet the cassette is cost-effective today. Cassette (Until Bubble Memory is Cost-Effective).

A cassette would allow easy transportability of courseware, and with cassette standardization, inter-changeability.

So, out of this discussion comes:

1) Sophisticated Package

2) Hardcopy

3) Graphics is a Luxury

4) 10 - 12K RAM with 2K ROM

5) Cassette Storage

More on Cassettes. We can get by on the audio cassette toys, but in increasing orders of sophistication, allow the author to:

a) turn the cassette on and off programatically.

b) select read or write mode (say play or record)

c) search a tape for a specific record.

d) select a high speed search mode (read).

e) play back a pre-recorded message (audio)

At level 'd' we have the capability of virtual storage, if the response time on the cassette transport is such that it will allow the author 1/4 - 1/2 second response time for student interaction. Once we reach the 'e' stage, we shall truly have an audio-video device.

Software

I initially prefaced my remarks by saying that the student is the most important element of the system. Yet, in this field we are all students. The obvious corollary is the the User is the most important element of the system.

The first requirement of our software is a clear, simple and direct syntax. There should be no need to re-educate a specialist. The function of the language elements should be clear. The variable names should be self-documenting (i.e. multi-character). The syntax should be easy to parse by a micro-computer based system.

Let me introduce the PILOT language at this point. This 'author' language, as developed by Dr. John Starkweather, meets our initial requirements admirably. I have taught 9 year old children, with no computer experience how to program in less than an hour. It is no exaggeration to say that one may become proficient in the language within a few hours.

I have selected PILOT for its unique adaptability to microprocessor implementations. There is a brief description of the language at the end of this paper.

PILOT is available for the INTEL 8080 chip through the National Library of Medicine for free (public domain). There are implementations available for most mini and maxi computer systems written in BASIC, FORTRAN and even PL/1 through

user's group THE PILOT INFORMATION EXCHANGE, Box 354, Palo Alto, California. Let me plug my interpretor system written in FORTRAN and funning on Honeywell 6600, Univac 70/7 and Burroughs 6700 systems. This is a program product from the EDUTECH Project, Box 1023, Encinitas, California 92024.

The PILOT syntax is as follows:

[label] [operation code] [condition] : [object]

The operation code for the core language is one character:

'T' - Type, 'A' - Answer, 'M' - Match, 'J' - Jump, 'U' - Use, 'E' - End, 'C' - Compute, 'R' - Remark. This will come up later.

Documentation. The next requirement for any software is clear, simple documentation. This requirement is also first chronologically. At least outline the documentation before you begin designing the software. The User requires, no matter what her/his level of sophistication, a clear concise description of each element of the software tool. Be sure your assumptions as to the requirements of the user are clear in your own mind and be consistent throughout your presentation. Except for audiences like this, I usually assume a sixth grade reading level, that is a high school graduate.

Include a table of contents, an index, and references to other sections of text that bear on the subject at hand.

For example, when I say a description of each element, I mean specify that in an assignment statement (or LET or computational) that evaluation proceeds from left to right and that exponentiation is evaluated before multiplication division, addition or subtraction. But don't say it like that. More like this:

"Arithmetic proceeds from left to right. For example,"

(and include examples). "X = 5+6-3" "6 is added to 5 to give 11, then 3 is subtracted from 11 to give 8. 8 is then saved is 'X'". But, powers of number are computed before multiplication or division. Multiplication and division are computed before addition or subtraction. For example, in "X = 10-2*3" "2 times 3 is computed first to give 6. 6 is then subtracted from 10 to give 4. 'X' now has the value 4".

Use illustrations, too. "A picture is worth..." and so on.

Order Of Computation	Operation	
1	**	Exponentiation
2	*,/	Multiplication
3	+,-	Addition

(Note that I am assuming the reader understands +,-,*,/,** operations.)

Arithmetic (Assignment or Compute State-
ment). Integer arithmetic will serve our needs
just fine. In fact, floating point arithmetic
will create some problems due to rounding error
that the student and teacher will not be pre-
pared for at this stage (infinite loops due to
improper testing).

There are, of course, many applications
where floating point is necessary. An optiumum
implementation would be a combination of float-
ing point and integer (or fixed point) arithme-
tic as in Fortran or PL/1.

The arithmetic operations necessary are:
addition, subtraction, multiplication and
division. Exponentiation is good, but not
really necessary. There are algorithms
available for performing higher level functions
given a set of lower level primatives: multi-
plication from addition, exponentiation from
multiplication, the 'n'th root given the
arithmetic primatives. (Note - Include some
capability for testing for a non-numeric input
rather than bombing out a program).

String Manipulation. This capability is
a necessity for any teaching algorithm more
sophisticated than arithmetic or multiple
guess (choice).

The primatives necessary are:

1) Input a string (assign to a variable)

1b) Assign a string (" ")

2) Index (where in string 'B' does string
'A' occur, if it does?)

3) Substring (Extract from string 'A'
a substring beginning with location 'B' and
ending with location 'C'.)

4) Concatenate (Form a third string by
placing string 'B' at the end of string 'A').

5) Output a string

6) Conversion (Convert string 'A' to its
numerical equivalent, or convert a number to
its character representation).

7) A 'replace' or substitute operation
would be nice, but it may be woven from the
above primatives.

Relational Operations

1) Equal or Not-Equal and Greater Than
Zero

2) Greater Than, Less Than, Greater Than
or Equal To, Less Than or Equal To, Equal To,
Not Equal To.

3) For strings, a byte by byte compare
using the EBCDIC or ASCII colating sequence up
to the length of the shorter string. If strings
are equal, then the longest string has a greater
value.

4) A value of 1 for TRUE, or a value of
0 for FALSE.

Logical Operations. Although these prima-
tives are not necessary, they are handy. If
you include the above two capabilities, these
might as well be included also. This capability
also lends itself well to structured programming

1) Logical Or

2) Logical And

3) Logical Not (Unary)

4) Logical Exclusive Or (While not a
primative, it is useful).

It might be interesting to use more contem-
porary logical primaries such as:

1) NAND (Not-And)

2) NOR (Not-Or)

3) Invert (Logical Not)

For evaluation purposes, a value greater
than zero is TRUE, while a value less than or
equal to zero is FALSE. A Result evaluated to
TRUE is set to one, while a result evaluated to
FALSE is set to zero (state this in your docu-
mentation).

Language Elements. The basic language
functions that a teacher's language must perform
are:

- Present Information

- Get Response

- Evaluate (Compare response to expected
result.)

- Decision (Branch or conditional execution
based on result of analysis.)

Present Information. We need an output instruction that allows the author to display text, values of variables, and primative graphics if available (clear screen, move cursor). An added capability would be to evaluate and display the result of variable manipulations.

The PILOT language 'T:" statement meets most of these requirements.

T:WELL, $NAME, YOU HAVE #CA OUT OF #TOTAL+.

GET Response. The teacher needs to input the student response for evaluation purposes. The language requires the capability to identify the student response and save it for later evaluation. It is therefore necessary to assign input to variables. We require both integer variable and string variable input assignments.

The PILOT 'A:' input statement is our example:

A:#RESULT   Would assign a numeric response (integer) to the variable #RESULT.

A:$NAME   Would assign the input string to $NAME.

It is also necessary that a non-numeric response, when a numeric response is expected, does not 'bomb-out' the program. Allow the programmer the ability to test for a valid number. There is a conditional in PILOT that is set or reset if the last inputted string can or cannot be evaluated as an integer, respectively.

*AGAIN A:#ANSWER

    T B: THAT'S NOT A NUMBER, TRY AGAIN.

    J B:*AGAIN

(B is set if the number is Bad, G is set if the number is Good. The second and third lines of PILOT code are executed only if the 'B' condition is set, i.e. last inputted string is not numeric.)

Evaluate. The simplest form of evaluation, and the most popular, is to compare a student response with an expected response. A straight forward character by character compare (match) is the obvious solution: Does the student input contain the character string? The PILOT implementation is:

    M:string 1,string 2,...,string n

If the last inputted string contains the characters 'string 1' or the characters 'string 2' anywhere in its text, then the match is successful. Success is indicated by setting

a condition flag "YES" for success or "NO" for not a successful match. This condition can be used in the same manner as 'B' mentioned earlier. For example:

    T: WHAT IS THE NAME OF THE PILOT LANGUAGE USER'S GROUP?

    A:

    M: PILOT INFORMATION EXCHANGE,P.I.E., PIE

    TY: RIGHT!

If the characters string PILOT INFORMATION EXCHANGE or P.I.E. OR PIE occured in the student response then the student would get the reinforcing message "RIGHT!". For example, if the student responded I LIKE PIE. The system would respond RIGHT!

We also require the capability to manipulate inputted strings and numbers. For example, should an English teacher wish to use a transformational grammar approach in an English course, it will be necessary to evaluate an input string for syntactic validity.

    T: WRITE A SENTENCE USING THE WORDS

    T: JUMPED, DOG, CAT, THE, OVER, HOUSE, FROM, A, BROWN, BLACK

    A: $ANSWER

Rather than test for all syntactically valid responses, we would like to substitute the proper part of speech for each word and evaluate the grammatical elements. Let us assume a REPLACE instruction:

    REP: pattern, string, object

where all occurances of pattern in object are replaced by string:

    REP: JUMPED,VERB,$ANSWER

    REP: DOG,NOUN,$ANSWER

    REP: CAT,NOUN,$ANSWER

    .
    .
    .

    REP: BLACK,ADJECTIVE,$ANSWER

    REP: ARTICLE ADJECTIVE NOUN,N-PHRASE, $ANSWER

    REP: ADJECTIVE NOUN,N-PHRASE,$ANSWER

    .
    .
    .

    REP: SUBJECT PREDICATE,SENTENCE,$ANSWER

    T ($ANSWER="SENTENCE") : RIGHT!

As evaluation proceeds, the teacher has the option of displaying the parts-of-speech and evaluated grammatical elements for the sentence:

```
WRITE A SENTENCE USING THE WORDS:
JUMPED, DOG, CAT, THE, OVER, HOUSE, FROM,
A, BROWN, BLACK

the black house jumped over the brown

art adj    noun  verb    prep art adj

n-phrase            verb    prep phrase

dog from the cat

noun prep art noun

  prep-phrase

subject        predicate

sentence.

RIGHT!
```

Similar evaluation may occur with arithmetic and math word problems.

Another implementation of this capability is in an arithmetic assignment statement with string operations as with BASIC or PL/1. That is the method of implementation used in our PILOT Interpretor/Editor system (PI/ES),

Using the index, extract substring, and contatenation operations, I have written a subroutine to replace a given word with its part-of-speech, or a grammatical element with its term.

Referring back to our requirement for a simple direct syntax, I don't think our PIES system meets this requirement in the area of string manipulation, although it is the best implementation I have seen. Something on the order of the REP instruction seems more straight forward. As they say in the education industry, I will leave this problem as an exercise for the student. Let me know if you solve it.

Decision. The fourth elementary requirement is the ability to make a decision based on our previous evaluation. The buzz word here is "conditional execution". Most language implementations have a conditional branch instruction:

```
IF A THEN GO TO 1000      for BASIC
IF (I) 100, 200, 300      for FORTRAN
J Y:*LABEL                for PILOT
```

The syntax of this capability is far reaching, perhaps even devious. The syntax of this implementation defines the structure of the programs that are written with the language.

Those of us who are teaching the art of computer programming look upon the BASIC syntax with horror. (Ever try to debug somebody elses 4000 statement BASIC program?) A syntax such as PL/1 or PL/M however, is a joy. The IF A THEN DO;...END; sequence lends itself exquisitely to structured programming.

Although PILOT is not a structured language, it has a syntax that is unusual and easy to explain: Every statement may be conditionally executed.

The condition may be in the form of a flag that is set by a previous statement: G/B set by the last A:; or Y/N set by the last M:, or the condition may be in the form of an arithmetic evaluation:

```
J (A*B=5):*LOOP
```

The language may be structured by adding another instruction that will group statements into a block. This may be considered grammatically as a pair of parenthesis:

```
BLOCK:

T:  THIS SECTION OF THE LESSON IS ON
SENTENCES.

    .
    .
    .

T:  THAT'S GREAT.  LET'S GO ON NOW.

ENDB:
```

The spelling is unimportant, it could be called PROBLEM, or BEGIN, or SECTION, or GROUP, etc. The critical function is the conditional execution of the entire block. If the condition associated with the BLOCK statement is false, then execution of the entire block should be skipped (down to corresponding ENDB statement).

```
BLOCK (SECTION=4):

T:  THIS SECTION (NUMBER FOUR) IS ON VERBS

    .
    .
    .
```

Let's talk about subroutines. A subroutine reference is a branch or jump to another location in the program code and a return to the next location following the subroutine reference. PILOT calls this a U: for Use. The subroutine performs a programmer defined function that is used frequently by more than one section of the

program. While both PILOT and BASIC include this capability, there is one shortcoming shared by both languages: There is no direct way to send variables to the routines. Notice the differences:

FORTRAN and PLI/1

CALL SUB (arg1,arg2,"STRING",arg4)

BASIC

   LET A1=arg1

   LET A2=arg2

   LET A$="STRING"

   LET A4=arg4

   GOSUB 1000

PILOT

C:ARG1=arg1

C:ARG2=arg2

C:$ARG3="STRING"

C:ARG4=arg4

U:*SUB

   I would like to suggest an extension to the
U: instruction so that it may allow an Argument List:

U:*SUB,arg1,arg2,"STRING",arg4

MISCELLANEOUS. Other capabilities that have been suggested I will mention in passing:

BRANCH: #VALUE,*LABEL1,*LABEL2,*LABEL3,
           *LABEL4

   Jump to the Nth listed address depending on the value of #VALUE. Similar to the FORTRAN Computed GO TO:

GO TO (100,200,300,400),JVAL

or in BASIC:

   IF V=1 THEN GO TO 100

   IF V=2 THEN GO TO 200

   etc.

   I would like to propose a select statement SEL that would select the Nth element from a list:

SEL:#n,target,element1,element2,element3...

Usage would be as follows:

SEL:#random,value,1,2,3,4,5,6,7,8,9,0

or

   SEL:#SWITCH,*TARGET,*LABEL1,*LABEL2,
*LOOP,*END

   J:*TARGET

or

   SEL:#NUMBER,$RESULT,$STRING1,"RIGHT",
"GREAT","FANTASTIC"

   This statement will allow the BRANCH capability and a pseudo dimensioned variabel capability.

   External Storage. A File input or output capability is limited by the hardware. Assuming the lowest common denominator, Audio Cassette or PPT, we are allowed to READ.

   So the first capability we wish to implement is a sequential file READ, i.e. READ next record and assign to variable(s).

   This hardware capability also will allow us to CHAIN to another teaching program:

   T: YOU HAVE FINISHED WITH THIS LESSON. DO YOU WANT TO GO ON?

   A:

   M: YES,OK,SURE,YEP

   CHAIN Y:*LESSON2,arg1,arg2,...,argn

   This instruction is a combination Load and Run. *LESSON2 would be located, loaded from the tape into memory, and executed using the specified argument list.

   With this instruction, we have the capability of n 6K byte modules or chapters of lessons.

   A File WRITE instruction would allow the author to store the results of student performance for later analysis (grading, student progress, course effectiveness, etc.).

   READ:%filename1,$variable1,#variable2

   WRITE:%filename1,$variable1,#variable2

   A syntax including a filename (%filename) will allow upward compatability with systems that have more sophisticated I/O devices diskette, digital cassette, etc.). We might even include a syntax for Indexed files:

   READ:%filename'key,$variable1,#variable2

where "key" (following the apostrophe (')) is an integer from 0 to 255 that refers to the Nth record of the specified file %filename.

Summary

In summary, our language consists of:

1) Documentation

2) Arithmetic Computation

3) String Manipulation

4) Logical Operations

5) Relational Operators

6) Output to TTY

7) Input from TTY and assign to variable

8) Compare (match) student response to expected response

9) Evaluate response

10) Conditional Execution

11) Structured Syntax

12) File READ/WRITE

13) CHAIN to program using argument list

If there is an OEM who can meet these hardware/ software requirements for less than $1000 please let me know, I would like a few.

APPENDIX

Pilot Language Syntax

[*label] [op-code] [condition] : [object]

[*label] a 1-19 character unique string. (optional)

[op-code] a PILOT instruction as follows:

T Type out to the terminal the string contained in [object]. Interpret a string prefaced by # as a numeric variable. Interpret a string prefaced by $ as a string variable, otherwise type out the object as is.

A Accept an Answer from the terminal and optionally assign the input to the variable contained in [object]. SET/ RESET G/B conditions.

M Match (compare) the last inputted response with the string optionally delimited by comas (,) in [object]. SET/RESET Y/N conditions.

J Jump to the label contained in [object].

U Use [object] as a subroutine (save this address for return).

E End subroutine (no [object]). Return to instruction following last U statement.

C Compute. [object] is of the form [variable] = [arithmetic expression] Evaluate [arithmetic expression] and assign result to [variable].

R Remark. Internal documentation, ignore [object].

Multi character [op-code]'s are used for system dependent or user defined instructions:

REP [condition] : [pattern],[new pattern] and update [old string] when finished.

CHAIN [condition] : %[program-name], [argument 1], ..., [argument n] Load and execute (with no return) [program-name] using argument list [argument 1] through [argument n]

SEL [condition] : [N],[target], [element 1],[element 2], ..., [element n] Select the Nth element from the list and assign to [target]. [condition] is of the form: [Y/N] [G/B] [(arithmetic expression)]

Y is true if the last M match statement was successful. N is complement.

G is true if the last inputted string could be evaluated as an integer.

B is G's complement.

(arithmetic expression) is evaluated as True if the result is greater than zero, otherwise it is False.

[type code] [variable name]

[type code] is: '*' for labels, '#' for integers, '$' for string variables.

[variable name] is 1-19 characters, first character a letter.

SO YOU WANT TO PROGRAM FOR SMALL BUSINESS

Michael R. Levy
Jethro
70 Boston Post Road
Wayland, MA 01778

Summary

The "Hobby" manufacturers are turning more and more to the low end small business market in order to take advantage of this fast growing segment of the computer industry. This has created an ever growing need for programmer - system analysts, and many hobbiests or "hackers" see this as a means to earn money and take advantage of their programming skills. This paper will attempt to show the fledgling business programmer what he can expect from the small businessman, how to write proposals and run a business, and how to do small business system analysis.

The serious hobbiest market appears to be finite. It is a market somewhat similar to the serious ham radio market in that it requires a great deal of technical proficiency and knowledge and previous experience in order to be successful. When the home computer first burst on the scene about two years ago, this was a natural market on which the new manufacturers concentrated. Of late, the realization has dawned on many of them that they will have to find another area in which to sell if they are to survive in a crowded marketplace.

What Is The Market

The Small Business Administration says that there are 13 million in the country. They produce 44% of the jobs and 36% of the GNP and form 97% of all U. S. business. The definition of a small business by this Small Business Administration is complicated and makes use of a number of criteria. These may be number of employees and dollar volume of the business, and whether it is dominant in its field of operation. By that definition, a company such as American Motors could be considered to be a small business. The Table A-1 presents some interesting figures, and Table A-2 is another set. You will note that there is little consistency in the numbers or definitions, and I would note that while the market is large according to all established surveys, I suspect it is not as accessible as most small manufacturers think.

Characteristics of a Small Business

Reams have been written about the definition of a small business and its characteristics. Its purpose is to produce on a timely consistent basis, at a profit, a service or product suitable for the intended use. It is usually managed and owned by a person who is a specialist in some particular aspect of a product or service. He or she is most usually a clever marketeer, or a superior technician or expert in his or her chosen field.

Small business is a place for mature, intuitive judgment, not a place for the simple execution of a prescribed and defined routine. It usually exists to meet a specialized need which cannot be met by a larger company. Usually it fits into a chink or seam in the marketplace where size of the market is considered not suitable for a larger company, or where the entry into a new market would pose an unacceptable threat to an already existing profitable large company market. For instance, in the computer business IBM is slow to introduce new technical developments because to do so prematurely would threaten their existing rental and lease base. In fact, the commercial small business systems producers are in the same boat since they are eying some of the same markets that the small hobbiest manufacturers are examining. However, the existing pricing level is much lower than what the commercial small business systems manufacturers are accustomed to. In short, most small business companies, computer or non-computer are characterized by great flexibility and adaptability to rapid change.

Characteristics of Small Business People

Given the previous description of a small business, it should not be unexpected that a particular type of person will tend to gravitate towards this type of enterprise. They are more than usually independant and agressive, they are not very empathetic, and they mix aggressive behavior with a sense of dissatisfaction with their environment that

impels them to change things. Last, but not least, despite all of this motion and movement they are usually not to careful about detail. Their "span of control" is established by artful perception, intuition, and detailed experience. In a larger business this type of control is done by classical financial methods and budgets, or by formal analytical procedures. A person with this type of personality characteristics tends to change things rapidly and does not tend to think in algorithms. This can make a problem for a programmer who wants to work with small business people.

On the other hand, the programmer - systems analyst tends to be the type of person who, while independent, is not usually aggressive. He does not have too much empathy and is usually satisfied with his environment. If he is to be a good programmer, he must have a high degree of attention to detail. If you compare these traits with the characteristics of the entrepreneur, you will find that the only thing they have in common is a lack of empathy and a failure to communicate. This may seem an over-simplification, but my many years of experience has shown me that there is this lack of similarity in outlook and objectives which cause many problems between the business man and the programmer.

How Do You Then Merge The Two Interests

First of all, the business man will appreciate your services more if you run them in a businesslike manner. I find reams of information that are written about programming skills, structured and nonstructured, and which cover the myraid other factors that make up the tool kit of the proficient programmer and systems analyst. But nowhere do I find a rationally written instruction which tells anyone what they should reasonably charge. The basic rationale is important. How the individual applies it should suit his or her particular circumstances. What I shall present will equally suit the fledgling programmer - analyst as well as the small systems house or the full time two or three man partnership that programs for small business people.

You start with 365 days a year and immediately you subtract 104 of those for the 52 weekends. Most people calculate 10 paid holidays, or in any case the legal ones when nobody else is working, and add another 10 to 14 days for vacation. This is a rough calculation, but you are now down to about 220 saleable days per year. If one multiplies this number by the standard 8 hours per day, you get 1,760 hours. If, for instance, you were to assume a reasonable salary of $15,000 annually, that breaks down to an hourly charge of approximately $8.50 per hour. The catch is that that is at 100% efficiency and

rarely does any systems house or consulting firm end up being able to charge for all of the existing hours. Some of this inefficiency comes from nonchargeable hours expended on a customer, or for administrative details such as sales, finance and/or research development for a saleable package. It would be my guess that most systems houses really do not function at much better than 50% to 60% efficiency. Seventy percent is a consummation devoutly to be wished. With this assumption, it is only prudent to say that the hourly charge should be doubled to approximately $17.00 an hour. That means that we are at a fairly high level in hourly charges even before we have added any classical overhead.

Overhead can be anything from equipment rental, regular rent, automobile expense, telephone, paper and storage supplies. A little imagination should allow the incipient programmer to make a list of this overhead. Let us just use an additional $200 a month overhead, which we will distribute evenly between telephone and motor vehicle expense. If we take our mythical 1,760 hours and divide it by 12, we get 146 available hours per month. At 50% efficiency, that says we have about 73 chargeable hours. As an aside, it would be the lucky new programmer that has more than half of his time chargeable when he is initially starting out. If we divide our miniscule $200 a month overhead by 73 hours, that will give us approximately $2.75 additional charge; so, we are now hovering in the $20.00 an hour range for our services, and we have not even done anything yet. Most overheads, in reality, are not that low even for a part-time programmer, and certainly the efficiencies I am talking about are far from imaginary. So, it would seem for a full-time programmer, the charges have to be $20 to $25 per hour, and the part-time programmer with no overhead perhaps could charge approximately $10 per hour. I would note that plumbers and electricians and the rest of service trades which are utilized by small business charge at least comparable prices. If there is any sin that the system analyst - programmer commits, it is under-charging for his time. The moral is, if you don't value your time, don't expect anybody else to. The second rule that I would formulate is that programming and systems analysis is a time-selling business very much like an accountant or a lawyer. If you are serious about entering this business, you should sit down with a sheet of accounting paper and list all of your expenses and their monthly costs and go through an hourly calculation such as I have indicated in order to figure out what an adequate charge is to get a proper return on your invested time.

Why Consider A Small Business Micro?

When you talk to your small business entrepreneur, you should use the same criteria as you think you would use for any other piece of equipment. He is going to look at this as:

A. A labor saving device
B. For an increase in productivity
C. To provide better information

There are two types of information he is likely to want. One type is an application that tends to have very heavy computational requirements and is analytical in nature, and has relatively light file or transaction volume. The second type of application is characterized by a heavy load of individual transactions, that are heavy file oriented, and with relatively easy computational requirements. These transaction-oriented applications are the things that you hear all the time like payroll, accounts payable, accounts receivable, inventory, sales analysis, order entry, and P & L and General Ledger. The business man's rationale for using DP is that it is going to make things either easier, cheaper, faster or more efficient. Most systems analysts and programmers that I know lose sight of this. In this case, I am on the side of the businessman and against the systems analyst or programmer who falls in love with complexity for complexity's sake. The projected use should clearly effect economics in time, money, or effort, or it should improve productivity. If it improves service or provides better information to make decisions, then so much the better.

The way to start is simply to ask the small businessman what he is trying to do and to listen carefully. If you can analyze the flow of his product or service, you are at the beginning of your project. Most projects can be divided into four stages:

A. Analysis and specification - 30%
B. Design - 30%
C. Coding - 30%
D. Testing - 10%

If you look at this carefully, you will see that an analysis and design in any well executed project can take up to two-thirds of the available time.

You've now listened to the businessman, you understand his product or services in some respect and he now looks at you expectantly with the idea that you are going to say something useful. Think about it. What are you going to say? What is more important, what are you going to do?

The first think that you should do is to say that you will submit a written proposal. Emphasis on the written. It is a source of never ending amazement to me that many large or small companies that I have dealt with over the years that do not have written specifications and objectives for their computer installations. It is for this reason that McKinsey & Company at one point postulated that about 80% of the business computer installations in the United States were failures. They defined a failure simply in the terms that the installation was not doing what it was planned to do. Installations that were meant to provide total management information systems have in many cases degenerated into producing only purchase orders, payrolls, or inventory status. In many cases, these are high volume operations in large companies that have little or no relationship to what you are going to with a micro in a small business.

I would approach the actual proposal in the following manner:

1. I would have a conversation with the proprietor in which I would try to pin down that single thing which is impelling him to computerize. If it turns out to be "that everybody else is doing it", and that is the reason that he wishes to computerize something, I think I might quit right then and there. However, if he has a legitimate reason for wanting to computerize some part of his business, you should determine what are the priorities, and which things are most important. Try to concentrate on those.

2. I would propose to him a study, _for which he or she would pay_, which would probably involve a minimum of 40 hours of your time both to prepare and document. This study would be paid for whether further work occurred on the system or not. As I pointed out earlier, probably 30% of the time that you should spend on a job is spent in specification, and is really the highest order of skill that you use. Coding is not the game in this case, and in fact if you start coding a point too early, you will probably create a system that is not responsive to your customer's need.

The study should encompass the following areas: It should state explicitly at the beginning what the purposes of the system are to be, and additionally it should contain flow charts of the overall system and of the specific parts; also, some input specifications, which can be screen formats or anything else which is visible, as long as they are representative of the way the input really is to occur. There should be file descriptions and some

sort of output or report specifications. Again, they can be dummies. They do not need to be typed and they can be done on the standard report forms that most of the hardware companies produce. Again the simple fact is that they must be written down. There should then ! 2 a simple narrative as to how all this is going to be tied together. The proposal should be specified in this manner and the proposal presented to the customer. It should be paid for, and if the customer wishes to continue, he should sign-off at this point that this is what he wants. You should then provide him with an estimate for the hardware and the software to accomplish your proposal. It is obviously possible to include the hardware and software time and costs in the systems study, if that is what is desired.

A note on schedules. It is quite common to underestimate time. Most people can estimate costs better than they can estimate time; so that you must be very careful that you can accomplish the specified tasks in the proposal in the time and hours that you allow. It is not a bad idea to total the times required for the individual tasks in order to get an overall job time. By that I mean it is far better to specify a job piece by piece in terms of trying to determine how much coding there will be and then look at the total at the end, rather than it is to look at a job and say that is a 200 hour job. Most of the time, those "gut feel" estimates are wrong, and it is compounded in many cases by the fact that you are probably undercharging as far as your hourly rate is concerned.

At each point in the process when a particular system piece is completed, you should have specified ahead of time test data that will demonstrate that the programs are working as per specification. Note that I emphasize test data, and that I emphasize agreeing on this ahead of time as part of the programming proposal. The rationale behind this is that you cannot really be responsible for the whole of your customer's information base. You cannot spend enough time to verify the integrity of all his information, and you are depending on previously specifying what the system will accomplish. There is many times a large gap in what the specified system will accomplish and what the customer's hidden agenda is, is related to what he wants to accomplish. For the uneducated, naive first time user of computers, it has been made to look ridiculously easy by much of the advertising and much of the media. So, in fact, he has high expectations where he or she should not. If you want to do a good job for your customer, and stay in business, you have to make sure that everything is done on a realistic basis. For that reason, test data should be specified ahead of time, and when that test data is run,

it should be considered that the job is complete.

Changes are extra, and you should make this clear from the beginning. If the changes result from a customer change in specification, they should be paid for. You should estimate what they are going to cost before you undertake the changes. In order to be fair to the customer, he should know what his change is going to cost him before you start to work on it. Probably more disputes have been caused by the failure to do this and the failure to specify things in writing then anything else in the computer business. I would repeat that software problems with customers come from unrealistic specifications, and from an assumption that the customer knows what it takes to change something. He does not if he is not familiar with computers. He thinks that he can change it the same way you erase something with a pencil. He knows nothing of program logic, and he has no real realization of how long it takes to reformat a report or a file with 10,000 records.

There are some additional problems of which the fledgling systems analyst programmer should be aware.

He should be aware of his customer's company atmosphere. How good are the regular administrative procedures of the company? How much is in writing? Is there discipline, not in the sense of punishment, but in the sense of following established procedures. Are there controls on the activities that are performed in the company or is he winging it on the basis of notes on backs of envelopes? Is there any administrative backup to the proprietor? In some cases, this company atmosphere should give you a clue as to whether you can design a successful system of not.

Hardware

In this context, I am talking about the use of so-called hobby or amateur equipment as opposed to that which comes from the commercial market. The difference is sometimes more in the support and maintenance areas than it is in the actual hardware. However, the so-called hobby systems are plagued by poor support. The small businessman expects to have any type of equipment he buys supported by the people from whom he purchases it. In the micro-computer business, so far that has not been the case. Unless the local computer store is well organized and able to support the customer's equipment, the first electronic failure that he has is going to be a traumatic experience. If somebody tells him to bundle up his CPU and send it back to the factory, and it becomes clear to the entrepreneur that

he is not going to be able to access his inventory or his general ledger until this mysterious entity comes forth with a new or repaired board. He is going to be in a rage. Secondly, if he finds out that there is not a serviceman available to him who can come to his office to fix this, he will probably be displeased. His criteria is probably the electric office typewriter. If he has a failure, the serviceman comes and fixes it, and likewise with his adding machine, or his lathe, or his milling machine, or his stamping press. The micro manufacturers have not yet achieved this maturity.

Another hardware limitation is lack of proper peripherals. In many cases, if you are going to use the system in even a very small business, there will be large requirement for printed reports. A proper high-speed dot-matrix printer is still an expensive item, and in most cases can come to at least 50% of the system. The CPU represents the cheapest part of the system, with some form of mass storage also being a necessity for most small businesses. Of late there seems to be a tendency for most people to include a double or triple floppy in the small business system. I would describe this as the minimum which can be tolerated, and I have severe doubts, even with very small businesses, whether they can last very long on a floppy based system.

A lot of the wonderful stories of success about systems being applied to small businesses come from a technically oriented proprietor who bought the system, using his business as an excuse somewhat in the manner that larger companies buy corporate airplanes and depreciate them at the company's expense. He has done all the programming and is intimately involved with the success of the system. I don't consider this as a true business application of the hobby micro. There are very few companies in terms of the total number of small businesses available in the market that have that capability. It is much more likely that you will run into an entrepreneur with no previous computer training who simply wants a packaged program that can be made to work.

For the foreseeable future we are not going to be CPU limited; we are going to be limited by the mechanical peripherals which are still purchased by the pound. I also have some considerable apprehension about hard disk systems. In many cases now they are being offered as an add-on to S-100 bus systems. The problem is that the disk operating system has not been designed as an integral part of both the drive and the CPU. This can make for some severe operating problems and I will touch on those later when I talk about systems software.

So, in general, we still have cheap CPU power; we con't have a good widely available, cheap, hard copy printer; and we don't have any standardized mass storage media. This situation should change when the electronic mass storage media become available. When either bubbles or CCD devices become widely available it will probably mean a big boost to the hobby-based system. There will be a big decrease in price because the fundamentally electronic devices are priced on a rapidly declining curve which is based on semi-conductor device yields. The electro-mechanical peripherals and storage devices are not subject to this kind of a learning curve. Therefore, their price does not decline as rapidly.

Software

There are probably only two fundamental differences between a micro processor based hobby system and a micro processor based commercial system. They are reliability and the presence of software utilities. Years back, when mini's first emerged, the situation was much the same as we find now with the micro's. We had a "gee-whiz wonderful" technical device that had very little in terms of peripherals support and nothing in terms of software utilities support. These utilities can be described as the tools which are necessary for the applications and systems programmer to use when he creates a system. It is probably not economically viable to write small business systems from scratch without the aid of some powerful programming utilities. These are file managers, report generators, and data based systems. Interestingly enough, there appear to be a couple of companies in the hobby field that have worked out data base management systems for micros and I have seen some advertisements for some of these on a commercial basis. They are still not "poor man" systems, but they are cheaper than the commercial systems and they appear to have considerable capability. It would be my advice for any fledgling systems - analyst - business programmer to get some experience on an equivalent mini based system so that he understands what is available in terms of this type of software. Not only do the micro systems not have this kind of system software, but what they do have is nonstandard. This lack of standardization applies from everything from the version of basic that is used on up through the communications protocols. It is a "Tower of Babel" and the user and the systems analyst and programmer are suffering because of it. There really is not going to be any way to make a good efficient set of programming tools until the industry can get together and determine what standards it is willing to support. Up until that time they will be forced to look for commercial

packages and utilities.

There seems to have developed a thriving business in package software. I have looked at much of this, and I am not impressed. Very little of it was written for the general case; most of it was written for one specific case or another, and it is not very adaptable or transportable because of the lack of standards and systems utilities that I mentioned.

Conclusion

In short, with careful study you can be successful; but, it takes the use of an equal, or greater, dose of caution and business acumen as well as technical knowledge and programming skill.

TABLE 1

NUMBER OF BUSINESSES FILING TAX RETURNS BY LEGAL FORM OF ORGANIZATION, 1973

Total Businesses 13,343,406
Corporations (1972) 1,823,335
"Subchapter S" Corporations . . . .
Partnerships 1,037,911
Proprietorships 10,482,160
Source: Preliminary Statistics of the Internal Revenue Service, 1973

AMERICAN BUSINESS ENTERPRISES

Over 20 million Small Businesses, 30,000 medium-size firms and the Fortune 1500 provide goods and services for our free enterprise economy. 17 million are sole proprietorships, 1 million are Partnerships (with an average of 3 partners each) and almost 2 million are Corporations. The following figures were compiled from I.R.S. Statistics of Income - 1968. "Net Profits" (less Losses) include "wages" to Proprietors (Sched. C) and Property Owners (Sched. D) and are before federal income tax except for incorporated businesses. About 1/3 of U.S. Businesses showed Net Losses. Almost 700,000 new businesses are started each year, while 500,000 go out of business.

	Number of Businesses (thousands)	Gross Receipts (millions)	Net Profits minus losses (millions)	Average Profit (dollars)
TOTAL ALL BUSINESSES	18,229	N.A.	$131,824	$7,1C0
Rental Property Owners	6,046	N.A.	2,419	400
Inventors & Explorers(Royalties)	506	N.A.	805	1,594
TOTAL - Mfg.,Commerce,Services, Construction,Transport & Farms	11,677	$1,800,218	$128,600	$11,000
Agriculture,forestry,fish	3,070	47,277	3,564	1,200
Agricultural services	288	5,155	902	3,131
Mining	68	16,708	1,749	25,720
Contract Construction	840	99,029	5,375	6,398
Manufacturing	397	652,966	45,339	114,200
Food & kindered products	28	43,655	4,247	151,678
Textile & apparel	35	45,435	2,081	59,500
Lumber & wood products	72	40,825	2,864	39,800
Printing & publishing	59	24,416	2,073	35,100
Chemicals	125	52,348	7,985	63,900
Fabricated metal products	35	37,608	2,535	55,100
Machinery (non-electrical)	46	50,847	4,829	105,000
Electrical Machinery	13	46,854	3,196	245,900
Rubber,leather,stone	16	33,571	2,241	140,100
Petroleum refining	1	66,096	4,138	4,138,000
Primary Metal Industries	4	43,389	2,068	517,000
Motor Vehicles	2	58,179	5,614	2,807,000
Other Transportation equip.	3	34,992	1,486	495,300
Misc. Manufacturing	15	22,215	2,243	149,500
Transportation & Utilities	367	119,307	11,538	31,400
Local Transportation	104	51,570	1,286	12,400
Trucking & Warehousing	208	5,501	745	3,581
Communication	10	27,752	4,953	495,300
Electric,Gas & Sanitary	22	33,573	4,468	203,100
Retail-Wholesale trade	2,592	591,099	20,166	7,800
Wholesale trade	453	238,113	6,737	14,900
Building materials	96	20,826	815	8,500
General merchandise	337	57,295	2,656	7,900
Food stores	286	74,934	1,881	6,600
Auto dealers	373	91,618	2,284	6,100
Gasoline stations	228	20,434	1,089	4,800
Apparel stores	103	18,168	1,021	9,900
Furniture	112	16,854	723	6,500
Household appliances	43	2,752	226	5,300
Eating places	274	21,696	1,054	3,800
Drinking places	114	3,856	419	3,700
Misc. Stores	388	39,772	2,087	5,400
Real Estate, Ins., Finance	1,223	165,414	18,743	15,300
Insurance	224	77,248	4,917	22,000
Real estate	795	26,750	2,825	3,600
Services	2,796	102,959	21,425	7,700
Hotels,Motels,Trailer Parks & Camps	211	9,109	599	2,900
Personal services	618	13,038	2,046	3,300
Laundry & dry cleaning	91	1,840	313	3,400
Beauty shops	215	1,852	485	2,300
Barber shops	126	953	458	3,600
Business services	401	22,153	1,764	4,400
Auto repair & services	200	9,588	788	4,000
Appliance & other repairs	196	2,235	526	2,700
Amusement,recreation & theatrical	204	9,955	577	2,800
Doctors,Dentists,Nurses & Health	434	17,172	9,255	21,300
Legal services	150	6,663	3,530	23,500
Educational services	97	459	137	1,400
Engineering & Architects	67	2,499	632	9,400
Accounting & Bookkeeping	130	2,780	1,043	8,000
Other services	161	8,646	708	4,400

Prepared by the AMERICAN FEDERATION OF SMALL BUSINESS
407 S. Dearborn St. • Chicago, Ill. 60605 • Telephone (312) 427-0206

BUDGETING FOR MAINTENANCE--THE HIDDEN ICEBERG

Wm. J. Schenker, M.D.
Medical Information Systems
2086 Essenay Avenue
Walnut Creek, CA, 94596
(415) 939-6295

Abstract

Earmarking capital for maintenance is
one of the things which sets a business
or commercial venture apart from a
typical hobbyist's activity. These cost
factors can be dealt with in two ways,
as a science and as an art. The former
is the most visible, providing subject
matter for textbooks, seminars, and
college credit. Complex as it is, it is
still based on the simple linear premise
that 2+2 really does equal 4, and such
like. This makes it relatively easy to
read and write about.

On the other hand, what the science
deftly avoids is a large object whose
icy tip spells disaster in the deep for
the unsuspecting businessman or
professional. Etched in large letters
on the submerged surface, hidden to
ordinary view, is the warning, "Murphy's
Law (and Cohen's Corollary) Reigns Ever
Supreme !"

It is this murky subject, the center
of many a hallway conversation among
insiders, and rarely discussed as the
computer systems vendor plies his trade
among the innocent, that will be
emphasized in this paper.

I. Introduction

EXHIBIT A. There is a medical clinic
which has had considerable experience
with computers in the last decade.
Three years ago, when operational
microcomputers were as rare and
expensive as hen's teeth it had the good
fortune to be offered on loan from a
local government scientific organization
a complete micro system of the highest
caliber. Here was a chance to evaluate
this new technology in a medical
environment and for free. The clinic
chief, wise in the ways of computer
vagaries, when approached with this
offer responded, "Sure we'd love to have
use of such a system - but only if
you'll pay the maintenance costs."

To the hobbyist, so much in evidence
here at the Computer Faire today, system
failure is a challenge which promises at
worst a broadening of knowledge, at its
best the ego rush of successful debug
and repair of a non-working
conglomeration of wires, chips and
boards. "1 did it, and 1 don't even
have an electronics background." This
challenge is not constrained by time
factors nor by responsibility to an
outsider such as your customer. Since
time is money you can see how hobbyists
can spend $10,000 in labor to repair a
computer costing $1000.

To the businessman or professional
however, equipment breakdown means at
best an added expense and at worst lost
income and a blemish on the
organization's market image. To
appreciate the significance of this
consider the following. When a computer
system suddenly stops running, or "goes
down" or "crashes" in the vernacular,
the obvious cost is loss of service to
the customer. What is probably in the
long run a much more telling loss is the
loss of data base integrity. This
occurs when the transaction in process
at the time of crash gets lost (or
duplicated!), a record during update is
lost, or a spurious record pointer
change occurs. This last can result in
possible loss of a massive number of
records. The bottom line effect of all
this on your business is loss of
customer confidence.

Let's look then at what must be done
to maintain uninterrupted performance to
your customer or client at the level the
latter's accustomed to or contracted
for. And some idea of what these
actions will cost in the way of capital
investment and added payroll.

These questions are important even
for a computer "application" or
assignment which is only periodic in
nature, such as getting a payroll out
every two weeks. But they can loom
large enough to become a pivotal factor
in the organization's overall success if
the computer's output needs to be close
to continous.

EXHIBIT B. A centralized medical
boratory is planning a program which
11 process up to 20,000 tests a day
d return the results via computer link
to outlying clinics the same day; any
breakdown in such a system for more than
a few minutes could create a backlog
forcing delay of some results over to
the next day. This would be considered
intolerable in many cases by the doctors
waiting for results.

Thus the problem can be seen to warrant close study and inspired application. Indeed so much that journal articles and textbook chapters are dedicated to it. Now this literature tends to follow the formula of computer science publications in general. The orientation is extremely rational in tone and the reader is assumed of a likewise bent. The people in this field tend to have a heavy background in or orientation to mathematics. Thus you will find the literature also based heavily on a mathematical or statistical approach worthy of the physical sciences. At the heart of it all is the reasonable assumption that 2 + 2 really does = 4, and such like. As a matter of fact the keyword here is reasonableness. It typifies the standard approach to maintenance strategy, the science of systems maintenance, and budgeting for maintenance.

Contrasting this approach is that ill-defined, scantily documented, and trivially regarded collection of anecdotal material and opinions best summed up as the art of systems maintenance and budgeting. It is this aspect of small systems or microcomputer technology that will be emphasized in this paper.

II. Some Standard Recommendations

A superficial appraisal of maintenance would focus on the obvious, the specs of the warranty and maintenance contract. The important facets of the latter include: location of service depot, minor parts or all parts covered?, charge for travel time?, preventative maintenance schedule included?, and is service agreement on an hourly ("on call") or contract basis?

However, experience dictates considering as well, the design and configuration of the system itself. Because these decisions made long before system purchase impact so heavily on subsequent problems, one must focus as well on these factors. Accordingly, note the following brief but representative list of points that an end-user would be advised to investigate prior to purchase.

1. Buy from one vendor. A mixed-vendor potpourri will find inter-vendor finger pointing at the time of breakdown, each one accusing the other as the basic culprit.

2. Vendor pedigree. The vendor should be big-name and well established, even tho the initial price is much higher. Avoid the fly-by-night and those without a track record.

3. Vendor "burn-in". Burn-in is the process of pushing the hardware close to its limits to see if it will stand up to prolonged temperature, vibration, humidity, dust, and electrical noise stresses. Investigate the details of this procedure by the vendor of your choice.

4. MTBF & MTTR statistics. MTBF is the mean time between failures, and MTTR is the mean time to repair such failures. Ask the vendor to show you his figures.

5. Vendor warranty. Investigate carefully the vendor warranty details. Consider this an important step.

6. Source of maintenance contract. Buy your maintenance contract preferably from the system vendor, or as a second choice, from one of the nationally-known service companies.

7. Track record. Buy a system that's been out in the field with a long, reliable track record.

8. Size of system. Buy a system larger and more powerful than necessary for the application at hand - to allow for subsequent expansion of the applications environment, thru the use of "multitasking" software. (This technique, allowing 2 or more jobs to be handled by one machine at times apparently simultaneously will be discussed in more detail later.) By planning ahead in this manner you can save on maintenance in the long run, by avoiding complex and unreliable retrofits, kluges and mismatches. A "larger than necessary" system here could mean a minicomputer, even tho the price is 3 to 10 times that of a micro system.

9. "Diagnostics". Buy all the available software packages of this genre. They allow you to test out your hardware in a routine and thorough manner, often enabling you to anticipate system failures before they cause a total "crash".

10. IC chips soldered in. Buy systems with the chips soldered, not socketed

in. The latter technique is a source of intermittent failures.

11. Mass storage peripherals. First choice for small systems is the "floppy" or flexible disc, with digital cassette or 3M cartridge devices for backup.

12. Memory. First choice is high density dynamic RAM, because of low power and chip count - therefore higher reliability.

13. Printer speed. In general buy the fastest printer you can reasonably afford. (By the way, low speed in the big-name world means about 200 characters per second or somewhat less than 200 lines per minute. Medium speed is about 600, and high speed goes up to an astronomical 30,000!)

14. Install an UPS. This means an uninterruptable power supply. It should be large enough to handle the power requirements of your entire computer system.

15. Ambient temperature. Keep your computer system cool; use adequate convection in the form of fans inside the chassis, and air conditioning in the room.

16. Nicotine. Avoid smoking in the computer room - the fumes are poison to magnetic disc and tape media.

17. AC power lines and grounding. Use good filtering and solid grounding procedures, respectively.

18. Modular vs all-in-one packaging. It's OK to buy the latter - it cuts down on troublemaking interconnect cables, dust, meddling by unauthorized personnel, and generally makes for a neater appearance.

19. Front panel. Your system should include this item. It's handy in troubleshooting (and incidentally helps in software debugging).

20. TLC during infancy. There is another factor, not apparent to the newcomer in the field of EDP (electronic data procesing), which should be considered for its impact on maintenance costs. Altho a system should operate in flawless fashion the first time it's powered up - that almost never happens. A computer system, like people, needs lots of tender loving care during

infancy to ensure getting started on the right foot. Lacking this care, troubles will hound the system possibly to its grave. So until this stage is reached you can't consider visits by service personnel as part of maintenance costs and loss. When the system is completely debugged and performing to vendor specs for say a month, subsequent breakdowns are then properly in the category of "downtime", or failure.

III. Observations on The Foregoing Plus Some Maverick Recommendations of my own

1. Single vendor systems. In the first place when you're dealing in small systems never buy from ANY vendor. Buy instead from your local retail computer shop. People you can talk to on a first name basis and whom you're likely to bump into at your neighborhood supermarket are much more likely to be responsive to your needs. In the second place if you buy what are called S-100 products, you can count on a relatively high degree of inter-vendor compatability. More on that later.

2. "Major league" vendors. There are no big name vendors whose primary business is small systems. Avoid the minicomputer firms and the big semiconductor manufacturers for whom micro systems are only a sideline - they're too big to care about you.
  Besides which, the "biggies" don't necessarily use the latest technology (because they feel somewhat immune to market pressures?). For example, the best support chip technology to day is Low Power Schottky (LS, for short). It is more reliable than its predecessor, TTL, because it produces less heat and electrical noise.
  EXHIBIT D. DEC's PDP8 uses TTL support chips throughout.
  EXHIBIT E. Persci's floppy uses TTL support chips throughout.
  EXHIBIT F. NLS (a biggy in test and medical equipment) in their new model clinical lab unit uses RTL throughout. This technology, even older than TTL, was already vintage in 1974, when I first got into personal computers.

3. Vendor burn-in. Temper the standard advice with the fact that the modern chip technology described just previously tends to be quite reliable once you're past the "infant mortality" stage. (This is chip failure within the first several hours of use.) If you do

feel you need it for your application environment don't rely on the micro vendors. They do very little of it. Instead have the local retail shop where you buy your system do it for you.

4. Ah, the beauty of those neat MTBF and MTTR statistics! So crisp, so scientific, so neat, so precise. So valuable?

At the outset the one thing to keep uppermost in your mind is that these MEAN figures apply only to Mr. MEAN End-user. He sits right in the middle of the bell curve of probability. Way off to one side of him is the fellow whose equipment never breaks down. But off to the other side is the fellow whose equipment fails before he even gets it out of the packing carton. Now no vendor will ever guarantee that you'll always fall between Mr. Mean and Mr. Superlucky - but many of their salesman will. But not in writing.

To put it another way if you need to rely on 5 years between breakdowns you can't pick a system with a MTBF of 5 years (even if such were available). You'd need a system with a MTBF of FIFTY years. To ignore this unpleasantry is courting a case of the "pre-demo blues" and full-blown operation of Murphy's law.

EXHIBIT Fa. A large Bay Area service organization had planned for close to a year in 1976 a demo for top echelon management of a complex multi-station, multi-tasked system. As D-day approached everything began to fall into place, a tribute to the careful planning, competence and experience of the EDP (Electronic Data Processing) staff. By a week before the demo everything was pretty well sown up. Confidence was building for a successful performance on which would be pinned the hopes for the new budget. Confidence increased further in the last few days as everything continued to fall together right up to the eve of the big day. THEN the system crashed, too late to pick up the pieces in time for the visiting dignitaries, who witnessed a limping, anemic ghost of what was once a vibrant template for future glory.

EXHIBITS Fb, c, and d. Three similarly painful experiences, on a smaller scale, with the first of my two systems in the past year.

By the way, have you ever noticed something peculiar about luck, that phenomenon which is the basic reference point for all statistical validation - except the statistician calls it

"chance"? Anyway have you noticed how so much of life demonstrates that the lucky get luckier and the .... you know the rest?

This may seem like rambling off the subject, but I'm reminded of how when I pull up to a toll gate pay station (or a supermarket checkout counter) I always look for the quickest line - and usually get the slowest. What bears mention is that altho I figured I must be a rare one, it's surprising how many others turn out to be members of the same club.

And sort of inversely related to this is how a fellow intern back in the '50s made more money at the end of every month than he got in hospital salary ($125!) - by beating us all at poker for 12 months straight.

5. Vendor warranties. Forget it, in the micro field. They're worthless in terms of turnaround time (up to three months) - another reason to buy local. (See the following.)

6. Maintenance contracts. The micro vendors don't offer them, and the national service companies don't have the necessary experience with the small systems. Again buy local, including your maintenance contract.

7. Products with long track record. Forget it. The entire market is only two years old.

8. Buy a larger-than-necessary system? Don't, don't, don't. Buy the minimum you need for the application at hand. Discovered another application later? Buy another minimum system. For each added major application add another stand alone system, as close to identical as possible in hardware and software to your previous systems.

Consider The Alternative. With today's labor costs in the form of programmer's wages, custom software implementation of the multitasking you'll need can cost 5 to 10 times the equivalent in hardware.

More on Multitasking. Just to ensure proper understanding of this point a moment's digression into some details. This technique allowing two or more applications to be handled by one machine, at times apparently simultaneously, actually is a process akin to juggling 87 balls in the air at once - the computer's software

interweaving the multiple applications or "jobs" one between the other. The technique harks back to the days when the only hardware available were maxis and minis which cost so much per system that you were easily persuaded to squeeze out the maximum performance per piece of hardware bought.

In fullblown versions used on the biggies it involves developing techniques for "queing" one job after another according to priorities, error checking of a complex nature, and complicated "rollback and recovery" of data when the system eventually crashes. It is responsible for a large software "overhead", i.e., software which is not earning you any money, while using up your computer's resources, both memory and processing speed or "throughput". Furthermore this is the kind of software that can take man years to develop and therefore costs plenty. It also helps to make the science of software development mysterious and their practicioners irreplaceable. Further, one would have expected by now something that complex and expensive would have made the large central computer systems "bullet proof" or impervious to error. It is certainly in part responsible for the somewhat unsavory reputation that computers have earned in non EDP circles over the past 20 years.

EXHIBIT G. In July 1977 I applied for a Sears credit card. When it arrived it showed a purchase made in mid-June, 2 weeks before I ever saw the card. So I wrote to the special place you write to at Sears when there's any problem with your card. Well, they've been dunning me ever since, the monthly finance charge ever increasing. My next move is to cut the card up in small pieces, staple it all together with my latest bill and ship it off to Mr. Roebuck.

EXHIBIT H. BART – spells Bay Area Rapid Transit, San Francisco's new train system. It also spells fiasco in connection with its originally designed computer control system.

EXHIBIT I. Social Security. One of the computer scientists I work with knows of at least eight other SS card holders with his number.

EXHIBIT J. Fill in your own favorite.

9. Vendor diagnostics. There's hardly any available so have your local retailer write it for you.

10. Soldered or socketed ICs. The need

varies with local climatic and other factors – let you local shop make this decision.

11. Mass storage peripherals. First choice is not floppies but MINI floppies, in particular North Star. For backup don't even consider digital cassette or 3M cartridge. Instead buy another North Star – it's in the same price bracket. If mini floppies don't meet your memory capacity needs bypass the full size floppy and go right to fixed discs. You'll thank me.

How can 1 make such a recommendation in the face of the floppy's popularity? Well, the full-size floppy mechanicals are very tricky to align before you can get rock solid reliability. Then in about 6 months you may need it again. The mini floppies on the other hand have different physical dimensions (less inertia?) which make their tuneup easier to obtain and maintain. Where do I get this information? From end users, not from magazine articles.

12. RAM memory preference. Spec your system to avoid dynamic RAM – it's too flaky with high speed peripherals using DMA (direct memory access), such as floppies and some graphics displays. Order instead FULLY static boards. Also avoid super high density boards such as 64K or 32K boards. Spec your system in increments of 16K. That way, if a chip goes bad on one board in a 64K system you can still operate in a degraded mode with the other three.

13. Printer speed. In general 1 recommend the opposite: buy the slowest printer you can get by with. (What you can get by with may surprise you – as will be described later.) Slower printers tend to stand up longer. Remember: "Speed kills."

14. Install an UPS.
EXHIBIT K. Somebody accidentally trips on the line cord to the computer and pulls the plug.

EXHIBIT L. Margaret, your secretary, is freezing because you won't turn up the thermostat to 80 degrees, so she quietly brings to work one day her 12 Amp portable space heater and plugs it into the same 15 Amp circuit your 3 Amp computer is on. Everything's fine from 9 to 930 AM with all transactions uneventfully processed and stored in RAM, at which time the program turns on your 3 Amp printer to get some hard copy and -- presto! -- the circuit breaker

pops and so does all your data from 9 to 930 AM. Or worse yet, your disc operating software gets bombed, too.

15. **Keep cool.** Yes.

16. **Avoid nicotine.** Yes, it prevents computer cancer.

17. **Watch AC lines and ground well.** Yes.

18. **All-in-one packaging of system.** - INSTANT DEATH from a maintenance point of view. Go modular all the way. This point is so important that it will be covered in detail later.

19. **Front panels.** Avoid them like the plague. The only one who might want one is your maintenance man, and he'll leave it in his toolbox on many jobs.

20. **TLC during infancy.** Amen. Another reason to have service personnel close by, which means your local computer store, again.

IV. **Even Stranger Recommendations.**

1. **"On Line" And "Real Time".** Computer people bandy about two terms you should become familiar with, on line and real time. There are many definitions extant, but all you need remember is that some applications allow the computer to work in spurts, with long pauses for resuscitation in between - and at the other end of the scale some applications require the computer working 24 hours a day, 7 days a week with nary a skipped heart beat. Now the closer your application is to this continuous type of affair the more on line or real time it'll be considered.
   A piece of advice. If you're planning an application which could classify as pretty much on line or real time -- STOP AND RECONSIDER.

2. **"Non Stop".** But suppose your application calls for nothing less than the extreme, a continous run? It's then logically enough, labeled non stop.
   More advice. If you're planning this kind of an application -- STOP AND DON'T RECONSIDER.
   Unless your retail shop can configure your hardware and write enough software to give you the micro systems equivalent of what Tandem Computers, Inc. claims in their ads they can do with their large system. (If you're interested they're at 20605 Valley Green Drive, Cupertino, CA, 95014.) Their system is what is called "multiple redundant".

3. **The Nitty Gritty.** Which brings us into the nitty gritty of my unorthodox approach to budgeting for small systems maintenance. My prescription for the typical business application which, altho not non-stop does have significant deadlines to meet, is simple.
   BUY TWO OF EVERYTHING. Two complete systems. Two computers, two sets of identical software, two sets of dual mini floppies, two backup storage devices (mini floppies again?), two keyboards, two video monitors, two printers, and lastly two complete sets of interconnect cables. (Ignore this last item and the whole deal is off.)

This strategy has three things going for it. The first is the ability to keep operating during a critical phase of activity when one computer crashes. True you lose the data that was in transaction and you can lose records, but with proper mass storage backup that barb can be dulled. Just flip the

switch of the other system and transfer your work to it. (If you get your retail store to write some software and add some minimal hardware you can get a semi-automated transition from one system to the other.) You've spent twice as much in capital outlay in exchange for almost instant repair service, unobtainable any other way at any price.

The second advantage has to do with that low speed printer I recommended earlier. By using your backup computer that is otherwise idle, to drive your printer you can overlap your application functions in time. For example you can be talking to one computer via its keyboard while the other is printing out your three hour report, but you could care les. In this manner you can usually do quite well with a 15 CPS (characters per second) printer where a 60 CPS would be considered the bare minimum, or a 30 instead of a 120, etc. (This same strategy can be considered if you're trying to get by with a low speed mass storage peripheral, the audio cassette, which can be quite reliable if properly set up.)

The third and most important advantage I save for last. It is the key element in a novel concept of computer technology, geared to match the microcomputer's role in the increasingly popular EDP trend towards "distributed intelligence" thru "distributed processing". What these imposing

phrases really mean is that instead of relying on one large computer to do all your work, you spread out some of this work by using a network of small computers scattered around the hinterlands.

In effect you trust your eggs to more than one basket. This trend is having a salutory effect on the whole industry, making systems less vulnerable to "total crash". When the big one goes down people out in the boondocks can still do some work while waiting for the system to come back up again. It's cheaper. It's also less complex and mysterious. (The enormity of the software problems associated with one computer doing everything was referred to earlier.)

Distributed Maintenance. I recommend we start doing the same thing, now, with the maintenance process. In a phrase we need what I like to call DISTRIBUTED MAINTENANCE. Spelled out this means that instead of relying on a central repair source (the vendor, or a national repair organization) we get this service out into the field as close as possible geographically and timewise to the one who signs the bottom line, you the end user.

EXHIBIT M. An excellent example of this kind of thing in actual practice is described by a resourceful Canadian, Jean Francois, in connection with a large minicomputer system he runs for the Ministry of State for Urban Affairs in Ottawa, published in DATAMATION, August 1977. This article should be must reading for you.

But let's suppose you can't see your way clear to buying 2 of everything. Then with one system at your disposal your best bet would be to sign up with your local computer store for as close to complete coverage as you can afford (and the store can provide). The contract specs to check apart from the usual are: what is the guaranteed time to arrive, and what about nites, weekends, and holidays? Expect to pay from 10% to 30% of the system's purchase price annually, depending on whether you get minimum or "total" time coverage. With that kind of annual cost facing you, my proposal of 2 of everything seems less outlandish.

Distributed Maintenance Protocol.

Setting The Stage. The success of this strategy will depend on your local computer store for three things.
1. A contract for repair of the defective equipment you track down with this method. It should specify the maximum "turnaround time" you think you're application can tolerate. One to three weeks will usually do for typical applications. (You'll save a bundle right there.)
2. Developing software, in the form of a short diagnostic package, that will tell you when in the course of your testing you've in fact bumped up against the troublemaker and have the system runnable again. This message will ordinarily be in the form of video screen prompts.
3. Installation of an electronically simple yet very effective monitor that tells you if your power supply secondary voltage outputs (usually 3) are in good health. This will be in the form of little red pilot lights set into your (otherwise blank!) front panel. Or they can be installed inside the computer

cabinet away from the high voltage end of your power supply, which should have a protective barrier placed over it by the shop personnel. This is so you can't monkey with it accidentally or on purpose. Using this last arrangement the lights should be in clear view on lifting off the computer cover.

The importance of these monitor lights is this. If any of them are out it meansyou've lost one of the voltages; you should PROCEED NO FURTHER with subsequent tests, but get your maintenance man on the phone. Fortunately this will be a rare occurrence .

First Phase. Swap modules or subsystems, one by one, FROM the known good TO the crashed system. Be sure the power is off both systems as you're making each swap. Off how long? Long enough for the capacitors to discharge = when the blower fans stop rotating. Do this until you find the trouble. Here's a good sequence to follow: interconnect cables, mass storage device, video monitor, keyboard, printer, and finally the computer itself. By now you'll have found out which module isthe problem. If it's any but the computer it's a job for your retail store. If it is the computer go on to the second phase.

Second Phase. Remove the cover from each of the computers. Start swapping boards, one by one, in the same manner you did with the modules previously (Remember, power off!). A good sequence to follow: memory boards, mass storage interface board, I/O board, and finally your CPU card or board.

Simple Procedure. Remember, don't make this a complicated procedure - do it "by the numbers", preferably written down on a large cardboard placard placed on the wall near your system. Teach yourself the technique first, then you secretary, nurse, bookkeeper, office boy, or Girl Friday. A maximum of maybe 1/2 hour using almost no technical expertise, is all you'll need to track down most troubles that can arise. You can then in a more leisurely fashion, send the defective piece of equipment to your retail shop where routine (and thus less costly) repair at the component and IC chip level can be performed.

Equivalent Performance. To achieve the equivalent performance in terms of ultra short downtime and repair of defects would require keeping a full

time computer tech on your premises (and payroll), AND he would have to have available a full set of replacement parts to achieve his goal. That salary in today's market is $15,000 and up a year. Then add the equivalent of "2 of everything" anyhow.

Cheaper in The Long Run. Distributed maintenance, made possible by modern technology, will move computer science considerably closer to the kind of performance you the businessman or professional, expected to get in the first place. This is the kind of performance which spells business or professional success, which translates to more income. To boot, distributed maintenance is cheaper in the long run than any other method.

No Fancy Test Equipment. While on the subject of cheaper let's cover a related point. Don't be talked into buying fancy and expensive products that enable you to troubleshoot like the pros do it. That's NOT what distributed maintenance is all about -- it's about non-electronic people using their time to get on with their own profession. So don't buy a scope, a logic probe, or a logic analyzer. And don't get into swapping IC chips either - you could blow a good one after a bad one that way.

4. Maintenance in a small town. With all this reference to relying on your local retail computer store, what if you're in a town with no such source available? Well in that case what might otherwise appear to you as a luxury, distributed maintenance, becomes a stark necessity.

One Proviso. Even with 2 of everything don't even consider a near-non stop application if you're located in places like Last Chance, Kansas or Winnemucca, Nevada. There you'll need THREE of everything. Also you'd better consider making a special (and costly) contract with the nearest retail store with provisions for (a) the technical personnel remaining on site until the system's thru its infancy and TLC period, and (b) paying them only 1/2 the total purchase price on delivery, the other half after certain clear-cut, mutually agreed upon tests can be passed by the system's operation. If you can't get this kind of arrangement consider (a) foregoing the pleasures of rural life and moving to or near an ugly big

city or (b) running your business as before, in the manual mode - and buying a cheap computer for use at home and calling it a hobby.

5. The weirdest recommendation of all. This one's saved for last, since it's so obviously beyond the pale in our culture where rationality is considered the final criterion of any scientific endeavor.

Right at the outset of systems planning, before you even look at your application needs, step back and ask yourself this question, and then answer it as honestly as you can. "Am I (a) consistently lucky in business or technical ventures, (b) lucky as often as unlucky, or (c) consistently unlucky?"

If the answer is (a) then much of my ramblings can be ignored. If it's (b) it'll pay you to reflect on them. If it's (c) you're in the same boat I'm in, and to ignore my warnings augers well to bring you deep grief and near insanity in the form of slipped schedules and broken promises. (Take heart tho in the maxim, "Unlucky in technology, lucky in love!")

V. Summary

This paper has emphasized the more quirksome aspects of small systems maintenance problems. The major points made were:

1. Cost-effective maintenance decisions depend heavily for their success on making the correct choices long before the system goes down. As a matter of fact, it is at the time of original spec'ing out the system to be purchased that most of the die is cast.
2. The single most important aspect of the system specs includes
    a. depending on purchase of 2 of every piece of equipment, thus allowing
    b. the full exercise of the distributed maintenance concept.
    Spelled out for small systems it means doing in-house swapping of interconnects, modules, and boards, backed up by a firm contract with a local reputable computer retail store for actual repair of the faulty equipment.

VI. How to Evaluate This Paper

After reading this far if you're a businessman or professional concerned about how he spends his dollars, you should be asking the question, "Sure

this presentation is witty and provocative, but does he know what he's talking about?" And if you ask this question you've got to ask the next question, "Who can I ask to get the answer to my first question?"

The Experts. So you're left to turn to the "experts" and the "authorities" in the field. Several things to remember about these fellows however.
1. The field is so new there hasn't really yet developed a large cadre of knowledgeable people.
2. Then there are those experts whose expertize is in the mini and maxi computer field. Therefore they're likely to give you the standard list of recommendations that are valid in their world. Ipso facto, most of my unconventional recommendations will be "thumbs down" for them.
3. Of those real experts we're looking for there are 2 general categories:
    a. The ones who've decided to capitalize on their know-how - they become vendors and computer store proprietors (I'm fortunately acquainted with some real honest ones, but you've got to be knowledgeable in the first place to recognized this honesty.)
    b. The other kind who are hard to find. They're usually not in the market place selling their know-how but busy

in their labs having fun. You won't find them in the yellow pages.

The 87-foot monster. But wait, there's more yet. Filling this void then is the 87-foot monster I've been making the butt of my argument throughout this paper, the vendor salesmen. He's dangerous for you computer system's welfare, because -
1. He's more accessible - you'll run into him everywhere: at the trade shows, on TV and Radio commercials, and in the slick business magazines.
2. He's got less scruples.
3. Mathematical and statistical "facts" are great tools in his hands to blur your vision of the nitty gritty you should be appraising instead.

The Saving Grace. With all the caveats just mentioned where then can you turn? The answer is actually pretty straight forward. Go to your local retail store and ask for a list of the purchasers of 10 or 12 complete business or professional systems they've installed prior to 6 months ago, and are presently serving on a full maintenance basis. If they haven't got that many to show you

chances are you shouldn't be doing business with them - they haven't got the experience. Customers more recent than 6 months don't bother with - that doesn't give Murphy's Law long enough to rear its ugly head.
    Then make up a short polite note saying you'd like to talk by phone to them for 3-5 minutes the following week to inquire about maintenance experience and costs with their system. When you're done making these dozen calls you'll know whether I know what I'm talking about.

VI. Cohen's Corollary

I've enjoyed writing this paper and hope you have enjoyed hearing or reading it. But the alert among you may have noted there's still one item mentioned in my introduction that has yet to be laid to rest, namely the enunciation of Cohen's Corollary to Murphy's Law. You can't have heard of it before, because this is its first public proclamation. It goes like this. "When you've taken that very last precaution possible to prevent Murphy's Law from operating in your applications environment -- THAT'S when it probably will."

MICROCOMPUTER APPLICATIONS IN BUSINESS: POSSIBILITIES AND LIMITATIONS

Gene Murrow, President
Computer Power & Light, 12321 Ventura Blvd., Studio City, CA 91604

Abstract

Computer Power & Light, Inc. has been installing micro- computer systems in businesses since Fall of '76. This presentation describes four aspects of our experiences: a short description of the hardware we use; some actual case histories of customers for whom we've provided business systems; some of the jobs we've turned down (which is almost as illuminating as some of the jobs we've accepted); and fourth, some of what we consider to be the important, but often overlooked aspects of micro-computing in business applications.

Introduction

It seems to me that in the heady atmosphere of the growth of this new industry, a lot of claims are being made that are somewhat fantastic. We hear phrases like "what you can do with these are only limited by your imagination", at worst, to the no less extravagant, but potentially deceptive, claims that you can "do your payroll" with a $600.00 computer. Computer Power and Light, as a company, has had to face the music. We have had the customers coming in, magazines rolled up in their hands saying, "I want a computer that will do all these wonderful things." We've had to educate this customer, sell him a machine when appropriate, and then make sure that it worked and kept working. That's the basis of the experience that I'm going to relate.

Hardware

The hardware we work with is our own Compal-80 microcomputer. Compal is a contraction of Computer Power & Light. The machine is an 8080 based micro. It has a serial interface for devices like Xerox Diablo printers, DECwriters, modems, and others. It has a video display of 16 lines by 64 characters, an operating system on ROM, an anywhere from 32 to 56K of memory. We also incorporate the Micropolis dual mini-

floppy disk drive. This is a high density drive; each diskette holds 315K bytes or characters of information, which is approximately 150 single spaced typewritten pages. We use either the Diablo daisywheel printer for our word processing applications or the Texas Instruments 810 printer for business applications. This is not a mini-computer, it is a micro- computer, and as such has all the price advantages and speed disadvantages that micro- computers have. A 56K system with the Diablo printer costs $8,605. A similar system with the TI 810 matrix printer costs $7,300. Of course there's lots of good hardware to be found. It's the software and the support and the other things that get tricky.

Word Processing

Now for some of the applications with which we have been successful and that you might consider as possibilities for a micro- computer system in business. The first application is word processing. It seemed like a natural to us. There's no number crunching really so the speed of some of the single chip micro- processors wouldn't be a factor. Our word processor, which is written to our own specifications, is written in the machine language of the 8080 and is consequently very fast. It is used to create business letters, assemble long documents from boiler plate material, retype multiple revisions automatically, send the same "original" letter to each of hundreds of names on a mailing list, and index archived documents. It has essentially three functions: an edit function, a print function and a storage and retrieval function. In the edit mode you can enter text in a normal way on the typewriter, scroll text on the screen, search and replace a word or phrase throughout a text, "cut and paste" pieces of text (re-arrange blocks of text), bring in boiler plate paragraphs that are stored in a diskette library to assemble long

documents from boiler plate material, retype multiple revisions automatically, send the same "original" letter to each of hundreds of names on a mailing list, and index archived documents. It has essentially three functions: an edit function, a print function and a storage and retrieval function. In the edit mode you can enter text in a normal way on the typewriter, scroll text on the screen, search and replace a word or phrase throughout a text, "cut and paste" pieces of text (re-arrange blocks of text), bring in boiler plate paragraphs that are stored in a diskette library to assemble long documents and things like that. In the print mode, you can set up your margins, your spacing, whether or not you want the right margin justified, you can have four or five different margin formats going at the same time, you can have variable character spacing, variable page lengths, variable line lengths and you can mix these up anyway you like, throughout a document. Finally, in the storage and retrieval mode, you can take a document, whether it's a full report or a paragraph or whatever, and store it under a name that's up to ten characters long, and then retrieve it just by typing that name on the keyboard.

So that's what we're doing with word processing. We had one eye on the Vydek and Lexitron systems when we designed it and the other eye on the pricetag. Our word processing systems start at about $6,0^0.00, which gives all of us micro- computer people a tremendous advantage over the so-called blind or non-video systems, such as the IBM mag-card systems or the Xerox 800 system. And, we're about half the price of the large video oriented systems. Most importantly, the WORDPAL (as we call it) is just a <u>program</u> which runs on the general purpose <u>micro</u>.

Secretarial Services. Who's using WORDPAL? Well, out in Van Nuys, there's a secretarial service called P & S Services. This was the first computer system we've ever sold to a woman. Harriet Wright became very knowledgeable about computers and looked all around before coming in. She told us the plight of secretarial services. They have very demanding customers who want the document today. If they wanted it tomorrow, they would bring it in tomorrow; that old story. The work has to be done rapidly and accurately. If the service isn't

accurate the customer would type documents himself. Oftentimes they'll work all day on a document only to have the guy come back with the thing with 500 different modifications that he wants. Or he wants all of the dates changed or something like that. So they need a flexibility for multiple revisions. Also, P & S needs to maintain steady customers who expect a consistent quality from her and consistent number of formats for the type of document they're going to be getting from her, whether they're doing speeches or manuscripts for a particular publisher, or whatever. So she was in the market for a word processor, and being a small company, like many of these secretarial services are, she wanted something she could afford and that had all of these capabilities. And, that's why I think she came to us. Another interesting aspect of the secretarial service business, and all you consultants listen hard, is that they all have delusions of grandeur, good delusions of grandeur. They would all like eventually to be business services, not just secretarial services. They'd like to offer bookkeeping help and record keeping and consequently they are very interested in the data processing capabilities of the micro- computers. So, the final clincher for P & S Services for us was that they knew that they could, by just changing the diskette, go from word processing capability to data processing capability. In fact, they are now starting to use our client accounting package to keep the books of the many customers whom they also serve as a secretarial service.

Lawyers. The other big market for word processing, especially in the micro field, is lawyers. The main thing lawyers do, as we all know, is collate things they've used before. There's all this boiler plate that's sitting out there that they just like to pop in to a document, one section right after the other, and assemble up a set of interrogatories or a contract or a will. We offer them the capability of storing all that boiler plate on our disk system. Once the appropriate text is booted in off the disk, into the computer, they can go through and tell the computer that wherever you see the name Gene Murrow, replace that with Adam Osborne because I'm on Adam Osborne's case today. And that's precisely what they do.

We hooked up with one lawyer, in particular, who's been very influential to us and has helped us develop a real first class legal package. His name is Tom Lambert. He specializes in personal injury cases arising from aircraft accidents. He's a very smart guy; he and two of his colleagues have engineering degrees and have put in their years at Lockheed before entering law practice. When they go after the biggies like Cessna or Bell Helicopters (whom they specialize in), they can bring to bear a lot of their engineering capability, because they can evaluate rollover rates and all the bad things that happen to helicopters, for example. But the problem was that there was only three of them; when they'd sue Bell Helicopters, Bell would come in with their corporate law staff of 407 typists, 343 clerks and the rest. Tom shared his typist with the other three man office down the hall. He saw our system as a tremendous equalizer. Now when he takes on Bell Helicopters he can spend his time worrying about the engineering data and his overall plans while the machine automatically runs off fat books of interrogatories, which is what lawyers do to make life nasty for the opposition. The interrogatories often, in an aircraft case, run three and four hundred pages per plaintiff. Often it's a lot of boiler plate and cutting and pasting. He used to take three weeks to do each plaintiff's set of Interrogatories, just in the discovery process (which is the opening salvos). But now he's got it down to about 3 days per book, from three weeks, a factor of seven. Plus, he's free now to concentrate on the engineering data. Once again, I think the clincher was that he saw the Data Processing capabilities. So, when he's done preparing the interrogatories and is ready to go to the brief, or whatever it is, he can put in the Data Processing disc. He has written with his colleagues several engineering analysis programs that will model what happens when a 747 hits another one broadside or what happens to a helicopter when one of the rotors begins to loosen. These have become critical to his profession as he's able to put out graphical displays of this data that a jury can understand. So he's got the equalizer between him and the large law firms plus he's enhancing the impact of his own evidence, using a micro- computer, right where it counts -- in front of the jury.

Academics. The third application for our micro-systems and word processing has been among academics: professors, departments and universities. They have a tremendous need for storage of yearly updated documents such as lists of required courses or lists of who's on leave this year. Plus, the department secretaries are under tremendous pressure to relieve the pressure that the professors are under in the "publish or perish" syndrome. They've got to get a few articles out to the journals every year and they depend on the department secretaries to do that typing. We hooked up with a fellow by the name of Dr. William Oldendorf at the V.A. Hospital in Westwood. He told us all about lab reports and articles to journals and things like that and once again he purchased a micro- computer based word processor so he could do those reports and submit those articles to the journals. When the editor comes back with 112 suggested modifications, which a typist can't do, he can sit down and waltz through the manuscript on the video screen, making the corrections. He then hands the diskette over to the typing department or the typist who prints it out. It's been a tremendous help for him.

So that's word processing in three actual applications where people are saving money and are enhancing their businesses right now. They're not playing Star Trek with these machines at all.

Accounting

The second major area, of course, is accounting. We wrote a client accounting package because two or three accountants walked into our store and insisted that they could tell us everything we needed to know. So we worked with these three accountants and came up with a General Ledger or Bookkeeping package that allows you to set up your own chart of accounts and specify what format you want your reports in. You then enter your transactions on a daily basis or weekly basis or hourly basis. And, at the end of whatever your reporting period is, whether it's a day, a month, a year or ten years... it'll go through the usual procedure. It does a trial balance in about 70 seconds. And, if everything eventually balances, (which you hope it does) out come an income statement, a cash receipts journal, check register

journal, five other regular journals, a complete General Ledger, a balance sheet, any schedules that come off the balance sheet and any subsidiary ledgers. All told it takes about an hour to run out all these reports.

One of the accountants that came in was a woman by the name of Audrey Roche. Mrs. Roche is a professor of accounting, in fact head of the department, at Santa Monica College. She also runs her own business which provides the accounting services to her own clients. She was the one that helped us the most in developing this package and she's running it right now. She has a small service bureau which does people's books. She wouldn't have been able to compete doing it manually because a lot of the service bureaus now are automated. So, she wanted to automate and still have her own business. And, that's why the micro became important. A nice side benefit of all of this is that she sees the tremendous movement toward this solution. Santa Monica College is going to be among the first colleges in the area to offer a course specifically aimed at small computerized accounting systems. We're looking forward to providing several accounting systems for the college so that they can do that.

Real Estate Investing

Another interesting application that we found and that we're able to provide micro-computer support for was the real estate syndicators. These guys abound in Los Angeles, where even the Arabs are buying up property at a horrendous rate. What a real estate syndicator does is he gets a group of people together and says "lets all pool our money and buy this building and make lots of money." The trick is to convince everyone that it's a good deal and not to drop out. So, what they need, of course, is a forecast, a spreadsheet, that tells what's going to happen to an income property over the years.

We sat down with two real estate brokers/syndicators. It was one of the most interesting weeks we spent at the store, and came up with a package that does the following:

You enter the purchase price of the property, the down payment, the depreciable basis, rate and term, (if you don't know it you enter some data from the Tax Assessor). Then you enter a whole bunch of financing instruments, which is real estate jargon for loans. The data you enter is the amount of the loan, the term of the loan, the interest rate. That alone would have been easy except these guys are always talking about refinancing, balloon payments, variable interest rates. For example, you can start with three loans and in month 27 let's say, pay off two of the loans with a third loan at a different interest rate, pull some cash out of the deal at that point to pay for the air conditioning system that you're going to put in at that point, which you're going to depreciate for ten years to get a tax write off in year five... So, those are the deals we had to sort through and which caused our programmer to lose most of her hair.

After you do the loans you then have to do an income package. You tell the machine what the gross income of the building is or what the income per unit is and it prints out a nice list of who the current tenants are, what the rate per square footage is on the building as a whole, who's way under, who's way over, and what the vacancy rate is. Then you put in an expenses package. There are 20 categories, everything from taxes and maintenance on down to the gardener and the swimming pool maintenance guy. What the program does after it chews up all this data is a year by year forecast or a forecast for a particular year that gives you the bottom line-- the return on investment. That's calculated in several ways: the income versus the cash you put in, or the build-up in equity, or income vs. equity, and so forth.

One of the biggest features of it is that a syndicator can put in the tax bracket of any investor before this thing flies off and it'll tell him the implications on his personal taxes. That's very important for doctors and others who often end up with too much money at the end of the year. They want to know what the implications are on their own taxes if they invest in this deal. The program also computes post-tax and pre-tax spendable and post tax and pre-tax income as far as the IRS is concerned.

Another nice thing that it does is allow you, after printing out any forecast, to manipulate any single

variable to change the picture. Suppose we fired the gardener, how would that impact our income in the year five?

The syndicator who was the first one to take delivery of the system about five months ago was Sheldon Allman in North Hollywood, California. He is a former stand-up comic. I don't know what that says about real estate. But, one of the things that interested him about the machine was sort of the inverse of what I was talking about before. He finally got sold on buying the machine because of its word processing capabilities. When he's all done running out all of these forecasts on different income properties he then turns to the word processor to write a very nice cover letter and cover description that goes "personally addressed" to each of his prospective clients as "this is a really good deal for you". So he uses both capabilities of the micro-computer.

Retailing

The final thing I'd like to mention in applications is in retail systems. This is a current project of Computer Power & Light. We've done the General Ledger, we have already done a payroll package and we've done some mailing list packages. We have finished up a sales anaylsis and marketing survey package and we're working on order entry, invoicing, aging of receivables, and payables package.

The person who got us into that whole business is Harry Margulies. Harry is the owner of Beverly Stereo and Electronics, which is one of the venerable firms in Los Angeles. They were in it very early on when stereo and hi-fi was much the way micro-computers were a year ago-- very much a new thing. Now he has a large number of employees with varying pay modes plus a tremendous inventory problem controlling the large number of items that a stereo store handles and all kinds of other things where a micro-computer could save a lot of time and money. Harry, right now, is running our payroll system, which handles the payroll for all of his employees weekly. He also uses our mailing system, which is part of the word processing packaging, to do direct mail campaigns.
Just to give you a brief description of the payroll program... it maintains a data base with employees' names and whether they're on an hourly rate or salary or commission or get a guarantee. It has all of the tax algorithms in it, not the tax table. We worked back from the tax tables and figured out what the algorithms were. They match in every case. It automatically will select the list of employees at pay day so you can run through and just tell whether he was there that day or that week or whatever and it prints the checks. It does the quarterly reports, the year end reports and it updates the data base.

So that's some of the applications, some of the things we've done. It maybe fleshes out some of the words like "limited only by your imagination" or "this thing will balance your checkbook" or "play Star Trek" or whatever. I hope I've given you some feel for the kinds of things we're doing.

Limitations

Now I'd like to talk briefly about the limitations. I'm going to tell you the jobs we turned down. And that I think any self- respecting micro- computer dealer should turn down.

They fall into two categories. One is that the job is too complex. The software is just too complex and any decently written package would so swamp the cost of the micro-computer that you might as well, since the cost of the machine is in the noise anyway, buy a mini-computer, a faster and bigger machine. That may sound heretical but that's it folks.

The second category is that in some cases micro- computers are just too slow, or the storage is too limited. I mean we've had people coming in with a check made out for $2,000.00 who actually needed 128 megabyte disk packs.

Here are some of the ones we've turned down. Medical Group Accounting. A doctor would come in and say "Hi, I work with five other doctors in my building and I'd like you to...I've heard about these micro-computers and I'd like you to do a system that'll compute all of the Medi-Cal and Medicare payments, all the insurance reports, keep all of my appointments, age my receivables, keep all of the patient records, and... " you know, 37

other assorted jobs. I think the medical thing is going to be a tough one to crack and we're going to have to work on it over the next few years. But right now I don't see any micro-computer really providing a cost effective solution to the entire medical problem.

The second one we turned down was scheduling classes for a school. I don't know if you're familiar with what it takes to do that. You might have a thousand students and 300 teachers and 275 classrooms and 6 periods per day and 1200 different courses. We had several people come in, from local private schools especially, who said "I'd love to have one of these micro-computers and this is what I want it to do. Build a master schedule, sign up the students into various blocks once the master schedule has been built, then print out rosters of every class so the teacher on the first day of class knows who's in his class. Plus, the room assignments for the students. And then every marking period I want all the grade reporting with class by class averages and department averages and all of that."

That kind of an application will hurt the industry if we attempt it right now. Because people are going to say it doesn't work. I'm convinced that it won't work right now.

The third area that we've avoided, that I feel represents a limitation in the micro- computer in a business area that we're all interested in, is point-of-sale terminals and other on-line or real time applications. It brought down a company by the name of Singer and I don't expect many of the micro-computer companies to attack that one right away. The micro-computer is ideal for it but there are some other problems. I don't know how I'd like to handle a phone call from a customer who calls up and says there are eight people waiting in line to buy something and the machine isn't working right. I think there are so many other factors in this that we had better tread carefully before we all...with new equipment...put in on-line real time systems. Even though I know the equipment can do it, we have to solve some other problems. We have to mature a little bit as an industry before we tackle that one I think.

Other Considerations

Finally, the last topic-- some of the factors in our experience that, as a retail store, meeting businessmen and being out there amongst them, we've discovered that are very, very important and that aren't immediately obvious to the person who's delighting over the instruction set of an 8085 chip or something.

Service. One of them is service. Do we emulate IBM? Do we say to our customers "if anything goes wrong pick up the phone and we'll be there in an hour". To some extent, we have to do that. If we are telling a businessman that he's going to place his life savings, the hard work of himself and his wife, and the success of his business on a little black box with your name on it that he's going to plug into the wall, we better really think twice about service. And, that's one of the biggest areas that we find our energies going into.

There are some nice things that micro-computers have going for them. One of the approaches we use is that instead of taking a service contract, which is expensive relative to purchase price (12% of the purchase price of the equipment, per year) we say a micro-computer is so small you can just tuck it under your arm and throw it into the back of your car after work and bring it on down to the store and we'll fix it while you wait. Which is what we do. You can't do that very well with a System 32 or a Vydek word processor. So, there are some distinct advantages to being in the micro end of this but service is going to remain a big issue.

Support. Another one is support. I don't care how "canned" a package is, the phone rings continuously for the first three weeks that the system is sitting in a businessman's operation. You have to have people on hand who can answer those questions, who can find bugs in programs if they exist. That one's a real sleeper. Having enough people on hand all the time who can answer those phone calls and straighten things out and can also modify packages as they need to be modified.

Software Tools. The third area is software tools. All the little things that the big guys have that we're just starting to develop. Things like utilities for sorting and data entry

and formatting. This is another area where Computer Power & Light has been working. Writing machine language sorts for the Micropolis disk BASIC and writing machine language data entry packages that prevent you from entering a four digit zip code. Things that prevent you from entering a part number using digits when it's expecting alpha-numeric characters. Things like that. It's a big area because when you're out in the business world, you know, the secretary might have been hired last week and doesn't know a computer from a Datsun.

Training. The fourth area that we have always been strong in, and that we feel is important, maybe others may not agree... is training. We say that our training is better than Xerox's, and we've got the affidavits to prove it. It's an area in which we had some experience. The people who are part of Computer Power & Light who were successful teachers earlier in their careers have expended great effort in designing training sessions and courses that are effective. Our classes are not an afterthought, as they are with so many other computer companies. They've taken a lot of our resources, but we feel that they give us a pretty good advantage in dealing with the problems and remaining profitable in this industry.

I hope my remarks have given each of you some insight into the realities of the low- cost business computer "revolution". I'll gladly answer any question you may have. Thank you.

* * * *

<center>MICROLEDGER - COMPUTERIZED ACCOUNTING FOR THE BEGINNER</center>

<center>Thomas P. Bun, MBA, MSEE, 2171 Sharon Road, Menlo Park, CA 94025</center>

Abstract

MICROLEDGER is a General Ledger system,reduced to the absolute essentials. Written in 'BASIC', it employs only two files, Chart of Accounts and Journal. It will run in 8 kilobytes of user memory.

The package is based on a foolproof,step-by-step procedure, designed with the novice computer user in mind. Some familiarity with simple accounting practices is required. The documentation includes a ready starter Chart of Accounts for the small business user.

After entering data, changes are easily made, both to the Chart and to the Journal. Even after posting, adjustments may be made promptly, both to the Profit and Loss Statement and to the Balance Sheet.

Decision problem for business accounting

Small business, just like its large enterprise counterpart, faces an early decision in setting up its accounting practice. Should it first deal with the most urgent parts of its detail procedures, like inventory, receivables, payables? Or should it in the first place, set up an overall framework for its entire accounting? This latter alternative corresponds to the proper organization of the general ledger as the first step. The detail procedures, then, follow as mosaic stones in a picture, whose major outlines already have been properly defined.

The purpose of MICROLEDGER is to supply this overall framework for the small business accounting. To choose this alternative is, therefore, equivalent to the TOP-DOWN systems approach, as opposed to the BOTTOM-UP approach. In this way, it is easier to design procedures that do dovetail together. The final goal, a totally automated comprehensive system,is made easier to achieve, since it can now be attained step by step. When a new module is set up, no change has to be made in old modules, since the general framework has been set up first.

General Ledger packages

There were hundreds of excellent general ledger packages available in the market. The need for a greatly simplified new package,like MICROLEDGER, arose when inexpensive microcomputers suddenly became available to thousands of small businesses that could not afford to use computers before. The existing general ledger packages were designed for more elaborate machines and for users with more resources. Typically, their use required people with computer background and training; not unfrequently, a package would require 15 or 16 files to be set up and manipulated. Complex procedures and lengthy operating manuals were involved. Most of todays microcomputer users would simply keep away from them, because they are too difficult to understand, set up and run.

Organization of MICROLEDGER

The new package is set up in the form of six 'BASIC' programs. According to the particular hardware - and corresponding 'BASIC' version - the programs may be stand-alone or chained. Each program is intensely interactive, maintaining a "conversation" with the user and leading him to the next step.

LEDGER1 is the input program for the Chart of Accounts file. This is the general ledger master file.

LEDGER2 is the input program for the Journal file. This is the transactions file of the system.See an example,Fig.1 in the Appendix.

LEDGER3 lists both files, to help the user in checking them out and to supply audit trails for a complete story of all transactions.

LEDGER4 runs the trial balance, displays the result in a table and then allows the user to ask for posting the transactions to the ledger, Fig.2, top part, in the Appendix.

LEDGER5 reports the Profit and Loss Statement for the period covered by the journal. An example is shown on Fig.2, bottom part.

LEDGER6 reports the Balance Sheet at the closing date. A balance sheet is shown in Fig.3 of the Appendix.

A dual drive diskette device is recommended, in order to enable the user to keep the six programs on one drive, on a protected diskette, while the two data files, the Chart of Accounts file and the Journal file, are kept on another diskette on the other drive.

RESTRICTIONS

The price to pay for such a greatly simplified setup is relatively small. The account numbers are kept to 3 digits and must obey to a predetermined structure:

101-199	Current Assets
201-299	Non-Current Assets
301-399	Current Liabilities
401-499	Long-Term Liabilities
501-599	Owners Equities
600	Retained Earnings
701-799	Revenue Accounts
801-899	Direct Expense Accounts
901-999	General and Administrative Expense Accounts

Fig. 4

The number of journal entries is only limited by the capacity of the diskette. A typical device, like a Micropolis floppy diskette, could contain 10,000 transaction entries. This is much more than would be actually required and recommended for use in practice for a single posting.

The final restriction is the fact that the system does not automatically handle a multiple department situation.

EXTENTIONS

MICROLEDGER was set up to allow for easy bypassing of this restriction. If a user would like to handle a multiple department situation, he can set up his Chart of Accounts for this, by using up to 10 accounts for each purpose allowing for up to 10 departments.

Each department will have its own journal file. These will be kept on separate diskettes, with the same filename, but a physical label differentiating each one.
At run time, the program asks for the name - now the department becomes identified on the report for free. And the proper posting reference being employed, the ledger now handles the multiple department situation correctly.

Other extentions will be added by COMPUMAX ASSOCIATES, the owner of the system, by the development of additional modules. The Accounts Receivable and the Accounts Payable modules are being developed currently. These future packages can be run both as stand-alone applications and as preprocessors to MICROLEDGER since their output files are compatibe in format with the MICROLEDGER journal file.

ADVANTAGES TO THE USER

Many small business users may set up this package with the sample Chart of Accounts as supplied. The sample chart is condensed from many actual small business users charts. This alone may, frequently, justify its employment.

The way the chart is set up, it is very easy to make small changes, additions and deletions for coustomization to any particular requirement.

The balancing outputs reported at posting, and at printing the statements, help a great deal in detecting inconsistencies in the actual figures entered.

Once the necessary adjustments are determined by the user, these can be effected very easily, by the use of the update options in the programs. These allow for insertion,change and deletion of any data item.

After the printout is finally acceptable to the user, it becomes an ideal frontispiece for a month of actual records. The printout is solid, clear, readable. It will be welcome by tax inspectors,I.R.S. auditors, accountants, and - particularly - by the user himself.

The employment of consistent and permanent accounting practices, month after month, take away the drudgery from the bookkeeping chore. Instead, it becomes a pleasant and rewarding occupation, a useful tool for the decision making by the small business owner.

SUMMARY

Two premises were employed in the design of MICROLEDGER:

- keep it as simple as possible, in contrast to extisting general ledger packages with too many options, files and complications;

- make it, nevertheless, powerful enough to accomodate the needs of a general small business user community.

It is obvious that many tradeoffs had to be made to accomodate these premises, that so often lead to conflicting requirements.

It is the hope of the author of this paper, that the product described met both these goals satisfactorily.

It is a pleasant duty to offer my best thanks for the significant assistance received during the development of this product, from the persons listed under ACKNOWLEDGMENTS.

APPENDIX

FLOADG"LEDGER3"

ENTER 1 FOR CHART OF ACCOUNTS LISTING, 2 FOR AUDIT TRAIL OF JOURNAL ? 2

AUDIT TRAIL OF JOURNAL FOR PERIOD FROM 07-01-77 TO 07-31-77

NBR.OF TRANSACTIONS IS 32

REC.#	DATE	TRANSACTION	ACC'T NBR.	AMOUNT
1	07-18-77	MICRODISKETTES TO STOCKROOM	145	1628
2	07-01-77	J.STERN PAYMENT	115	138
3	07-29-77	CREDIT SALES THIS WEEK	130	29603
4	07-04-77	CHECK # 226 U.C.B.	110	-100
5	07-01-77	J.EWING NOTE	120	361
6	07-17-77	E.P.R.I.CONTRACT	125	1151
7	07-31-77	MONTHLY PRODUCTION	135	8282
8	07-31-77	UNFINISHED WORK	140	5569
9	07-31-77	PARTS SENT TO SHOP	145	6342
10	07-20-77	BARCLAYS BANK DRAFT	225	4516
11	07-20-77	ROCKWELL INTERN'TL STOCK	210	5000
12	07-22-77	NEW ROOF ON SHED	250	2668
13	07-23-77	NOVA 3 COMPUTER	270	12951
14	07-31-77	DEPRECIATION-BUILDING	241	-1845
15	07-31-77	DEPRECIATION-FURNITURE	261	-4260
16	07-31-77	DEPRECIATION-MACHINES	271	-352
17	07-31-77	JULY SALES	710	242677
18	07-31-77	CONSULTING INCOME	720	80110
19	07-31-77	ROYALTIES EARNED	730	3959
20	07-31-77	PRODUCTION	810	20193
21	07-31-77	STOCKROOM	820	106429
22	07-31-77	SUPPLIES	830	87308
23	07-31-77	JULY PAYROLL	840	42015
24	07-31-77	SOCIAL SECURITY	845	4191
25	07-31-77	AMORTIZATION EXPENSE	910	2615
26	07-31-77	DEPRECIATION EXPENSE	920	6447
27	07-31-77	BILLS PAYABLE	320	58418
28	07-31-77	MONTHLY OFFICE BUDGET	930	28900
29	07-31-77	RENTALS PAID	975	4000
30	07-31-77	FEDERAL TAXES PAID	980	11414

READY

Fig.1

```
                    BALANCE SHEET - MIDWESTERN MANUFACTURING CO.

              DATE OF 7 - 31 - 77

   CURRENT ASSETS

   CASH IN BANKS                         4000
   CASH ON HAND                           738
   NOTES RECEIVABLE                      1770
   CONTRACTS RECEIVABLE                 10203
   ACCOUNTS RECEIVABLE                  90541
   RESERVE FOR DOUBTFUL A/R        (     1215 )
   FINISHED GOODS INVENTORY             29020
   WORK-IN-PROCESS INVENTORY            35934
   MATERIALS-PARTS INVENTORY            35315
   PREPAID EXPENSES                      9547
                            TOTAL CURRENT ASSETS          215853


   NON-CURRENT ASSETS

   INVESTMENTS                          18626
   OTHER NON-CURRENT ASSETS             20139
   ACCUM.DEPRECIATION-BUILDINGS    (     1845 )
   LEASEHOLD IMPROVEMENTS               17392
   ACCUM.DEPREC.-LEASEHOLD IMPR.   (     6808 )
   FURNITURES AND FIXTURES              14189
   ACCUM.DEPREC.-FURNIT.& FIXT.    (    12840 )
   MACHINERY & EQUIPMENT                62951
   ACCUM.DEPREC.-MACHIN.& EQUIP.   (    21455 )
   AUTOMOTIVE EQUIPMENT                  3994
   ACCUM.DEPREC.-AUTOMOT.EQUIP.    (     2050 )
   OTHER INTANGIBLES                     1944
                            TOTAL NON-CURRENT ASSETS       94237

                            ---------------------------------------
                            TOTAL ASSETS                  310090
                            ---------------------------------------


   CURRENT LIABILITIES

   NOTES PAYABLE                        24600
   ACCOUNTS PAYABLE                     78954
   ACCRUED LIABILITIES                  13307
   FEDERAL & OTHER INCOME TAXES         12839
   CURRENT INSTALLMENTS ON LOANS          192
                            TOTAL CURR.LIABILITIES         129892

   LONG-TERM LIABILITIES

   LONG-TERM DEBT                       73551
                            TOTAL LONG-TERM LIABILITIES     73551

   OWNERS EQUITIES

   COMMON STOCK                         58154
   CAPITAL SURPLUS                      23066
   RETAINED EARNINGS                    25427
                            TOTAL OWNERS EQUITIES          106647
                    ---------------------------------------------------
                    TOTAL LIABILITIES & OWNERS EQUITIES    310090
                    ---------------------------------------------------

                            Fig.2
```

```
PLOADG"LEDGER4"


TRIAL BALANCE FOR PERIOD STARTING 7 - 1 - 77  TO  7 - 31 - 77
-----------------------------------------------------------------


EXPENSES        313512      REVENUES         326746

INCOME          13234

TO ASSETS       71652       TO LIAB.         58418

                            TO R. E.         13234

ASSETS BALANCE LIAB.+O.E.
DO YOU WANT TO POST, Y OR N? Y
 30 JOURNAL TRANSACTIONS POSTED TO CHART OF ACCOUNTS FILE.

READY
PLOADG"LEDGER5"

P. & L. STATEMENT PROGRAM
------------------------
ENTER COMPANY NAME (UP TO 30 CHARACTERS),START DATE OF FISCAL YEAR (MMDD
YY)? "MIDWESTERN MANUFACTURING CO.",010177


              PROFIT & LOSS STATEMENT - MIDWESTERN MANUFACTURING CO.

                 FOR PERIOD FROM  1 - 1 - 77   THROUGH  7 - 31 - 77

        REVENUES

SALES                              242677
OTHER OPERATING REVENUE             80110
INVESTMENT AND ROYALTY INCOME        3959
                            ---------------------------------
                            TOTAL REVENUES        326746
                            ---------------------------------

        EXPENSES

MERCHANDISE FOR INVENTORY           20193
MATERIALS PURCHASED                106429
SUPPLIES PURCHASED                  87308
DIRECT LABOR                        42015
LABOR OVERHEAD                       4191
           TOTAL DIRECT EXPENSES           260136

AMORTIZATION EXPENSE                 2615
DEPRECIATION EXPENSE                 6447
OFFICE EXPENSES                     28900
RENTALS PAID                         4000
TAXES PAID                          11414
           TOTAL G.& A. EXPENSES            53376

                            ---------------------------------
                            TOTAL EXPENSES        313512
                            ---------------------------------
                                INCOME            13234
                            ---------------------------------


READY
```

Fig.3

REFERENCES

The following textbooks were used during the development of MICROLEDGER:

Earl A. Spiller: "Financial Accounting" Irwin, 1971

Richard Mattesich"Accounting and Analytical Methods", Irwin, 1964

John Dearden & F.Warren McFarlan:"Management Information Ststems", Irwin, 1966

Anthony, Dearden and Vancil:"Management Control Systems", Irwin, 1965

These books might be found at the libraries of the University of Santa Clara and Stanford Business Schools.

ACKNOWLEDGMENTS

The following persons were of significant assistance during the development of the MICROLEDGER.

Paul J. Terrell, as President of Byte,Inc. served as inspiration and support from the beginning.

Mike D. Lipschutz and John K. Borders of Microcomputer Software Associates – MSA, contributed with valuable advice.

Boyd Wilson and Mike Black, of the Mountain View Byte Shop, contributed with many hours of patient help in setting up the hardware.

Zulmira McMorrow of the McMorrow Engineering Group,one of the first users,completed the first large-scale data entry.

ABOUT THE AUTHOR

Thomas P. Bun
  President
CompuMax Associates
505 Hamilton Avenue
Palo Alto, CA 94301

Thomas P. Bun was born in Budapest, Hungary. Having graduated in general engineering,with a major in planning, from the Budapest Polytechnical University,he worked in Europe and Latin America,finally settling in California. Mr.Bun obtained an M.S.degree from Stanford University,majoring in Digital Systems/Computer Science, and an M.B.A. from the University of Santa Clara.

Positions occupied by Mr.Bun have included: senior systems analyst with Stanford Research Institute; project manager in microcomputer application design with Rockwell International; manager-systems with Light S.A., the largest privately owned electric power utility in Latin America; director of Reduto, a Brazilian engineering company; president of the Latin American Astronomical League; member of the Council of the National Commission for Space Activities of Brazil; management science analyst with I.S.I.Corporation and product engineer with Smith-Corona Marchant Corporation. Since January 1977, Mr.Bun is president of CompuMax Associates, a Palo Alto-based software house, specializing in the production of application packages for microcomputers.

Money for Your Business—Where to Find It, How to Get It.

Don Dible
468 Robert Rd.
Vacaville CA 95688

One of the recurrent problems that plagues most owner-managed businesses is the shortage of adequate capital. In the case of new businesses, this problem can be severe.

Promising small businesses traditionally have relied upon such sophisticated money sources as investment bankers, venture capitalists, and federally licensed and leveraged Small Business Investment Companies (SBICs) for their equity capital needs. In today's market, these sophisticated investors have many options from which to choose. Regardless of market conditions, their objective continues to be the maximization of return on investment, consistent with intelligent risk analysis.

Their first option is the purchase of securities in publicly traded companies at bargain basement rates. Their second option is the private placement of *growth* capital in young, but already established, companies of demonstrable merit. In a tight money market, the terms of such an arrangement may be very attractive for the investor. The investment option representing the highest risk, obviously, is in financing the start-up company. Given the first two options, a private placement with a start-up company is, I think you'd agree, a most unlikely choice as an investment vehicle.

What about banks? Surely, you can get a personally endorsed loan for your business from the friendly loan officer at your commercial bank. Not necessarily. In evaluating the financial statement of any business, a banker will pay particular attention to the ratio of debt to equity capital. A general rule is that you cannot borrow more money than has been invested as equity capital. Furthermore, most bankers are not eager to lend money to new businesses at all, preferring to wait until there is at least some history of profitable operation.

How, then, can you assemble the resources you'll need to get started and to keep you going until you begin to turn a profit? Take heart; there is hope. There are, in fact, four different money sources that can now help you stretch your business dollars: 1) personal financial resources, 2) trade credit, 3) customers, 4) economizing.

Personal Financial Resources

Many of us significantly underestimate the value of our personal resources when taking a financial inventory. Let's look at some of the more obvious (and not-so-obvious) resources you may have:

Home In the recent inflationary period, the value of practically all residential real estate has appreciated substantially. If you have owned your own home during this period, you are indeed fortunate. The difference between the current value of your home and the balance outstanding on your mortgage may represent an equity asset of between $20,000 and $50,000. You might consider converting this asset to cash by 1) taking a second mortgage in the amount of your equity, 2) refinancing your mortgage, 3) subdividing your lot and selling part of it, or 4) selling your house and moving into an apartment. Every year tens of thousands of people go into business using "house money" to bankroll their ventures.

Life Insurance A loan based on the cash value of your life insurance policies can be a low-interest source of money. Many policies provide for automatic loans from the insurance carrier at interest rates far below the prime rate charged by commercial banks.

Stocks and Bonds You may own stocks and bonds that, for a variety of reasons, you do not wish to sell. Such instruments make ideal collateral for loans, and you can borrow anywhere from 40% to 60% of their current value, depending on the lender.

Credit Cards and Personal Credit When you prepare your financial business plans, you will surely want to take the fullest possible advantage of your credit worthiness. At the same time, you will want to minimize the amount of money that you have to take out of the business to meet your personal financial needs. This is particularly true when the business is just getting started and every dollar in the treasury is needed to finance business growth. During this period, personal credit cards of all types and descriptions can give you just the extra leverage you need. Instead of drawing a heavy salary in the first year of operation, supplement your modest cash draw with the judicious use of credit cards.

Credit cards come in a great variety of classifications: bank cards such as Master Charge and BankAmericard, travel and entertainment (T & E) cards such as Diners Club and American Express, as well as cards for department stores, oil companies, airlines, and so on. With a good credit record, it should be no great feat for you to secure enough credit cards to give you a personal line of revolving credit in excess of $30,000.

Now, obviously, you can't buy a carload of raw materials for your business with your bank credit card, but you can cover a lot of your *personal* expenses with it—as well as such business expenses as transportation, food, and lodging. With a little thought, I'm sure you can come up with a variety of ways to employ credit cards in financing your particular business.

While we're on the subject of personal credit, you should give serious consideration to arranging for the installment purchase of an automobile and other major consumer items *before* you start your business. You'll find it a lot harder to qualify for this kind of financing after you've started. Lenders tend to worry a great deal about the financial stability of new small businesses and their founders. Your credit application looks a lot better when, under "Current Employer," you show that you have been employed with "Solid as a Rock, Inc." for the last six years instead of with "Shaky at Best" for the last six weeks.

Relatives and Friends I happen to consider relatives and friends to be extremely valuable personal financial resources. Where else can you find lenders who, simply because they like you, will advance money on anything as risky as starting a new business? However, do yourself and your lenders a favor. Document their loans. Draw up a formal note showing interest charged and the dates on which principal and interest are payable. You never know when a personality problem may arise that could precipitate calling the loan. That can become awfully messy.

Take the case of the young man whose rich aunt loaned him $100,000 with which to start his business. The formality of a loan agreement was ignored. Six months later the aunt died. The heirs then forced the entrepreneur to liquidate his business so that they

could each get their share of the aunt's estate.

Relatives and friends can also come in handy as cosigners. Let's assume that you apply for a loan and the lender decides that your qualifications are marginal. The availability of a financially strong cosigner can swing the balance in your favor. Your relatives and friends can prove to be very important personal assets. Don't overlook them in your financial planning.

Trade Credit

According to a U.S. government study, trade credit constitutes a 33% larger factor in financing the business community than that represented by bank loans. Clearly, every company treasurer should give serious consideration to trade credit in financing a small business.

Customarily, suppliers provide trade credit as an inducement to their customers to do business with them. Although credit terms in most industries are extended on a net 30-day basis, overall averages may stretch this to 45 days, 60 days, or even longer depending on the condition of the overall economy and the particular industry involved. Under the guise of "cash management," many companies make a habit of vigorously enforcing their collection policies while simultaneously treating their accounts payable with cavalier disregard.

Just how you handle your payables is your own business. But you should be aware of the fact that not everybody in business makes a habit of paying bills the day the first invoice arrives.

When it comes to making use of trade credit as a tool in financing your business, I recommend that you establish with your suppliers, in advance of purchase, the best extended credit terms you can negotiate. Depending on how hungry your suppliers are for your business, you may be able to arrange extremely attractive terms. Securing competitive bids will greatly improve your bargaining position. However, once you have agreed to terms of payment, honor your commitment if you value your credit. Nothing is more difficult than trying to operate a business when your suppliers will ship to you only on a COD basis.

Customers

There are many ways in which customers can be induced to help you finance your business. The degree to which they are *willing* to do so depends on the extent to which you enjoy a monopoly on the goods or services that you offer. In other words, no one will pay in advance if he can get essentially the same goods or services from another supplier under more liberal credit terms. However, if you have the only game in town, you may be in a position to get your customers to finance your business.

As you know, the telephone company, electric and gas companies, the post office, and a host of government monopolies require deposits in advance (independent of your creditworthiness), so that you may enjoy the benefits of their services. The same kind of policy may be applied in the operation of certain small businesses and selected industries.

A San Francisco Peninsula electronics company, which sells a patented device available nowhere else, provides a good illustration of customer financing. The company's customers are required to make a deposit of one-third of the purchase price when the order is placed. Another one-third is payable on delivery, and the balance is due in 30 days. As a result of these favorable terms of sale, this company has been able to finance a highly satisfactory growth rate while experiencing almost no problems with cash flow.

An extreme example of this type of financing is the case in which the customer is required to pay the full purchase price at the time the order is placed. You may be surprised to learn that you have been financing certain of your own personal suppliers on this basis for years. I am referring, of course, to magazine publishers. If you take a three-year subscription to a magazine, you may get the first one or two issues on credit based on your promise to pay. However, in order to continue receiving the magazine, you must pay the subscription bill, even though you won't receive your final shipment for almost three years.

Hugh M. Hefner, publisher of *Playboy* magazine, played this customer-finance game with admirable success when he initially offered lifetime subscriptions for $100. Obviously, he needed the money to finance his growth. Had the subscribers to some of the early issues bought $100 worth of *Playboy* stock instead of a subscription, they could have picked up a tidy profit.

In many industries, including a number of the construction trades, it is usual for the suppliers to receive progress payments when previously agreed-upon levels of project completion are achieved. Highway construction, aircraft assembly, and other large-scale projects are often financed in this way.

A special form of customer financing occurs in a number of service and manufacturing industries. Here the customer provides raw materials that the vendor transforms into finished goods. In the book printing trade, a publisher may supply the printer with paper; in a machine shop, the customer may provide the metal to be worked; and in a tailoring service, the customer may provide the cloth to be fashioned into a dress or a suit. In each case, the vendor avoids the expense of financing the purchase of raw materials.

Another way of getting the customer to pay in advance for goods or services is to establish some kind of "membership" arrangement. Here the customer may pay an annual fee for the privilege of attending meetings. Using a similar membership approach, some discount department stores require customers to pay a membership fee for the privilege of shopping there.

A novel twist on this customer-backed approach to business finance is seen in physical fitness spas, dance studios, and other contract service organizations. Customers sign an agreement to purchase the service offered for a period of one or more years. In most cases, the intentions of the customer are honorable and sincere at the time he executes the contract. However, many people lose interest in fitness and other self-improvement programs after a short time, and the incentive to pay the installments on the service contract may falter. In anticipation of these long-range collection problems, the original holders sell the contracts to finance companies at a substantial discount. While the average individual may balk at paying the original contractor for services not used, he is more likely to pay a finance company when notified that it has taken over his contract. The result is that the original contractor gets a handsome chunk of cash almost immediately after the customer signs the contract, whether or not the customer continues to use his facilities. Once again, the customer has financed the business, albeit indirectly.

Economizing

In preparing your pro forma financial statements, you probably allocated quite a bit of money for office equipment and other capital expenditures. If you figured on buying new equipment, figure again. Used equipment is what you want. That way, you may find that you need a lot less cash than you originally thought. Forget the rosewood paneled office with the Italian marble-topped desk, too; that comes later. And most financially strapped entrepreneurs quickly learn the delights of night coach and special tour package rates to save a few dollars when traveling. There are many other ways to economize in the operation of your business; we'll get to them shortly, but first let's concentrate on where to find used equipment.

1. Used equipment dealers may handle anything from office equipment to laboratory test equipment to cash registers to display cases for butcher shops and retail stores. You'll find these dealers listed in the Yellow Pages of your telephone directory under such headings as "Used Equipment Dealers," "Second Hand Dealers," and "Surplus Merchandise." You'll also find dealers listed under generic headings such as "Office Furniture—Used."

2. Dealers in new merchandise invariably find themselves stuck with a variety of goods that cannot truly be represented as new. "Demonstrator" and "loaner" equipment (provided for the temporary use of customers who are awaiting delivery of new equipment or who are having their own equipment repaired) fall into this category. These dealers may also have floor samples, warehouse- and freight-damaged goods, and obsolete-but-serviceable rental equipment available at a substantial savings.

3. Bankruptcy and liquidation auctions provide an opportunity to get some real bargains. To secure information on where and when auctions are to be held, consult the Yellow Pages of your telephone directory under "Auctions" or "Auctioneers." Many auctioneers maintain mailing lists of interested clients and send out brochures announcing forthcoming auctions.

    You may also obtain information on bankruptcy auctions by contacting the bankruptcy court in your area. You'll find this court listed under "U.S. Government" in the White Pages of your telephone directory.

4. Bankrupt companies that receive protection under Chapter 11 of the Bankruptcy Act are very likely to be interested in liquidating some of their assets. If you are lucky enough to learn about such a company in your own industry, you are in a unique position to fill your equipment needs quite reasonably. Bankruptcy filings are announced in the legal newspapers of record serving various communities across the country. Also, when a large company in a particular industry files for voluntary bankruptcy under Chapter 11, the trade journals serving that industry will usually carry mention of this fact. Upon learning of such a bankruptcy, don't hesitate to call the company involved to determine whether the owners are interested in selling some of their assets.

5. Now and then major corporations simply decide to get out of a particular industry and shut down one of their divisions. A friend of mine who has his own company read about such an instance in a trade journal. He purchased a $100,000 (price when new) piece of test equipment for a mere $7,000 cash. Then he called a leasing company and received $45,000 for the same equipment when he agreed to lease it back from them over a five-year period. He realized an immediate cash infusion of $38,000.

6. The United States Government is the single largest consumer of goods and services in the world. Not surprisingly, it also disposes of enormous quantities of surplus goods on a more or less continuous basis. For information on the sale of surplus government equipment at locations all over the world, write to the Department of Defense Surplus Sales Office, Box 1370, Battle Creek, Michigan 49016, and the Assistant Commissioner for Personal Property Disposal, Federal Supply Service, General Services Administration, Room 926, Crystal Mall, Building 2, Washington, D.C. 20406. Local representatives of these agencies are listed in the White Pages of your telephone book under "U.S. Government."

7. Classified ads provide a simple, convenient, and inexpensive means of finding used equipment. You can consult the listings under the headings of interest to you, or you may want to advertise the fact that you are looking for a particular item. In some instances, trade journals carry classified listings, thereby permitting you to confine your search for specialized items to the more likely sources of supply.

Barter   Bartering your goods and services is a primitive-but-fun way to save money. It also affords certain tax advantages. I know of a carpenter who remodeled an orthodontist's home in exchange for having his daughter's teeth straightened. I also know of a plumbing contractor who paid for his appendectomy by plumbing his surgeon's vacation home. Radio stations have been known to exchange advertising time for consumer merchandise to use as premiums. The opportunities are unlimited.

Do It Yourself   When the cash supply is limited and the money for meeting a payroll is nil, you simply have to learn to do things yourself that you might otherwise hire someone else to do. You have undoubtedly heard that many small businessmen work 80 or more hours a week. Many of these companies are also husband-and-wife operations, where the wife works virtually without pay until the business starts turning a profit. This kind of toil may not sound like much fun, but it does save money.

Then, too, a lot of businessmen, in response to economic pressure, find that they have many talents and skills they never appreciated. A janitor used to empty their wastebasket when they worked for Big Business, Inc., but they now take care of this occupational specialty themselves. On opening a restaurant, they may find that they possess the dexterity of a short-order cook. The may learn how to write advertising copy, how to repair their machinery, or how to operate a typewriter and a ten-key adding machine.

In some cases where a specialist is needed but there is no money with which to hire one, the entrepreneur may have to take courses at a local college to learn the skill himself. It is often surprising what you can learn to do when your economic survival is at stake.

Free Publicity   Free publicity can do wonders for your business —and the price is right. You don't have to hire a public relations firm on retainer to get it, either. What you *do* need is guts enough to call a newspaper or trade journal editor, a radio announcer, or a television personality and explain your story. If you have something to say that will be of genuine interest, educational value, or amusement to the audience served by the medium you have selected, you have a good chance of getting publicity.

One businessman I know sent out new-product releases describing a $200 device to 12 trade journals serving his industry. He

received more than 1,500 inquiries in response to articles carried in the two journals that printed his story. His total cost was less than $10.

I could doubtless catalogue dozens of other ways of saving money in your business, but I'd like to conclude by suggesting a different point of view. It is easy to become preoccupied with saving nickles and dimes instead of figuring out how to make dollars. Nothing succeeds like a company with the right product at the right price at the right time. The electronics giant Hewlett-Packard was started in a garage in the 1930s. The company grew rapidly with limited invested resources for a very simple reason: the founders were selling unique and superior products with high profit margins to a market eager to buy. Go thou and do likewise.

RECOMMENDED READING

Baty, Gordon B. *Entrepreneurship: Playing to Win.* Reston, Va.: Reston Publishing Company, Inc., 1974.

Brady, Frank. *Hefner.* New York: Macmillan Publishing Co., Inc., 1974.

Deiner, Royce. *How to Finance a Growing Business.* New York: Frederick Fell, Inc., 1965.

Putt, William D., ed. *How to Start Your Own Business.* Cambridge, Ma.: The Massachusetts Institute of Technology, 1974.

SELLING YOUR HARDWARE IDEAS:
HOW TO START AND RUN A MANUFACTURING ORIENTED COMPUTER COMPANY

by Thomas S. Rose
President, Astro Electronics Co.
P.O. Box 1429
Alameda, California 94501

Introduction

So, you're thinking about starting your own business. You have a great idea for a piece of hardware that will revolutionize computers. Not wanting anyone else to profit from your ingenuity you decide forming your own company is the best thing to do. If that is your inclination I have one piece of advice for you; DON'T! Your chances of succeeding are about 50-50. Worse than that, though, your chances of failure with possible bankruptcy and personal financial disaster are about the same.

If you are reading this you probably are not interested in taking my original advice. Consequently, I will give you some advice that should help improve your chances of success. Some of the important things you should know about are financing, government regulations, production, marketing, organization structure, bookkeeping, and ethics.

Is This Really For Me?

The first question you should ask yourself is what are the chances I will succeed? There is very little point in going through all of the trouble of starting up if you know in advance that you are doomed to failure. As you consider the product you intend to sell the first indication that it may fail is the existence of similar products on the market. For example, there is no way you can manufacture an 8080 microprocessor and compete with the likes of National Semiconductor, Fairchild, and the handful of others that already make 8080's. I should really say there is one way you might compete. You could if you had a few million dollars to invest, a top management, production, and marketing team and a bit of luck. If you are like me, though, you have none of the above. Even so, if you had the dollars you would probably just want to retire (I would anyway) and if you had all those top people they would tell you not to try competing with the big boys. Therefore, you should try to find a product which is largely unique.

Let us say for example that you were going to produce a memory board with 4K of RAM. Now that does not sound very unique. If you sell that 4K for $20.00, though, you have got something. Note that the physical characteristics of the product are not the only things which may make it unique.

Of course, the best kind of uniqueness is a big breakthrough. Something akin to the invention of the vacuum tube or the transistor is what I have in mind here and with an idea like that your chances of success dramatically improve (but are still not certain).

Consider next the size of your market. In the case of hobby and personal computers your market is large and growing. A walk through the Computer Faire will convince you of that. I have read estimates that indicate there are 35,000 $\underline{/1\_7}$ or 50,000 $\underline{/2\_7}$ personal computer installations in this country. These estimates are not current. By now there are likely to be many more and millions of installations are foreseeable by the 1980's. The buyers are there.

Price is a factor in determining the size of your market. There are more people who can fit a $5.00 item into their budgets than there are who can fit a $500.00 item in. Not everyone of those 50,000 or whatever people will buy your product but a lot more will consider it if its price is low.

Now, bring all of these factors together. Take a rough guess at how many people will buy your product at the price you want to sell it. Now, figure out roughly how much it will cost you to make the product. Next, divide your sales estimate by 2 or more and multiply your cost estimate by 2 or more (these revised estimates will probably prove more accurate than the originals) and determine whether or not you will make a profit. If it looks like you will lose money do not bother going into business (unless you are a philanthropist dedicated to all of your fellow computer hobbyists).

Financing

If you have gotten to this point and still believe you have something worthwhile your first step in starting business is to get together some working capital. There is no manufacturing business on Earth I know of that can come into existence without money/3/. In the case of manufacturing the saying "It takes money to make money" is an ironclad rule.

How much money will it take to get started? This will vary from business to business. You will need money to satisfy requirements for deposits for various government agencies, to pay for raw materials, to meet payrolls, and for a variety of miscellaneous expenses. Additionally, most businessmen embarking on a new venture will make at least one very serious and costly mistake during the first two years of business. If you do not want your business to end when that mistake is made you should reserve something to cover it. After you have determined what you will need double it or triple it and that is the extent to which your business should be financed.

The first place to look for financing is to yourself. Personal savings may be your first financial source. Some people may want to mortgage their property, sell off other investments, or adopt a more frugal lifestyle. Unless you are deeply committed to your business you should have strong second thoughts about mortgaging your property. Ask yourself if it is worth the loss of your home to gain the independence and other potential rewards that come from owning and operating your own business. For some people it is and by all means they should take this action. Selling off other investments is relatively easier than mortgaging your home. Again, though, you must determine for yourself how badly you want to go into business. Keep in mind that the money you invest in your own firm is one of the riskiest investments you will ever make in your life. All but the most poverty stricken people should be able to make the adjustment to a more frugal lifestyle. If you are unwilling to take those steps I do not believe you have the determination necessary to succeed in your own business. If you think you are unable to change your lifestyle then you have not scrutinized your budget closely enough and I suggest you take another look at it. Trim the fat and perhaps some of the lean.

If your business requires more financing than you alone can provide you will have to turn to outside sources. You may want to consider relatives or friends. Be careful! If your business is a success and you failed to allow relatives or friends to invest they may be upset that they could not share in the rewards. If you do include them and your business fails they will also be upset. Your relations and friends may be more willing to assist you with financing than institutions which know relatively little about you and have no track record to look at. Depending upon how you relate to your friends and relatives you may or may not want to ask for their financial participation.

The two major institutions that most people think about in connection with small business financing are banks and the Small Business Administration(SBA). Most of the SBA's aid is in the form of loan guarantees for loans actually made by a bank. It is usually easier to get an SBA loan if your financial situation is not particularly strong. In most cases it is best to provide as much financing yourself as you can. Even if that is not enough by itself a financial institution will look more favorably upon an entrepreneur who has committed a substantial part of his own fortune to the enterprise. An essential ingredient in securing financing from an institution is a comprehensive formal business plan. This may also be useful to you in encouraging or discouraging yourself in a relatively objective way as you consider your new business. As an aid to preparing your business plan it is absolutely essen. al that you purchase the book, Up Your Own Organization by Donald M. Dible/4/. I guarantee that as a novice entrepreneur this will be the best purchase you will make.

In addition to the suggestions Mr. Dible has for a business plan you will find an extensive list of financing institutions. Among those he suggests which may not come readily to mind are consumer finance companies, credit unions, the Economic Development Administration, venture capitalists, and some thirty-five or so others. His book also tells you the procedures for securing financing from these institutions. If you cannot finance your business with all of the various sources available your business probably is not worth financing.

The Government

The government is a monster. You

will have to deal with three sets of government; federal, state, and local. With each set you will have a plethora of agencies, bureaus, and administrations that want some form of control over your business. With every agency you deal you have a couple of strategies you may employ. The first I call the compliance and conciliation method. The second I call the adversary method.

In the first strategy you make every attempt to comply with the written laws and regulations that apply to you. additionally you get to know the bureaucrats you deal with and by so doing secure the best treatment you can get. For example, when I applied for a resale certificate (business liscence) from the California State Board of Equalization I was told I would have to deposit a substantial amount of money as security for payment of sales taxes. This was one of those items I had not budgeted. To have continued in business after making a security deposit of that size would have been impossible. I made one phone call to a man responsible for setting deposit requirements and succeeded in getting my security deposit requirement cut in half. I was not beligerent and explained cooly and rationally to the man why I felt the size of the deposit was excessive. Do not antagonize the bureaucrats. They are like German shepherds. If you poke sticks at them they get mean and will make things very difficult for you. If on the other hand you pat them gently on the head they will be friendly to you.

The adversary method is a bit more difficult but you may reap rewards in the long run. By the way I only reccomend this method for those of you who elect the proprietorship and partnership forms of organization and not the corporate form. Furthermore, I am not advocating that you do anything illegal. In the adversary method you make no assumption that the government has no business messing around in your business. You do not register in any way with any government bureaus or agencies unless the laws clearly require it. If they do not know you exist they cannot come around and bother you. Even if they know you are there and they start pestering you you should be well aware of your rights. You are not required to give anyone any information which may be used against you in court. Since as a practical matter any information you give may be used against you you need not give anyone any information about yourself or your non-corporate business which is really an extension of your-

self. The reason this method is difficult is that it is largely based on your natural rights as expressed in the U.S. Constitution. Most judges are much more interested in the complex words in some statute than the simple and clear words in the Constitution and consequently do not accept Constitutional arguments. Nonetheless, signifigant savings may be realized if you do not have to apply for endless permits and pay never ending fees.

There are a number of agencies you need to be concerned about. First and foremost of these is the Internal Revenue Service (IRS). If you are using strategy one you should go to the IRS and ask them for all of theliterature they have that is relevant to your situation. You should explain the type of business you do. Secondly buy the Proceedings of the First Annual West Coast Computer Faire and read the article on pge 202 by Kenneth S. Widelitz/ 2_/. In the article Mr. Widelitz gives you some ideas for tax savings that you may employ. You should subscribe to the Wall Street Journal/ 5_/ and read their regular collumn and frequent articles on changing IRS policy. You should buy the book Small Time Operator by Bernard Kamoroff/ 6 _/ for his tax saving ideas and general tax comments. After Up Your Own Organization this is the second most valuable book to buy. You should hire an accountant who is a specialist in taxes. This is especially important as your sales and income grow. If you employ the second strategy I suggest you find an alternative bookstore and read some of their tax avoidance (not evasion) literature.

Other agencies of the federal government you may find yourself dealing with are the Occupational Safety and Health Administration (OSHA), the Federal Trade Commission (FTC), and the Federal Communications Commission (FCC) among many hundreds of others.

If you have employees in your business, especially in a factory or on an assembly line you will have to be concerned with OSHA. Under a major federal act they administer certain safety codes. The FTC deals with warranties and truth in advertising. For a fine discussion of warranties I suggest you read another article by Mr. Widelitz starting on page 72 of the First Faire Proceedings/ 2_/. If any of your products involve long distance communications (farther than from one point in a building to another point in the same building roughly) the FCC may be involved.

Of course, I cannot begin to list all of the agencies which may concern you in your business. If you have any doubts about your relations with federal government agencies I strongly suggest that you do some research on the agencies and that you hire a lawyer competent in federal regulation.

At the state level (in California) you have two agencies which will cause you the most concern. The first is the State Board of Equalization (SBOE). The SBOE primarily deals with the administration of the sales tax. If you are buying materials that are to be resold you will need a resale certificate issued by the SBOE. This will allow you to buy the materials without paying sales tax and requires you to collect sales tax on all final sales you make.

The second agency is the Franchise Tax Board (FTB). The FTB is to California what the IRS is to the U.S. The advice I gave for dealing with the IRS largely applies to the FTB as well.

If you have any questions about any other California agencies or agencies in other states you should enlist the services of a lawyer.

Regulations at the local level vary widely. Localities generally impose zoning, thereby restricting certain activities such as manufacturing to certain specific locations. If your company's name is other than your own you will need to file a fictitious business name statement usually with the county clerk.

Production

This is probably the easiest topic for people who are knowledgeable about computers. For the most part production considerations are just common sense. Still t ere are a few tips I can give you.

When you design your product try to use parts which are widely available. Every time you use a custom part your product will cost more and it will be more difficult to design around custom parts. Also, when these type of parts are used you are more likely to experience delays in production.

Try to watch your costs. Shop around for the best price possible for all of your materials. This is especially true if you have lots of time devoted to your business and not much money.

There are a number of quantitative techniques available to production managers. Generally, these are only useful in large production facilities. In any case I do not claim to be an expert in this area and for further information you should consult one of the texts available on this subject.

Marketing

Marketing systems generally consist of three elements; distribution, pricing, and promotion. Distribution deals with the questions of how your product gets from you to your customers. In the personal and hobby computer industry you have essentially three choices; sales to middlemen or wholesalers, to retailers, or directly to the ultimate consumer. To the best of my knowledge wholesalers are an insignifigant force in this part of the electronics industry. If you sell to retailers you cannot charge as much as you would if you sold directly to consumers. However, you will have larger amounts of sales at one time. With the proliferation of computer stores in the past couple of years this can be an attractive distribution channel. If you sell directly to ultimate consumers you can sell units at a higher price but you will have higher expenses as well. It will cost you more per unit to process an order for one unit than for twenty units.

Pricing is an important consideration. If your price is too high you will not sell anything and if it is too low you will lose money. You should consider the prices your competitors are charging. Due to rules of the marketplace the price they charge is probably about the price that is right for you. Your price may vary around that standard because of differences in such things as quality and features.

What most people think of when they think of marketing is promotion or one of its categories, advertising. Promotion involves not only advertising but the areas of publicity and personal selling as well. It is my opinion that you cannot through promotion make someone buy something he is not inclined to buy anyway. You can make someone who is unaware of your product or its features more knowledgeable about it. Publicity is favorable news or information about your products for which you do not pay. Some publications even solicit information about your products which will appear for free. Publicity is clearly the best deal available in the promotion area. Personal selling is most useful if your sales are to retailers. It may also be important if you are dealing with a large number of customers at one time or a relatively high priced item.

I think that salesmen are born and not made. I know that I could not sell a cure to a dying man. If you have the knack you may want to do some personal selling yourself. If your business requires personal selling and you do not feel qualified you should hire salesmen who are able.

In advertising you have two major considerations; how much and where? For a manufacturer about the only place for advertising is in one or more of the magazines that now cater to the growing group of computer hobbyists. The broadcast and newspaper media reach too broad an audience including many people who are not interested in computers (yes, some people could not care less). They are not cost effective. Advertising is not cheap. Prices for a small black and white ad in a typical small systems magazine will range from $200 to $300 and will go up to around $2000 for a full page color ad. Still, you will reach a large number of people who may be interested in your product this way. If your advertising budget is large enough you can enlist the aid of an advertising agency. The nice thing about an agency is that they cost you little or nothing. They make their money by receiveing discounts from the media that they place your advertising in.

Form of Organization

I think far too much attention and worry is paid to this topic. Unless your special financing or tax needs dictate otherwise (your accountant can help you here) you should organize as a sole proprietorship. Enough said.

Bookeeping and Accounting

Since my training is in accounting I could go on at quite some length about this topic. I will not. As long as the records you keep show clearly how much your company owns, owes, what your investment is, how much you sell and what it costs you to sell it you will be in good shape. If you want to be able to handle more complex topics such as depreciation, double entry accounting, deferrals and accruals, and periodic reporting I would suggest that you contact your local community college for information about their accounting courses. One or two nights a week for one semester will usually qualify you to handle all of the day to day accounting tasks. For additional help you should seek the services of an independent accountant.

Ethics

Your primary obligation is to maximize your profits. The free enterprise system is based upon profit maximization. Note that profits are not measured in dollars alone but may be measured in such intangibles as personal satisfaction and happiness. As long as you do not use force or fraud to make your profits I believe you are operating ethically.

Conclusion

I can only begin to tell you about starting your own business in this short space. There is much more you will need to know. You can gain much of that knowledge by reading the materials I have suggested. Your local community college can be a great resource if you will enroll in some of the business courses they offer. Ultimately, though, the best teacher is experience. In that experience I wish you the best of luck.

References/Bibliography

1. Alan Kaplan, "Filling the Need for Consumer Software", Computer Decisions, October, 1977, p. 14.
2. Jim C. Warren, Jr., ed., Conference Proceedings of the First West Coast Computer Faire.
3. When I refer to money throughout the paper I am using the definition "A medium of exchange". This differs considerably from the legal definition which is important in some tax matters.
4. Donald M. Dible, Up Your Own Organization, 1974, Entrepreneur Press. Available through most bookstores.
5. The Wall Street Journal is available on most newstands and subscription information is available in all issues.
6. Bernard Kamoroff, C.P.A., Small Time Operator, Revised Edition, 1977, Bell Springs Publishing Company. Available from major bookstores. If you have trouble finding it write to the publisher at P.O. Box 322, Laytonville, California 95454 or have your book dealer contact Bookpeople, 2940 7th Street, Berkeley, California 94710.

BRINGING YOUR COMPUTER BUSINESS ON-LINE

Stephen Murtha
D/A Associates
3 Altarinda Dr.
Orinda, CA 94563
(415)-254-7100

Elliott MacLennan
MacLennan & Lillie
2855 Mitchell Dr. Suite 130
Walnut Creek, CA 94598
(415)-938-5120

Robert Jones
Interface Age
13913 Artesia Blvd.
Cerritos, CA 90701
(213)-926-6629

This is an abstract of a panel discussion which we have given to a large number of people considering entering the microcomputer business or have already entered, but are not beyond the infancy stage. We will examine some of the legal, tax, financial and tactical considerations which must be considered for the venture to be a success.

The first topic of discussion here is the form which the business should take. Should it be a sole proprietorship, partnership or a corporation? This question can only be answered intelligently by examing the legal and tax environment, the nature of the business and the way these factors interplay.

A microcomputer business can be one of three types; manufacturing, retailing or consulting. Each category has its own unique requirements in terms of capital, labor, etc. to run it. Consideration must be given to the number of workers needed to run the outfit. It must be determined if they can be hired, or if they will require an ownership interest in the business. In addition to this, the capital requirements of the business must be dtermined and the most appropriate funding secured.

There are four main business forms used by small firms today. They are the sole proprietorship, partnership, Subchapter S corporation and Subchapter C corporation. At this point lets briefly review the salient characteristics of these forms from a legal and tax point of view as there is no one form which is automatically best for a small business.

A sole proprietorship presents no real legal or tax complexities.

With the exception of some simple legal formalities such as registering with local or state government when operating under a fictitious name etc., a person may operate as a sole proprietor with relative ease. From a tax point of view, the sole proprietor simply reports any income less allowable business expenses as he would any other income on his tax return.

A partnership is not much different than a sole proprietorship except in the number of people involved. As with the sole proprietorship, there are no complex legal requirements. Even the partnership agreement may be oral. However, this is often a hidden trap. The nature of running a joint venture requires that all aspects of running the business such as share of profits and losses, capital and work contributions, etc. be agreed on in writing. Since there is no legal requirement that this be done, many partnerships neglect this step, only to be torn apart by disputes later on. For tax purposes the partnership pays no taxes itself, it simply acts as a conduit for income and the partners report their share of the income on their tax returns.

A Subchapter S corporation is a unique form of business created by federal tax law. From a legal point of view it is the same as any other corporation and must comply with all of the laws of the states in which it incorporates and does business. From a tax point of view it is very similar to a partnership in that the income flows on through the corporation to the shareholders without being taxed at the corporate level. This is especially useful in the start-up phase of a new business

where there is often a net operating loss as it can be passed on to the shareholders instead of being buried in the corporation. This tax treatment is for federal taxes only and the corporation can usually have no more than 10 shareholders in the start-up stages.

The final form is the Subchapter C or regular corporation. A corporation is very different from a partnership or sole proprietorship. State and federal laws impose a considerable number of formalities and regulations on corporations having to do with everything from bookkeeping, corporate minutes, to the sale, transfer of stock, etc. Since the corporation is legal entity separate from the owners, there are considerable tax and accounting considerations which far exceed those of other forms. However, this same separation leads to tremendous tax advantages in certain situations.

Raising the capital for your business requires a fair amount of work and planning. The first step is to make out a business plan. The business plan must include detailed projections of income and expenses,plant and equipment requirements, working capital, inventory, etc. for as far into the future as can reasonably be done. When completed, the well drawn business plan will provide a documented answer to any question an investor might have.

Considering that most people have never done this type planning before, help is often required. There are three main sources; your attorney, your accountant and the many federal, state, and local government agencies set up to aid small businesses.

One of the better of these is SCORE (Service Core of Retired Executives) which is part of the Small Business Administration.

Choosing the investment vehicle and the potential investors are closely related topics. Both the investor and the company have the same goal in mind, namely profitibility. However, the investor will want maximum protection of his investment while the company will want maximum flexibility in the running of the company. Often these two short-term goals will be in conflict and create problems.

There are two main ways of financing a business; debt and equity. All investment vehicles are a variation or a combination of both. Common and preferred stock are ownership vehicles and as such tend to dilute ownership and control. They pay no fixed interest or dividend. Debt, such as notes, bonds, etc. offer the investor a fixed rate of interest, but no ownership or share of the profits. Many debt instruments are convertible into equity instruments to overcome this limitation, particularly in high growth firms.

It is of critical importance to note here that the company must make sure that the proposed investment transaction complies with all securities law including both Federal and State. Serious penalities including civil and criminal await the company and its officers that fail to observe these regulations, even when dealing with friends and relatives or with very small firms.

The time and money invested in weighing these factors will be returned many times over and will free you up to make your business a success.

The material presented in this article is intended for the reader's general information. The authors request that the reader consult professional advisors prior to applying this material to his or her specific situation.

TOWARD A COMPUTERIZED SHORTHAND SYSTEM

W. D. Maurer, Professor
Department of Electrical Engineering and Computer Science
George Washington University
Washington, D. C. 20052

Abstract

By a computerized shorthand system is meant a system having the following components:

(1) a language of abbreviations for common English words;

(2) a microcomputer program which accepts English words as input, in either abbreviated or unabbreviated form, and prints them out in unabbreviated form; and

(3) a typewriter connected to a microcomputer, upon which is resident the above-mentioned program.

The purpose of a computerized shorthand system is to allow the user to take dictation "at speed." As the dictated words are spoken, the typist enters words and their abbreviations into the system, which types out words in unabbreviated form. The result is a one-stage system of transcribing, in contrast to the two-stage systems presently encountered.

Fundamental Design Considerations

The idea of computerized shorthand started very simply as follows: With microcomputers smaller than, and cheaper than, typewriters, what enhancement to the capabilities of a typewriter could be effected by imbedding a microcomputer within it? Perhaps if the user could strike the keys bz, and the typewriter would type out the word business -- or gm for government, and so on -- the capabilities of typewriters would be enhanced.

It is clear even from this minimal description of the idea that it resembles that of shorthand; so let us look at the existing systems of shorthand to see if they could be adapted for a microcomputer system. Ordinary (Gregg) shorthand as it is used today, of course, involves a number of symbols which do not appear on a typewriter. But there are several shorthand systems which involve alphabetic characters, such as ABC Shorthand [1] or Speedwriting [2].

A cursory glance at the principles of these systems, however, uncovers an immediate difficulty. The idea of any of the presently commonly used shorthand systems is to use a symbol to represent a sound or a collection of sounds. In many cases the same symbol is used for several sounds and thus the same combination of symbols is used for several words. Even were this not the case, however, the same symbol or combination of symbols would certainly be used, in such a system, for words whose sounds are the same (homonyms).

A microcomputer system, on the other hand, must be able to read a shorthand word and tell immediately what longhand form is represented by that word. (We have here one of the basic drawbacks of microcomputers when compared to the human brain.) Thus it is clear that a new shorthand system must be developed for this application.

Before describing what such a new shorthand system ought to be like, let us briefly examine another alternative to shorthand, namely stenotyping. This involves a machine having a keyboard like a typewriter, but with many fewer keys. Any combination of keys may be struck at once, and each key that is struck causes a letter to be printed on a roll of paper tape. At a later time, this tape can be transcribed to ordinary English. The fact that combinations of keys may be struck is a distinct advantage of this system; it allows recording at 240 words per minute or more to be done, whereas ordinary typing takes place at 70 words per minute or so.

The code which is used in stenotyping, like shorthand, is a sound-based system. Not every letter of the alphabet is represented by a single key on the machine; those which are not are represented by combinations of keys. However, in all cases, it is the sound of the letter, rather than its written form, which is represented. A serious drawback of stenotyping is the time it takes -- normally two years -- to learn the code.

Can stenotyping be adapted into a computerized shorthand system? First of all, one would have to devise special codes for the various homonyms. The drawback mentioned in the previous paragraph now becomes an advantage; if you have already spent two years learning the code, a few more months relearning new codes for homonyms do not represent appreciable extra effort. This has, in fact, been done, and there are computer systems which perform the transcription (see, e. g., [3]).

In practice, stenotyping is not used in the great majority of business applications involving the taking of dictation. This is undoubtedly due to the amount of time it takes to learn the code and the attending scarcity of (and, therefore, high salaries paid to) people who know the code. We are, on the other hand, concerned with a system which can be used in business applications, much as word-processing systems are used.

The fact that a system such as we are describing must be word- (and possibly phrase-) based, rather than sound-based, presents us with two immediate problems. One is that, obviously, not every English word can be separately represented by a shorthand form -- the dictionary is too big for that. The other is that learning shorthand for words is obviously quite a bigger chore than learning shorthand for sounds, because there are a lot more words than there are sounds.

The solution to the first of these problems is to represent only the most common words by shorthand forms. Any word that is not represented by a shorthand form can simply be typed out in longhand. Thus the computer is always either in shorthand mode, in which it is translating shorthand into longhand, or in longhand mode, in which it is simply copying out longhand.

Surprisingly, this turns out to be the solution to the second problem as well. Since any word (even a word which has a shorthand form) can be typed and printed out in longhand form, it is possible to use a computerized shorthand system with only a bare minimum of knowledge. One learns the first few shorthand forms and then starts typing, using only these shorthand forms, and typing everything else out in longhand. As one learns more shorthand forms, one's shorthand percentage (and, therefore,

typing speed) increases. Thus anyone who has mastered ordinary typing skills can expect a slow but steady increase in typing speed attendant upon the learning of more and more shorthand forms. Mistakes in shorthand are minimized by the constant application of a cardinal rule: if you are not absolutely sure of the shorthand form in a given case, use longhand.

Design Details

Let us now look into the details of shorthand, treating the broadest outlines first. What is the most commonly used key on a typewriter? Obviously the space bar. This key is used so commonly, in fact, that in shorthand we would like to see if we can get away with not keying it at all. If every shorthand form were exactly three characters long, for example, hitting the space bar would never be needed; the computer program would simply read three characters at a time, look up the corresponding word, and then print that word, together with a space, either before or afterwards. A little thought convinces us that the space should be typed beforehand, not afterwards, since words are sometimes followed by periods or commas.

Such a system, however, is much too inefficient. As long as no shorthand form is an initial substring of any other shorthand form, the computer can tell where a shorthand form ends. The most commonly encountered words may be represented by single keystrokes. Any key which does not represent a word in this way can be the first character of a two-character code. Any two-character combination of this kind which is not a two-character code can then be the first two characters of a three-character code, and so on.

Another immediate problem is now apparent: How do we know which words are the most commonly encountered? Some are obvious -- "the," "and," "of," and so on -- but when we start to face the question of whether a particular word such as "this" or "that" should be represented by a one-character code or by a two-character code, it is clear that something more scientific is required. Fortunately, the frequency analysis of English words is a well-known subject, and many such analyses have been done. Unfortunately, almost all of these analyses were done in connection with the teaching of children, and the materials analyzed were children's materials.

One frequency analysis, however, namely that of Kucera and Francis [4], satisfies our requirements. Besides having been constructed from writings meant for adults, it is a computation-al analysis which avoids the pitfalls of, on the one hand, counting all forms of a word as the same word (see [5], for example) and, on the other hand, distinguishing between various forms of capitalization, so that Time, time, and TIME are counted as three different words, each with its own frequency (see [6], for example). In the particular case discussed above, Kucera and Francis tell us that "that" is considerably more common a word than "this." (This is plausible; we speak of "this table" and "that chair" but we also say "I know that he will come," and so on, so that "that" has two common meanings whereas "this" has only one.) In our system, "that" is represented by a one-letter code (x), whereas "this" is represented by a two-letter code (ts).

In order to see in a bit more detail what such a system should be like, let us, for the moment, ignore the codes of three or more letters. It is clear that there are several hun-dred possible two-letter codes, which already take quite a while to learn, so that the form of the overall scheme is determined to a great extent by the form of the typical two-character code. It seems natural that the typi-cal two-letter code should begin with the first letter of the corresponding longhand form. What does that leave us for one-letter codes? There are, first of all, the digits; then there are the letters (such as x) which are not the first letters of many common words; and finally there are the punctuation symbols. However, at least some of the punctuation symbols, particularly the period and the comma, are common enough that they ought to represent themselves; one ought to be able to produce a comma by typing a comma.

Whatever a punctuation symbol represents ought to be much the same, in most instances, whether the compu-ter is in shorthand mode or longhand mode. However, there are two important exceptions. In longhand mode, typing a space signifies a space between two longhand words; in shorthand mode, however, spaces are not typed. Simi-larly, hyphens can appear, and often do appear, in longhand forms, where they do not (or at any rate not near-ly as much) in shorthand. Thus the

space bar and the hyphen may be used as single-character codes. Similarly, the ten digits and the letters j, k, q, u, v, x, y, and z are used in our system as single-character codes. The codes and their corresponding longhand forms are as follows:

2	to	j	be
3	the	k	he
4	for	q	at
5	it	u	on
6	is	v	of
7	as	x	that
8	a	y	by
9	in	z	his
0	was	blank	and
1	I	minus	with

It should be noted that the word "I" is, according to Kucera and Francis, the 20th most common English word; it has been assigned the single-character code "1" and an alternate code "ij" as well, for use on typewriters which do not possess the "1" key. Of the other 19 representations, some of them (2, 4, v) are strongly mnemonic, while others (3, 8, 9, y, z) are slightly mnemonic.

Any word that begins with j, k, q, u, v, x, y, or z, and that has a short-hand form, has one beginning with some other letter. In our system, the two-character forms of this kind are as follows:

fy	very	ir	your
gb	job	iu	you
gi	kind	iy	yet
gj	John	oa	usually
gk	keep	od	used
gq	quite	oe	under
gs	just	og	understanding
gu	knew	oi	united
gw	know	oo	upon
ie	year	op	up
ig	young	os	us
ik	York	oz	use

Note that f substitutes for v, o for u, i for y, and g for any of j, k, q, x, and z, as the first letter of a short-hand form. All other two-character shorthand forms start with the first character of the corresponding longhand. (The last two forms above are interes-ting. The form "ik" is marked for obso-lescence in a future version of the sys-tem; Kucera and Francis do not analyze phrases, but it seems obvious that "York" -- in this country, at least -- almost always occurs as part of the phrase "New York." The word "use" is an example of the opposite of a homonym,

that is, we have two words that are spelled the same but have different pronunciations -- "Use this tool" and "Put this tool to good use," for example -- and this causes no difficulty in the system at all; the same shorthand form, oz, stands for both words.)

Let us now ask where our forms of three characters and more are going to come from. If we stick to the rule, implied by what we have said above, that a two-character code is always actually a two-letter code, then it follows that three-character codes of the form (letter, digit, letter) are always permissible. This gives us a wide variety of possible three-letter codes. Furthermore, since all words contain vowels, and since there are only six different vowels, we can let the choice of digit in such a form specify the choice of vowel. In fact, in our three-character codes, we use 2 to specify "a," 3 for "e," 4 for "o," and 5 for any of the vowels "i," "u," and "y," all of which are less common.

Let us now discuss forms of a word. It will clearly not do to type a word followed by s, for example, to denote a plural, because the computer will take the s to be the start of the next word. However, it would also clearly be advantageous to be able to follow any shorthand-representible word by any of the three commonest endings -- -s, -ed, -ing -- by striking only one key in addition to the shorthand. The solution we have adopted is only slightly difficult to learn: we use the key 6 for -s, 7 for -ed, and 8 for -ing, to be struck as the second character of any shorthand form. It should be clear that this does not conflict with any of the other rules outlined above.

We have to have a way of getting into longhand mode from shorthand mode, and vice versa, and the character we use for this purpose is the slash. The sentence above, for example, may be rendered in shorthand by

```
/We have/2/have/8/way/v/getting
 into longhand mode from
 shorthand mode,/ /vice versa,
 / 3/character we/oz4ts/purpose
 /63/slash.
```

This is of course only one possible shorthand rendering, and a "minimal" one at that; it uses only the shorthand forms that have already been introduced in this paper. From an ori-

ginal sentence of 144 characters we have produced a shorthand sentence of 125 characters, obtaining a character count reduction of 13.2% and a typing speed increase (assuming a constant number of characters per second) of 15.3%. In this connection it should be noted that any shorthand form is to be considered as learned only when the typist can produce it in context at an unreduced character speed (without stopping, however momentarily, to think of what the shorthand form is).

If we were to represent the same sentence, using as many shorthand forms as possible in the abbreviation language we are using at the time of this writing, we would obtain:

```
wehv2hv8wyvg8eioln;;hn/mode/
fm/short;hn/mode,/ /vice versa,
/ 3c3cweoz4tsp5p63/slash/.
```

This shorthand rendering is to be interpreted as follows:

we	we	;	(see below)
hv	have	hn	hand
2	to	blank	and
hv	have	blank	and
8	a	3	the
wy	way	c3c	character
v	of	we	we
g8e	getting	oz	use
io	into	4	for
ln	long	ts	this
;;	(see below)	p5p	purpose
hn	hand	6	is
fm	from	3	the

This time, the shorthand sentence is made up of 85 characters, a character count reduction of 41.0% giving rise to a typing speed increase (again assuming a constant number of characters per second) of 69.4%.

It must be noted that there are actually two ways of getting from longhand mode into shorthand mode, or vice versa, the other one being the semicolon. This has the effect of not inserting a blank; thus "short/hn" (with the slash) corresponds to "short hand" (two words) whereas "short;hn" (with the semicolon) corresponds to "shorthand" (one word). When we have a word, such as "longhand," which is made up of two words ("long" and "hand") each of which has its own shorthand form (ln and hn, respectively), we can use the semicolon twice, once to get into longhand mode and another time to get back into shorthand mode, without inserting a blank either time.

Semicolons are used mainly for prefixes and for suffixes other than -s, -ed, and -ing. We have learned above that p5p represents purpose; p5p;ful is therefore purposeful. Similarly, hv is have, and so be;hv is behave.

It should be clear that an abbreviation language of this kind is extremely tolerant of the user with incomplete knowledge. If one forgets, for example, that the word "short" has the shorthand form s4t -- as this author did when typing out the above -- it doesn't much matter; the typing speed increase obtained is only 69.4% instead of 73.5%. This is in marked contrast to the situation in natural languages, where, for example, not knowing the German word for "telephone" can be quite serious in Munich.

Let us now consider the period. Most periods are at the ends of sentences, but many are not (as in a person's initials, for example). In our system, any period typed in shorthand mode is assumed to be at the end of a sentence; any period typed in longhand mode is not. In particular, when a period is typed in shorthand mode, it is automatically followed by two blanks (reflecting standard secretarial practice), and the next word is automatically capitalized. This is why the second form of the shorthand sentence above starts with a small w; it is assumed that the preceding sentence ended (as this one does) with a period typed in shorthand mode. A period typed in longhand mode simply appears as a period, without any side effects.

What if the user needs a slash or a semicolon in the printed output? Our solution to this problem is to use a standard punctuation key, which is the key immediately to the right of the P on a typewriter (whatever symbol might appear on that key on an actual keyboard). Typing this key (which we shall refer to as ½ since that is the symbol on our own trusty Selectric) followed by / or ; or, for that matter, by any punctuation character which appears in lower case on the given typewriter, causes that punctuation character to be printed out.

For punctuation characters which appear in upper case (as, for example, the ones above the digits, or the colon) we have the standard capitalization key. This is the key to the immediate right of the semicolon, which, as before, we shall refer to as ' from the character on our own typewriter.

Typing ' before any key produces the capitalized version of the key; this includes the letters, so that 'a produces A, for example. Typing ½a, on the other hand, produces the ampersand, and, in general, typing ½ followed by a letter produces a punctuation symbol whose name (usually) starts with that letter or is otherwise mnemonically related to it, according to the following table:

½a	ampersand
½b	left bracket
½c	cents sign
½d	dollar sign
½e	equal sign
½f	one fourth
½g	right bracket
½h	one half
½i	' (single quote)
½j	" (double quote)
½k5	***** (5 asterisks)
(5 may be replaced by 2, 3, ..., 9)	
½l	left parenthesis
½m	minus sign
½n	number sign (#)
½o	colon
½p	plus sign
½q	question mark
½r	right parenthesis
½s	star (asterisk)
½t	at (@)
½u	underscore

Learning this table -- or at least part of it -- makes it easier for the user to switch from one typewriter keyboard to another in which some of the characters may be in different positions.

The shift key can also be used, either in shorthand or longhand mode. In longhand mode, it simply causes capitalization as usual; in shorthand mode, this is true only for punctuation and for the first letter of a shorthand form; any other use constitutes an error. It is less efficient to hold the shift key down while typing something else than it is to use the capitalization key; the use of the shift key is retained, however, both because people often use it out of force of habit and because some typewriters have no key to the right of the semicolon.

The carriage return key is used to end paragraphs only. The program has stored a maximum number of characters per line, and calculates where old lines end and new lines begin. (The standard word-processing functions of right justification and hyphenation may be added to a system such as ours, as optional extras.) It may be noted that, in coun-

ting characters in our original long-hand sentence, we have omitted the four carriage returns and the hyphen (in the word "character") that would not be typed if the system were used. If these are included in the calculations, the typing speed increase rises, in the best case, from 73.5% to 79.5%. When a carriage return is typed, a new line is started, blanks are printed (we print six) and the first character of the first word of the next paragraph is marked for capitalization.

List Of Two-Character Codes

We now list the two-character codes which we are using, in an order that makes them reasonably easy to learn.

(1) The following two-letter words represent themselves in shorthand:

    PRONOUNS   me, my, we
    ADJECTIVES   an, no
    VERBS   am, do, go
    CONJUNCTIONS   if, or, so

In addition, "up" and "us" are represented by op and os, as noted above, since u all by itself represents the very common word "on."

(2) The following words are represented in shorthand by their first two letters (thus bo represents "body," ch represents "children," and so forth):

    NOUNS   body, children, course, data, education, effect, end, equipment, experience, form, idea, interest, life, mind, Mr., nature, night, part, rate, right, road, state, will
    PRONOUNS   him, it's, our
    ADJECTIVES   all, big, black, certain, each, even, far, full, great, human, last, least, low, more, old, one, own, right, same, such, three, two
    ADVERBS   away, even, here, too, where
    VERBS   are, came, did, end, find, form, gave, get, had, last, may, must, obtained, own, put, read, run, see, should, state, will
    PREPOSITIONS   about, after, behind, during, near, off, over
    CONJUNCTIONS   but, either, however

(3) The following words are represented in shorthand by their first and last letters (thus aa represents "area," bd represents "board," and so forth):

    NOUNS   area, board, being, back, boy, club, can, car, city, data, death, door, days, day, evidence, field, fact, general, group, head, half, help, heart, history, leadership, law, man, membership, Mrs., nothing, nations, place, problem, point, room, return, something, states, thought, tax, world, way
    PRONOUNS   her, I'll, I'm, its, myself, they, who
    ADJECTIVES   all, any, both, better, dead, done, dark, English, every, free, following, four, few, good, general, high, less, likely, most, north, natural, public, red, real, recent, small, these, this
    ADVERBS   back, ever, generally, here, later, now, nearly, only, probably, rather, really, then, too, well, when
    VERBS   being, began, brought, could, can, cannot, done, doing, don't, going, given, got, having, help, has, looked, let, making, need, not return, said, see, saw, say, thought, went
    PREPOSITIONS   around, among, against, following, from, into, off, out, toward
    CONJUNCTIONS   how, nor

(3) The following words are represented in shorthand by their first and next-to-last letters (thus cc represents "church," cg represents "change," and so forth):

    NOUNS   church, change, case, home, hand, light, name, present, period, past, research, state, time, type, wife
    PRONOUNS   himself, what, which
    ADJECTIVES   another, best, early, east, first, five, left, large, light, little, long, much, next, past, some, their, what, which
    ADVERBS   again, also, once, there
    VERBS   become, believe, change, come, didn't, felt, found, give, have, left, like, made, might, make, move, present, state, take, told, type, were
    PREPOSITIONS   above, down, like
    CONJUNCTIONS   than, though

(4) The following words (in addition to the four-letter words in the preceding list) are represented in shorthand by their first and third letters (thus cu represents "country," eg represents "England," and so forth):

    NOUNS   country, England, increase, labor, level, objective, power, report, river, service, table, woman
    ADJECTIVES   different, economic, expected, important, local, limited, major, military, necessary, other, objective, social, technical, these
    VERBS   called, develop, expected, require, would

PREPOSITIONS along, before, except

CONJUNCTIONS because, since

(5) The following words are represented in shorthand by their first letter and some other letter in the given word, or "z" where the word contains the sound of z, as indicated:

NOUNS anything (ah), business (bz), community (ci), company (cp), development (dl), department (dm), example (em), everything (et), eyes (ez), government (gm), husband (hb), headquarters (hq), hands (hz), information (ia), individual (iv), lines (lz), members (mb), means (mz), number (nb), newspaper (np), needs (nz), president (pd), program (pg), project (pj), people (pp), problems (pz), reason (rs), result (rz)

PRONOUNS themselves (tv)

ADJECTIVES American (ac), close (cz), difficult (dc), developed (dp), enough (eu), historical (hc), national (nn), possible (pb), professional (pf), political (pi), religious (rg), those (tz)

ADVERBS always (az), further (fh), frequently (fq), perhaps (ph)

VERBS became (ba), close (cz), developed (dp), does (dz), provide (pv), reached (rh)

PREPOSITIONS between (bw), inside (ii), through (tu)

CONJUNCTIONS although (au), therefore (tf)

(6) Finally, there are the following exceptions (somewhat analogous to the way we pronounce busy "bizzy," etc.):

NOUNS art (aj), cut (cj), college (cx), defense (dx), feet (ff), family (fx), God (gx), house (hx), love (lu), men (mm), office (ox), product (pk), set (sj), school (sk), top (tj), women (ws), war (ww)

PRONOUNS I (ij), itself (ix), she (sz), them (tq)

ADJECTIVES many (mx), new (nu), per (px)

ADVERBS almost (ax), why (wq)

VERBS asked (aq), been (bx), cut (cj), drive (db), look (lq), love (lu), set (sj)

PREPOSITIONS without (wz)

CONJUNCTIONS while (wx)

Each of the above lists is in alphabetical order of the shorthand forms involved, even though this may not be the alphabetical order of the corresponding longhand. Homonyms can be easily seen to have different shorthand forms; thus we have "there" (tr) and "their" (ti), etc.

If a word (such as "English") has a shorthand form and begins with a capital letter, the capitalization is automatically supplied. Note that a few words have such common plural forms that the plural has a shorthand form all its own; thus we have "day" (dy) and "days" (ds). In other cases a form of a word will be much more common than the word itself, so that, for example, "members," "reached," and "frequently" have two-character shorthand forms, whereas "member," "reach," and "frequent" do not.

There are many duplications in the above lists; thus "end," "state," and "will," for example, are both nouns and verbs, while "all," "off," and "see" appear both in paragraph 2 (first two letters) and paragraph 3 (first and last letters). Occasionally a word will have a two-letter code, not because it is particularly common, but because that code does not seem to fit any common word; "leadership" (lp) and "headquarters" (hq) are examples of this.

Summary

A computer-aided shorthand system has been demonstrated to be feasible. Details of the actual computer system which implements the shorthand translation as described here are deferred to a subsequent paper.

References

1. Smith, Joseph W. H., Jr., ABC Shorthand, Smith Business School, Washington, D. C.

2. ITT Corp. Speedwriting Series, Principles of Speedwriting, Bobbs-Merrill, Indianapolis, Ind., 1977.

3. Koomanoff, L. G., and A. J. Gasdor, Computer Compatible Machine Shorthand For Expanding Careers, Stentran Systems, 380 Maple Av. W., Vienna, Virginia, 1973.

4. Kucera, H., and W. N. Francis, Computational Analysis of Present-Day American English, Brown University Press, Providence, R. I., 1967.

5. Thorndike, E. L., and I. Lorge, The Teacher's Word Book of 30,000 Words, Teacher's College, Columbia University, New York, 1944.

6. Carroll, J. B., P. Davies, and B. Richman, The American Heritage Word Frequency Book, Houghton Mifflin Co., Boston, 1971.

MICROCOMPUTER APPLICATIONS IN COURT REPORTING

Douglas W. Du Brul, BSEE
5681 Mary Lane Drive
San Diego, California 92115
Phone: 714/583-3733

This paper describes a need for a set of compatible word-processing systems for use by court reporters and attorneys. By taking a "systems engineering" approach to the design of an integrated set of word processing equipments it will be possible to simultaneously improve the transcript production process for court reporters and reduce the cost of computerized data searching for attorneys. The key to potential improvement is digitization of the data as early as possible in the transcript preparation cycle.

This is a systems level article. It describes an applications area for small word-processing systems, describes the manual process to be replaced, indicates possible functional improvements which may open up new markets, and presents several design concepts for equipment to meet the identified needs. Technical considerations are not addressed in detail.

Documentation support advantages of early digitization include the transfer of data over telephone lines, CRT text-editing prior to any printing, compact record storage, and low-cost document reproduction. And if digitization takes place at the source data level (as shorthand is being used to record testimony), it is even possible to computerize the translation of the shorthand symbols into plain language.

The applications area being addressed is a very good potential market area for small computing systems. Almost all of the court and deposition transcripts are produced by "independent" contractor" typists (working at home). Amazingly this slice of highly dispersed free enterprise accounted for about one billion dollars worth of documentation in 1977. That dollar amount was obtained by taking estimates of transcript-related income for the reporters in San Diego County, California, reducing it to dollars per area resident and then extrapolating that value to the population of the entire country.

Roughly 25% of the billing cost of transcripts represents direct expenses incurred in the actual production of the documents (equipment, typing labor, duplication, binding and distribution). In addition to these traditional and presently funded activities, there exists a potential market for equipment and services in the "litigation support" category which could involve both attorneys and court reporters.

Cooperation between a court reporting firm and an attorney client can result in a reduction in the need for capital investment by the attorney. A reporting firm which uses a computer in the preparation of transcripts can make computer searches for key words and phrases identified by clients. The results of the searches can be presented in the form of an index to the transcript pages on which the words and phrases appear. The reporting firm could also make digital tape recording of transcripts so that it would be possible to re-enter the data into the computer to make further searches at a later date should it become desirable to search for other key words and phrases.

The service outlined in the previous paragraph will have special appeal to small law firms and attorneys who do not have computing equipment. A similar service may also be appractive to law firms having computers. To indulge in computer searching of transcripts it is necessary to first digitize the text of interest. This digitization is frequently accomplished by use of an optical character recognition (OCR) unit, frequently referred to an an optical scanner. These OCR units are capable of reading typed pages, provided certain type fonts have been used, and placing the data in a computer in proper digital form. The need for these expensive pieces of equipment (about $14,000 to $32,000) might be avoided by having a reporting firm provide a tape recording of the document in proper form. If the reporting firm uses a computer in

transcript preparation, the digital data will exist in the reporting firm's computer and it would be a simple matter to make a tape recording of it for delivery to a client.

Much spade work has already been done in applying computers to court-related tasks, but there is still room for innovative thinking. There is a real need for adapting small general-purpose computers to tasks presently being done by large time-sharing systems or dedicated computers which are integral parts of single-function equipment. This trend will probably not significatly impact on the need for time-shared services or special-purpose equipment, but it will bring the power of the computer to more small businesses - and with no compromise in the quality of the product or service provided.

Existing Methods

The present manual process for producing court transcripts starts with a court reporter who records the original testimony in shorthand. Most reporters now use shorthand typewriters (stenotype machines), but some reporters are still "penwriters". When an order is received for a transcript, it is necessary for someone to "transcribe" the shorthand notes and type up a formal document.

Several approaches are used to transcribe shorthand notes. The most frequently used method involves having the reporter read the notes and dictate into a dictating machine. The dictation tapes are then given to a transcriber who types up the finished document, which is then returned to the responsible reporter for proofing prior to delivery to the client. Alternatively, the court reporter might deliver the shorthand notes to a "note reader" (a transcriber who is also skilled in reading shorthand) who will type the transcript directly from the shorthand notes. The note reader relieves the reporter of the dictation task, but charges more for note reading than for typing from dictation tapes (about twice as much). A third method for manual production of transcripts is for the reporter to do the typing directly from the shorthand notes without involving a third party. This approach is too time consuming for reporters who carry a normal work load.

The documentation methods mentioned in the previous paragraph are all manual methods. These have been in use for many years, but they are now being challenged by computer-aided transcription. Computer-aided transcription (CAT) services are available commercially, but it is not yet possible to purchase CAT software. Various systems are under development using a number of approaches to preparing data for entry into the computer. Presently available commercial CAT systems all use the input device shown in Figure I (or a male counterpart). The basic analog-to-digital converter is still the human brain, and interfacing to the computer system is accomplished with an encoder installed in a stenotype machine, which sense the operator's "keystrokes". All-electronic voice digitizers exist, but they work best when fully optimized for one person at a time.

Figure 1, Adaptive analog-to-digital converter for computer-aided transcription systems.

CAT systems constitute a threat to the traditional court-reporting community because CAT systems can produce computer-generated indexes to key words and phrases. This service cannot be duplicated manually without an inordinate expenditure of time and effort. Since there are several good reasons for small reporting firms to avoid an immediate involvement with CAT equipment it becomes very attractive to consider the use of small word-processing systems which enable manual typists to digitize the data while producing hard copy in the traditional manner. The digital record can then be delivered to a reporter client having computer facilities for making the required data searches.

Computer-aided transcription is too big a subject to be covered in detail in this paper, but a quick overview is in order. Commercially available CAT services at present are of two general types. Baron Data Systems of Oakland, California provides a

complete "stand-alone" system under a lease arrangement. The equipment provided is all that one needs to transcribe shorthand records to plain language, to edit the text on a CRT, and to print the final document. Stenographic Machines, Inc., of Skokie, Illinois, provides a service which requires a telephone connection with their host computer. This is typical of the service offered by several other CAT service firms also. When the shorthand-to-plain language translation is performed by a remote computer, the customer usually has to have text-editing and printing facilities, although arrangements can usually can be made to have hard copy returned by mail.

Although CAT seems to be gaining in popularity, there are a number of reporters who have reservations about becoming involved. Some fear that they are not "computer compatible". Others object to the cost, which is about the same as for a note reader's services. However, the per-page cost of CAT-produced copy decreases as the number of pages per unit of time is increased. Until very recently CAT was considered as just another way of producing a transcript, but that picture changed when the CAT service firms started to offer computer-generated indexes to key words and phrases.

The most obvious reason for the limited use of CAT is the lack of commercially available software. Apparantly the owners of CAT software see a greater return on their investment in the sale of CAT services than in the sale of CAT software. Speaking as one who has researched the task of writing CAT software, I can sympathize with those who have developed acceptable programs. The program structure is basically simple, but the work involved in building up the master dictionary is a discouraging prospect. Hopefully someone with a CAT capability will start to provide a service in which the shorthand notes will be read optically and the first-pass computer output will be returned in the form of digital tapes. The responsible reporter could then do the text-editing, proofing and printing. This approach would appear to be very satisfactory between now and the time CAT software becomes available at a reasonable price.

New System Concepts

The system of Figure 2 is a simple typing station capable of producing a magnetic tape recording and hard copy simultaneously. Input to this station will be dictation tapes or shorthand notes, depending on whether the operator is a transcriber or a note reader. Preferably the electronics should be capable of processing simple error-corrections, entered at the keyboard, so as to produce an error-free magnetic tape recording. Buffering of several pages of text will be required. It should be possible to print final copy a page at a time under computer control while the recently typed-in text is still in the buffer memory. The output tape format must, of course, be compatible with all "downstream" equipment. It may be necessary to include a capability for producing magnetic tapes to more than one recording format (and physical size) if this station is required to feed several mutually incompatible systems.
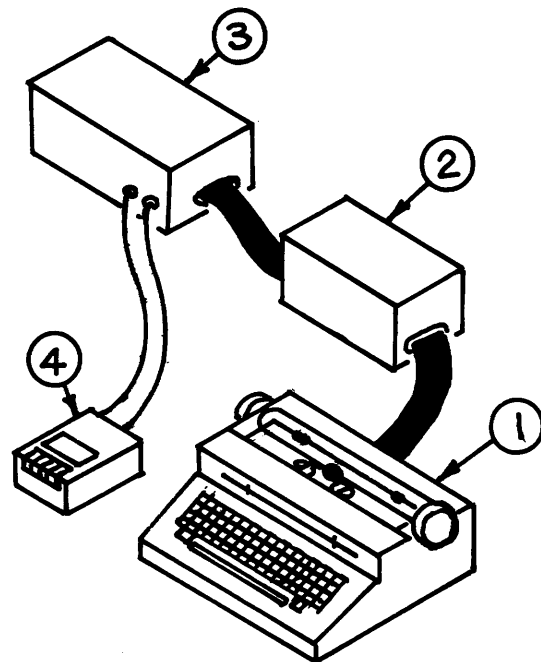
Figure 2, A simple transcriber's station capable of capturing typewriter keystrokes while producing hard copy.

1. Typewriter.
2. Typewriter/computer interface unit.
3. Microcomputer.
4. Tape recorder.

In court reporting the reporter who does the initial reporting is responsible for the accuracy of the final document, so it is common practice for them to proofread at home. When digital recordings become available, the reporters will need a low-cost text-

editing station similar to that shown in Figure 3. The input to this station will be the output of the system shown in Figure 2 or the unedited output of a computer-aided transcription service. The system should be capable of reading a block of data into computer memory from where it can be called up for display on the CRT for editing. Edited text is to be recorded on the output recorder.
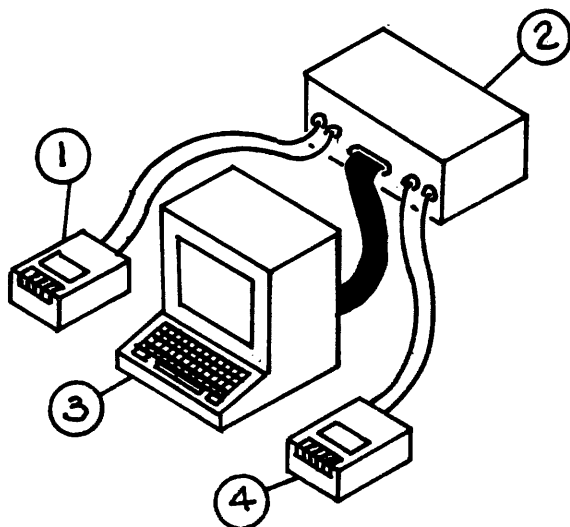
Figure 3, A minimal court reporter's text-editing station.

1. Input tape reader
2. Microcomputer.
3. CRT editing terminal.
4. Output tape recorder.

The system shown in Figure 4 is a full-featured reporter's station. It is intended that this station be capable of all of the operating features of the systems shown in Figures 2 and 3 and, in addition, be capable of a number of other special word-processing manipulations. It would be highly desirable, though not absolutely necessary, that this system be capable of assigning page numbers, indexing of key words and phrases, and computation of "folio counts" (the number of blocks of 500 characters). The folio count is used for billing purposes. It is likely that this system will need a dual floppy disk drive in addition to the equipment shown in Figure 4. The output of this system will be the final hard copy and, if required, digital tape recordings for use as a record storage medium, or for delivery to a client as an additional "product".
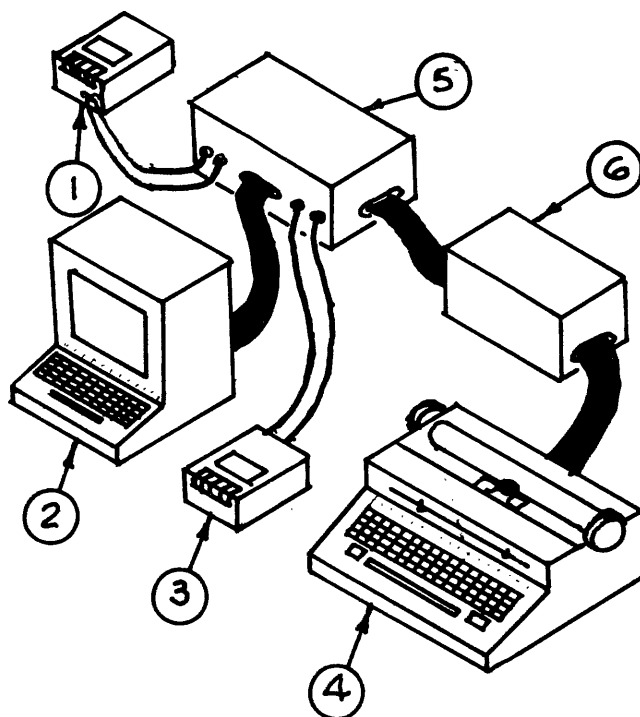
Figure 4, A full-featured court reporter's system.

1. Input tape reader.
2. Manual input or text-editing station
3. Output tape recorder.
4. Printer.
5. Microcomputer.
6. Printer interface unit.

Figure 5 is a simple system intended for use by an attorney in searching a transcript for which a digital tape recording is available. This system should be capable of simple searching for key words and phrases. A more elaborate version of this system would be required for long documents. A dual floppy disk drive would not only increase the processing capability but would make it possible to accept floppy disk recordings as input.

Figure 6 illustrates a system which is functionally equal to that of Figure 5 except that it assumes the availability of a "smart" terminal capable of performing the required functions without the need for a separate microcomputer unit.
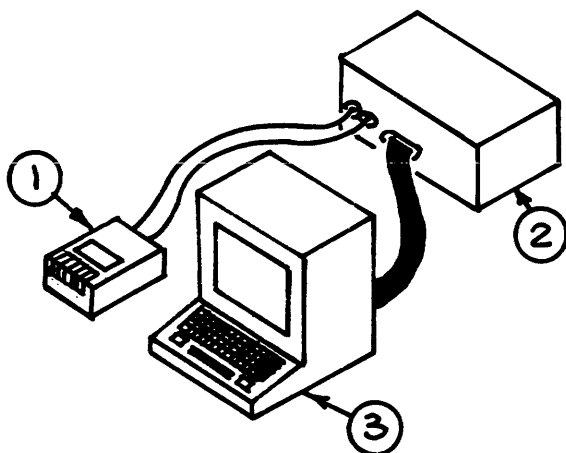
The design concepts presented in
this paper are being refined by the
author and several associates, primarily
at the concept level. Suggestions from
persons aware of equipment or software
potentially capable of supporting any of
these system concepts will be greatly
appreciated.

Figure 5, An attorney's document-
searching station using a "dumb" ter-
minal.

1. Input tape reader.
2. Microcomputer.
3. CRT terminal.
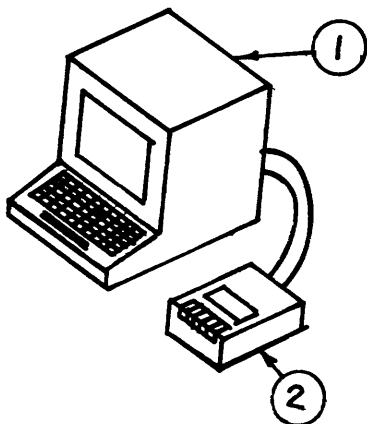
Figure 6, Attorney's document-
searching station using a "smart"
terminal.

1. "Smart" terminal.
2. Input tape reader.

Figure 7 is a conceptual sketch of
a unit that could be developed for use
in both the court reporting and attor-
ney communities. It could be supplied
with or without a printer or floppy
disk unit, depending on the application.
In this way it would be possible to
shave cost where necessary while making
it possible to upgrade the system with
standardized modules at a later date.
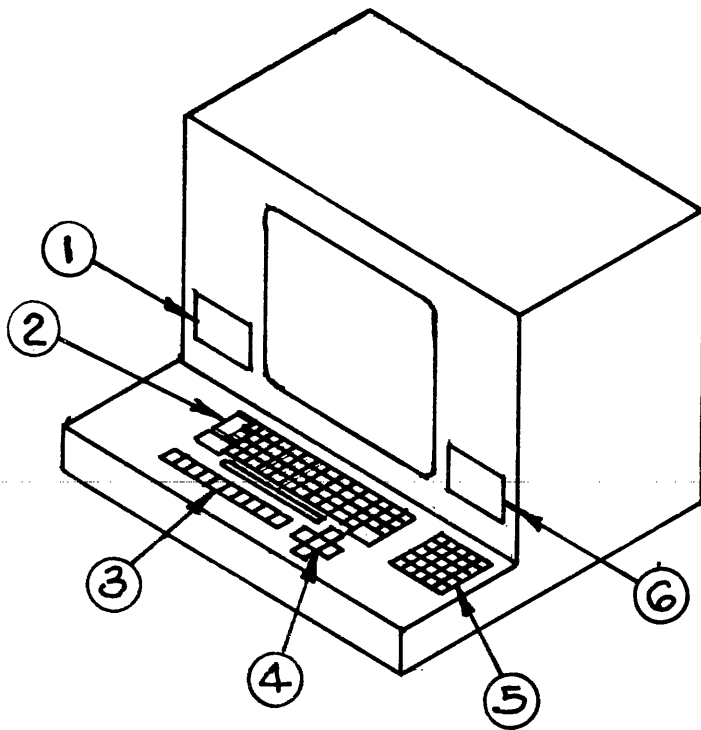
Figure 7, An integrated "court-support" terminal.

1. Input tape reader/recorder.
2. Output recorder.
3. Computer control keys.
4. Cursor controls.
5. Number pad.
6. Output tape recorder.

REAL TIME HANDWRITTEN SIGNATURE RECOGNITION

Kuno Zimmermann, Dept. of Electrical Engineering

Lafayette College, Easton, Pa. 18042

Abstract

Problems of computer security and entry control have over the last few years been approached in various ways; fingerprint recognition, handwriting pressure patterns and/or accelerations recognition, to mention just a few. Handwritten signature analysis appears to be reasonable in terms of recognition effectiveness, required capital investment and computational load. This contribution will present an introduction to the state of the art in handwritten signature recognition as evidenced by patents held by various corporations and by recent publications on both signature recognition algorithms and signature input devices. A basic system consisting of a simple graphic tablet, an A to D converter and a microcomputer will be introduced, as an example of the possibilities available to today's microcomputer user. Experience acquired on such a system comprised of a modified Intel SDK-80 linked to an HP-3000 will also be discussed.

Introduction

The need for an accurate, fast and low cost personal identification system has increased steadily in our computerized society. Possible applications for such a system are in computer memory protection, security access, entry control, etc. This identification problem has, over the years, been approached in various ways: fingerprint recognition, voice print recognition, handwriting pressure patterns and/or accelerations, to mention but a few. Handwritten signature analysis appears to be reasonable in terms of recognition effectiveness, required capital investment and computational load. With this in mind, a systematic search of patents and literature in this field was undertaken. An experimental signature analysis system was built to test some of the approaches proposed, as well as some other ideas.

Dynamic Signature Recognition Concepts

A handwritten signature is essentially a 3 dimensional trajectory, where the x and y axis should be taken as corresponding to the usual left to right and top to bottom displacement of normal handwriting, and where the z axis is taken along the direction of displacement or pressure into the writing surface.

Assuming the availability of means to translate such 3 dimensional components into a form suitable for computer processing, what sort of features should one be looking for to attempt recognition? Such questions and their answers were pondered more than 50 years ago by graphologists. To quote from Downey's Graphology and the psychology of handwriting [1]: "The existence of graphic individuality, often of a very pronounced type, will hardly be questioned." And later on: "The mathematical probability of two complete handwritings being identical is one in something more than 68 trillions". And to quote from Saudek's Experiments with graphology [2]: The pressure is among those features of the handwriting which are very difficult to alter, and is therefore in a high degree individual and characteristic." This second quote, in particular, encourages one to investigate the pressure patterns of a signature process. A quick glance at the literature along this idea reveals a wealth of patents and publications on both pressure input devices and processing methods. A patent assigned to Veripen,Inc., [3], for instance, describes an input device which is basically a capacitor consisting of two parallel plates separated by an elastomeric material. A writing stylus exerting pressure on a writing surface resting on the upper plate will change the spacing of the plates and thus introduce a capacitance change related to the pressure. Three patents assigned to Burroughs Corp. [4] [5] [6] describe input devices consisting of a handwriting surface suspended in space on a cantilever arm [4] or on rubber support members [5],[6]. Pressure variations induced by handwriting are translated into a strain in the cantilever arm [4] or a change in magnetic coupling between various coils traversed by a magnetic rod rigidly attached to the handwriting surface [5],[6]. Two further patents assigned to Burroughs Corp. describe respectively another plate-like handwriting surface resting on two elliptically shaped members where the pressure is related to the strain induced in these members and picked up by strain gages [7], and a force responsive transducer fixedly mounted within the housing of a writing instrument such as a ball point pen [8]. An implementation similar to the latter is found in a patent assigned to the Mosler Safe Company [9].

The recognition algorithms proposed in conjunction with pressure patterns range from

simple human or optical comparison or direct correlation methods [9], to more sophisticated and at times also more empirical approaches involving averaging, normalization of certain parameters, and data encoding (compression) to decrease memory constraints [10].

While pressure patterns obviously provide a solution to the problem of signature authentication, one should ask whether the x and y components of a signature trajectory might not be used also, either independently or in conjunction with the z-axis pressure patterns? Here comparison of the y ordinates and x abscissas of two signatures will not help much, since this is precisely the kind of duplication forgers excel at. Clearly, a better idea would be to compare the vertical and horizontal velocities, or even more so, the vertical and horizontal accelerations of the signature process. A priori deliberate and careful forgery is certain to fail an acceleration comparison test with an original. However, are the accelerations of each of one's handwritten signatures characteristic to the point where they are the same time after time? The definite answer to this question is found in the work of some IBM researchers. Herbst et al [11], using anterior research [12],[13], [14], established that the act of signature handwriting is the consequence of muscular interactions of ballistic rather than feedback nature. Motions controlled by sensory feedback are usually slow and precise. Ballistic motions are generally rapid, and their accuracy increases with speed. Walking, playing a musical instrument, swinging a tennis racquet or a golf club, writing a signature all are ballistic motions. The muscular forces involved in a signature are therefore predictable and predetermined, and as a consequence, the accelerations of the movement will exhibit the same characteristics. The exploitation of this property of handwritten signatures for the purpose of comparison and recognition of signatures requires suitable input devices. A two dimensional graphic data entry tablet for transforming a positional information into a digital input to a computer system is the subject of a patent assigned to IBM [15]. Another position measuring system of the writing tablet-stylus type is described in patent [16] which is also assigned to IBM. Both of these tablets require analogue or digital 2nd order differentiation of the x-y signals in order to obtain the accelerations. An apparatus consisting of a horizontal platen, connected by a pair of inner beams to an intermediate structure itself connected by a pair of outer beams to a supporting frame, allows direct measurement of the accelerations through strain gage measurement of the forces in the connecting

beams, and is the subject of patent [17] assigned to Stanford Research Institute. More recently developed input devices include x-y pressure transducers [18] or x, y accelerometers [11] mounted within pen -like housings. In both cases, z axis information is acquired from pressure sensors located under the writing surface platen. The recognition algorithms proposed in conjunction with acceleration patterns range from the conceptually simple correlations proposed in patent [19] assigned to Sylvania Electric Products, Inc., to a more sophisticated method of segmented correlation proposed by IBM [11], to a similar method including data compression also proposed by IBM [20], to a method of segmented "rubbery" correlation introduced by Stanford Research Institute [21]. The latter Institute also is the assignee of patents [22] and [23], which present two approaches for computing template vectors from the x, y and z signals, as well as algorithms to compare template vectors of reference and sample signatures.

Whereas all input devices and recognition algorithms mentioned up to this point use either pressure patterns or x, y, and sometimes also z acceleration patterns, the input device and recognition algorithm introduced by RCA and presented in [24] use handwriting speed as a discriminant. This is a natural consequence of the rather unusual but very ingenious input device, where a stylus deposits charge on an insulating writing surface, the total charge laid down being proportional to the total linear extent of the handwritten piece of information, and the magnitude of the charging current being proportional to the instantaneous speed at which the stylus is being moved.

A Signature Recognition Facility

A simple signature recognition system consists of a graphic tablet to translate the x and y deflections of the handwriting into analogue electric signals, a two channel multiplexed analogued digital converter, and a microcomputer to control the A-D converter and to store the samples of the vertical and horizontal deflections. From the surveyed literature and patents, as well as from our own experience it appears that a sampling frequency of 100 Hz is reasonable; in fact, it seems that handwriting signals exhibit major spectral components in the range of 2 to 25 Hertz primarily. Since most signatures are written in less than ten seconds, a storage capability of 2000 pts (10 seconds of x and y deflection both sampled at 100 Hz = 1000 + 1000 points) is amply sufficient. The low sampling frequency of 100 Hz (corresponding to successive pairs of x and y samples spaced at 10 ms. intervals) leaves plenty of time available for any late model microprocessor to perform other tasks such as preprocessing, recursive filter-

ing, validity testing, etc., in between samples. The system constructed at Lafayette College uses a slightly modified Intel SDK-80, a DATEL 8-channel differential A-D converter (obviously an overkill since neither the differential capability nor more than two channels are needed), and a simplistic graphic tablet. This tablet consists of two super-posed conductive sheets (graphite coated Mylar or paper) spaced a few thousands of an inch apart. Electric fields are alterna-tively established on these sheets by supplying them with out of phase pulsed bias voltages. Whenever a writing instrument forces the two sheets into contact, the sheet not presently pulsed acts as the wiper of a potentiometer and picks up a fraction of the supply voltage to the other sheet propor-tional to the position of the stylus. The output of the tablet thus consists of two pulse amplitude modulated x and y signals. These are subsequently low-pass filtered (integrated) in order to obtain two analogue signals suitable as inputs to the A-D convert-er. Reconstruction of signatures from the stored samples of the x and y signals demon-strates the satisfactory working of the acquisition system (Fig. 1). To compute the accelerations, various experiments with digital 2nd order differentiators led to the use of a finite impulse response (FIR) filter of order 25, with coefficients smoothed by a Blackman window. Comparisons of the x and y accelerations of stored reference signatures with submitted original or counterfeit signa-tures were attempted under various criteria. Current recognition algorithms rely on conven-tional dicrete correlation, using a threshold dependent on the original signer's ability to consistently duplicate his own signature. Compression of the data was also attempted, with the conclusion that great care is needed to avoid losing vital information. Presently, success rates of the signature recognizer vary in the range of 80 to 95%, a valid recognition experiment being defined as the submission of a true or counterfeit signature and the correct acceptance or rejection of said signature. The exact success statistics are difficult to assess accurately at this point, since the data base is felt to be too small.

While the original configuration called for the modified Intel SDK-80 to simply trans-mit the x and y samples to the HP-3000 who then performs all the computations, the more recent implementation provides for the micro-computer to perform some minor computations. Ulterior developments will call for the microcomputer to assume a larger part of the data processing load, thus freeing the HP-3000 for more demanding tasks.

Conclusions

A quick glance at the state of the art in automatic handwritten signature recognition reveals a wealth of patents and a few publica-tions on the topic. Two main approaches are evident: (i) recognition by pressure pattern comparison, and (ii) recognition by accelera-tion pattern comparison. Either can be implemented fairly simply using today's micro-computer technology. Simple experiments performed on a system consisting of a simplis-tic graphic tablet, a modified Intel SDK-80, a DATEL A-D converter, and an interconnected HP-3000 show encouraging recognition rates and promise better results in the future.

References

[1] Downey, June E., Graphology and the Psychology of Handwriting, Warwick and York, Inc., Baltimore, 1919

[2] Saudek, Robert, Experiments with Handwrit-ing, William Morrow and Cy, New York 1929

[3] Boldridge, Austin G.,Jr., U.S. patent #4,035,768 July 12, 1977

[4] Radcliffe, Arthur J. Jr., U.S. patent #3,956,734 May 11, 1976

[5] _____, U.S. patent #3,991,402 Nov. 9, 1976

[6] _____, U.S. patent #4,008,457 Feb. 15, 1977

[7] Roggenstein, Edwin O., et al., U.S. patent #3,563,097, Feb. 16, 1971

[8] Johnson, Robert R., et al, U.S. patent #3,528,295, Sept. 15, 1970

[9] Clark, Robert K., U.S. patent #3,621,720, Nov. 23, 1971

[10] Sternberg, Jacob, et al, U.S. patent #3,959,769 May 25, 1975

[11] Herbst, N.M., and C.N. Liu, "Automatic signature verification based on accelerom-etry" IBM J.Res.Develop., May 1977

[12] van der Gon, J., and J.Thuring, "The guiding of human writing movements", Kybernetika, 1,145, 1965

[13] Eden, M., and M. Halle, "The characteriza-tion of cursive writing", Information Theory, C. Cherry, Ed., Butterworths Publishing Co., Washington, D.C. 1961.

[14] Vredenbregt, J., and W.G. Koster, "Analysis and Synthesis of Handwriting," Philips Tech. Rev. 32, 73, 1971

[15] Dym, Herbert, U.S. patent #3,668,313,
June 6, 1972

[16] Mazza, Robert V., U.S. patent #3,582,962,
June 1, 1971

[17] Kamphoefner,Fred J., et al, U.S. patent
#3,988,934 Nov. 2, 1976

[18] Eernisse, Errol P., et al, "Piezoelectric
sensor pen for dynamic signature
verification," Proc. International
Electron Devices Meeting, Dec.5-7,1977,
Washington, D.C.

[19] Dyche, James W., U.S. patent #3,699,517,
Oct. 17, 1972

[20] Herbst, Noel M., et al., U.S. patent
#3,983,535, Sept. 28, 1976

[21] Crane, Hewitt D., et al, U.S. patent
#4,040,012 Aug. 2, 1977

[22] Crane, Hewitt D., et al, U.S. patent
#4,040,011 Aug. 2, 1977

[23] Crane, Hewitt D., et al, U.S. patent
#4,040,010 Aug 2, 1977

[24] Engelbrecht, Rudolf S., U.S, patent
#3,962,679 June 8, 1976

Fig. 1   COMPUTERIZED SIGNATURE

INPUT HARDWARE DESIGN FOR CONSUMER ATTITUDE RESEARCH WITH A MICROCOMPUTER

Dr. H. P. Munro
Director, Laboratory for Research and Instruction
in Rhetoric and Communication
Kent State University
Kent, OH. 44242

Possible business uses for a microcomputer include consumer research for product developers or advertising agencies. Of the areas of concern for the owner of a microcomputer system who wishes to adapt it for such a purpose, two are the foci of this paper: 1) some psychological and statistical cautions which should be observed in order to generate useful data, and 2) how to design input hardware to implement these cautions. Reference is made to "home-grown" hardware in a university-based lab which could be used to do pre-market studies of goods, services and advertising messages.

Introduction

One possible business use for a microcomputer is consumer research for product developers or advertising agencies. Quite a few such companies are located in places other than New York, Chicago or L.A., and local ad campaigns for products and services might more quickly, intensively and efficiently utilitze a microcomputer system with appropriate input hardware than by the traditional interview methods or written questionnaire and attitude surveys.

The system which has provided the experience for my observations in this report is a university-based social psych communication research lab used for measuring the impact of various messages, appeals and sources on audiences in the investigation of theories of rhetoric and communication, but the same configuration of equipment could just as easily be employed -- for a fee -- to do pre-market studies of prospective consumer goods and advertising messages.

My remarks will focus on two areas of concern for the owner of a microcomputer system who wishes to adapt it for consumer research: 1) some psychological and statistical cautions which should be observed in order to generate useful data, and 2) some "how-to" suggestions on appropriate input hardware to implement these cautions.

Psychological cautions to observe include: (a) choice of target concepts to be judged; (b) choice of judgment scales on which to record audience responses; and (c) number and identification of intervals (degrees) between the bi-polar extremes of each scale. Statistical cautions are many, but the main ones germane to the design of hardware have to do with the constraints usually imposed on sampling and scaling; (a) number of judges to be sampled; (b) preservation of the interval assumptions; and (c) exclusion of multiple or ambiguous responses on a given scale.

Psychological Cautions

Our laboratory at Kent State [1] uses CCTV videotape and a full range of A-V apparatus (most remotely controlled) to study the impact of speakers and other message sources on audiences. The lab is located in three rooms and is often used, in both research and instruction, to obtain response data from listener/viewers correlated simultaneously with a split-screen videotape of the message source and selected portions of the audience (to record visual feedback). The response collecting hardware has undergone several generations of R&D change, with construction financed by limited funds and subsidized by graduate assistant labor; none of the gear in this report was bought "off-the-shelf" -- economics and unavailability of hardware to meet our needs have caused it to be "home-grown." This takes awhile, but in the end it is much cheaper and better adapted to specific needs.

Certain constraints from social psychology and learning theory guided the design of our hardware. Some of our faculty do research on attitudes and attitude change. Thus, evaluations (good-bad, like-dislike, agree-disagree) and probability (likely-unlikely) were known to be frequent parameters of judgment. Others of us cut our teeth on the Semantic Differential of Charles Osgood and associates at the University of Illinois [2], and so other dimensions of judgment are needed: potency (strong-weak) and activity (active-passive). Most of what follows in this section will be based on considerations imposed by these schools of psychology, as they

bear on the study of human communication.

Choice of Target Concepts. One of the assumptions in Osgood's work is that a concept to be described through the bi-polar scales (good-bad, certain-impossible) and quantification (+3 to -3) of semantic differentiation will occupy a single point in an n-dimensional semantic "space." There has been considerable criticism, however, that this seems not to be true, as (careless) users of the Semantic Differential have reported that concepts like "myself" and "car," for example, shift around over time (are unreliable) even with the same subjects. The problem here is that such concepts are over-broad and may be differently qualified by a subject at various times. That is, "car" could mean, when one is asked to respond to a set of scales resulting in a semantic profile, 1) the auto I have (junker), 2) the car I would like to have (Mercedes 450 SLC), 3) or the sub-compact a favorite student was killed in a few weeks ago. Obviously such "cars" will have different "locations" in the semantic space, i.e., will have different meanings, because they are in fact different concepts. So, the caution to be observed in choosing target concepts is to avoid ambiguity: qualify and condition each concept to be tested as fully as possibly, even if that requires, not one word as is so commonly found, but a phrase of several words, or even a full sentence. For example, don't ask subjects to respond to the "Your Town Last National Bank," for how do you know they might not be thinking of the appearance of the building, and that would certainly not fully tap the bank's desire to probe its total "image" with present or prospective depositors. Instead, be more specific: "Quality of Drive-Up Window Service at Your Town Last National Bank" would be better.

Choice of Judgment Scales. The quality of audience responses, in either ivory tower or marketplace research, depends on their validity and reliability. Simply put, do the concept judgments measure what they're intended to measure, and are judgment stimuli reasonably stable (across subjects and time)? These are usually thought of as statistical constraints, and indeed they are, but validity and reliability are also products of the psychological sophistication of the researcher and

his instruments of measurement. They depend not only on the proper choice and precise wording of concepts (just discussed), but also on the choice of scales by which to measure those concepts.

Detailed advice on this, as on other matters cannot be developed within the compass of this paper, but should come from professionals in the discipline of psychology and/or a careful reading of their work. For example, even though one could select 9 or 12 scales from lists in Measurement of Meaning (Osgood et. al.), too many users of the Semantic Differential have ignored the advised pre-testing of a given set of scales to make sure of their fitness for a given set of concepts and subjects. Thus we have, in the technical literature, such gaffes as "hot-cold" chosen as a hopefully "pure" scale (no loadings on evaluation [Factor I] or potency [Factor II]) to represent Factor III (activity) in the Semantic Differential and applied to such concepts as "pizza" and "beer," where the emotive connotations (which is what is measured by the Semantic Differential) become confounded because of referential or denotational associations (hot pizza is good and so is cold beer . . . except to an Englishman maybe!).

Commercial users of the Semantic Differential and similar scaling devices in the marketplace seem subject to even more ridiculous lapses of common sense. Even without consulting the technical literature on the proper choice of bi-polar adjectives, one should know better than to oppose "good" with "dull." And yet that is precisely what has been done by a rating firm in Hollywood (a major tester of live audience reactions). According to a recent report by Associated Press columnist Jay Sharbutt [3], this firm asks audiences to dial a choice from "very dull," "dull," "normal," "good," or "very good" in judging TV shows and commercials. What happens when the action in a television drama is both exciting and morally reprehensible? Perhaps the TV industry is finally ready to assert that murder, rape and other violent crimes are really "good" because they are not "dull," but my guess is that whoever chose the scales wasn't thinking too clearly. Lots of things are "good," in many senses of that umbrella word, including anything that is useful to some further goal, but good is not equal to pleasant, so it would seem more sensible to oppose "dull" with "lively" or "exciting,"

or "interesting." It would appear that this would be in the interests of a testing service as well, because it can only muddy the results to mix in moral judgments with ratings of entertainment value.

So the caution here is to choose rating scales which are true opposites, and which are not "noisy" or ambiguous with several differing interpretations possible.

Degrees Between Ends of Each Scale.
The identification and number of intervals between the bi-polar extremes of the scales chosen will depend somewhat on the scale names and the probable nature of the audience. If one is measuring agreement, then the usual qualifiers on a seven-point scale (like "agree-disagree") are "very strongly," "strongly" and "slightly" on each side of the neutral or ambivalent ("either/neither") center point. On the other hand, Semantic Differential qualifiers are usually "extremely," "quite" and "slightly." A way to resolve this problem, and provide for a statistical concern yet to be mentioned, will be suggested in the hardware section of this paper.

The question of how many divisions to have in one's scale has occupied a number of psychologists. Usually the answer is: as many as can be discriminated reliably (on test/retest trials) by a given audience. That is, the more divisions, the more useful the data because discrimination among concepts can be finer, until the point is reached where subjects are "overloaded" and can't really tell, say, between "slightly" and "quite." In this case, it would be wise to shrink the seven point scale to five. Osgood, in his lectures in the fifties, used to observe that studies suggest a nine-point scale for college graduates, seven for sophomores and freshmen, five for high school students, three for less-then-high school education . . ., and perhaps two for members of the American Legion ("If'n yer not fer it, y' must be agin it")!

Statistical Cautions
This is even less appropriate a place to presume to review the many caveats that have to do with designing studies and processing data. Best to get the advice of a competent statistician, and if you find one whom you can understand, put him on a commission basis for any revenue you might generate.

All that I seek to do in this section is mention a few concerns that will influence the design of input hardware.

Number of Judges to be Sampled.
This is an important concern because a product or message testing agency will need to know how many response stations to build. Generally, the minimum number of subjects (for parametric statistical measures, fewer are needed for "non-parametric" tests) is 25 or so. At Kent State, my Lab was designed for 24 response stations, partly because the room won't hold many more people and because the mainframe of a previous generation of response stations lent itself to multiples of 12. (Those earlier machines used interlocked multiple pushbutton switches and stepping relay counters, but the present system, having 75154 IC's in the I/O interface to the computer, suggests multiples of 16, and 32 would be a better number of stations).

But neither 24, 25 or 32 are actually needed. A rating service could get by on quite a few less because the statistical caution to be observed has to do with the number of subjects, not stations. If one is willing to give a number of presentations to a smaller number of subjects each time, then six, or ten or some similar number of stations would be adequate. The trade-off is between time and money: more stations are more expensive but quicker, while saving construction time and money will later eat up testing time. In the final analysis, how large a room (for screening messages), how much capital one has to invest and how much business one might expect should all help determine an optimum number of stations to build.

Preservation of Interval Assumptions.
What's involved here is the difference between nominal (by category), ordinal (rank order) and interval judgments. Scales are assumed (hoped?) to be equal-interval means of measurement; that is, the distance between 3.0 and 2.0 should be the same as between 1.0 and 2.0. With pencil and paper it is easy to suggest this to subjects, but electro-mechanical devices need to be more carefully selected. Dials are troublesome, especially if connected to potentiometers. "Pots" are analog devices; what is needed is a digital selector. A dial connected to a wafer switch is better, but even with this the labelling of the dial could cause response problems. Our solution at Kent was a bank of seven MicroSwitch units, as will be demonstrated in the

hardware discussion.

Exclusion of Multiple or Ambiguous Responses. Each given scale should receive one, and only one response. In pencil-and-paper tests, subjects are instructed to put their "X" marks on the lines, not between them, and put no more than one per scale. Here is one place where a machine substitute can be much more efficient than pencil-and-paper scoring, at the cost of a little care in design. Human scorers must spot and then throw out violations of these conventions, whereas a machine can be designed to prevent such actions from occurring. More on this shortly.

Hardware Design Considerations
It is not the purpose of this section to provide full details for the construction of a given design of input response station; even if space permitted such specifics, it is doubtful that the same parts and materials would be obtainable, perhaps not even desirable to others. Thus, no schematic wiring diagrams will be provided; instead, I shall suggest how certain choices were adapted and useful in our situation. To facilitate the discussion, reference will be made to the several photographs at the end of the text.

Target Concepts. Once selected, target concepts are probably best photographed in 35mm or 2¼ square format transparencies and projected onto a screen adjacent to the message or object being judged. At Kent we have a 4 by 6 foot rear projection screen in one wall, onto which several projectors may be directed. Thus, the concept language may be put on the same screen with still visuals, 16mm movies, or television monitors (or even live performances) can be arranged nearby. If only a few concepts are to be judged, then overhead transparency production, perhaps by the thermal process, is quicker and cheap. Also, it is not necessary to use rear projection (we do simply to keep the noisy equipment out of sight and earshot); all that is required is to be able to display the message (e.g., a film or videotape) along with enough "concept" language to focus the judgments of an audience (e.g., "If I were to use the product in the message, I would probably find it: _____"). The blank at the end of the concept statement

is for the various scale judgments desired (e.g., "good-bad," "worthwhile-worthless," "strong-weak," "active-passive," etc.) As can be seen by the comparative length of the language involved, projection onto one screen common to all subjects is a practical necessity -- to fit so much into each response station would require a fortune in 5x7 matrix alphanumeric LED R/O's or a CRT in each station.

Judgment Scales. Judgment scales, on the other hand, while they might also be projected onto a common screen (and unusual ones would have to be) could, we found, be put into each response station: they are short (one word at each end of the scale) and a handful tend to recur in much of our research. As Figure 1 reveals, the arrangement of the "business end" of each response station is an electronic and mechanical translation of the geometry of a single scale as it usually appears on paper-and-pencil tests, e.g.;

good __:__:__:__:__:__:__ bad

The squarish windows on either end, slightly separated from the bank of seven rectangular push-buttons are actually small projection screens for I.E.E. numeric readouts, originally containing a piece of film with "0-9," "M" and a red filter. What we did at Kent was to substitute a new film negative into the optics of each readout which now projects the scale designations we choose. In our case, we finally chose the following bi-polar adjectives as being of most utility for us:

Left Hand Readouts	Right Hand Readouts
Good	Bad
Strong	Weak
Active	Passive
Believable	Unbelievable
Certain	Uncertain
Possible	Impossible
Pleasant	Unpleasant
Agree	Disagree
Aware	Unaware
Clear	Unclear
Right	Wrong
Lively	Lifeless

Although there are only twelve apertures in each film/optic sandwich, we actually have a total of 13 scales, as a bit of diode circuitry allows us to oppose "certain" with "impossible" to get a full $p.=0$ to $p.=1$ range on one scale.

Identifying the Intervals. Identifying the intervals between scale extremes gave us fits for some time, until we hit upon a novel, but simple, non-

verbal way of communicating any of the various qualifiers discussed above. We could have inserted Xerox or Thermofax etched transparency film into each of the translucent caps for the seven MicroSwitch [4] units, but this would require changing caps whenever a different set of qualifiers was used. We can still go this way for use of the seven selectors as nominal categories but since most of our work is with interval scaling we chose, instead, to leave off all language, and communicate degrees ("very strongly," "strongly," "slightly") by the intensity of the cap, as illuminated by the #327 bulbs located behind it. This was easily accomplished by a resistor network, resulting in the +3 to -3 appearance in Figure 1, and the 7 to 1 look in Figure 2. One bulb and resistor network is used for each of these two aspects, with the third bulb (through a red filter) on all seven switches serving as a "stop" warning. The fourth bulb signals when only the extreme and middle positions are active and is controlled by a nearby spring-loaded lever switch (for 3-degree judgments, a more practical number for continuous judging during a longer message).

Preservation of Interval Assumptions. Preservation of interval assumptions is, we hope, accomplished by the symmetry of 7 identical push-buttons, ranging in a row from one scalar extreme to the other. This configuration also seems to remove possible confusion when a -2 PB cap may appear to have almost the same intensity as the -1, as will sometimes happen with bulb aging. The subject sees the pattern of the whole scale and seems to understand the idea, even without elaborate and lengthy verbal instructions which is one of the pains of paper-and-pencil testing.

Exclusion of Multiple or Ambiguous Responses. Exclusion of multiple or ambiguous responses is accomplished by the design of both the input sampling I/O and the interconnection of the bank of seven switches internal to each unit. Multiple responses are excluded (except when they are desired, as in sequentially judging continuous phenomena) by a controlled sequence of activities: 1) present message stimuli (with a concept and scale) to audience (e.g., activate the advance on a slide projector); 2) wait a suitable period,

perhaps 15-20 seconds, for subjects to arrive at their judgments (during this period, they may change decisions already entered into a unit simply by selecting, and pressing, another switch)--no data is recorded until the end of this period, when 3) a selector circuit samples each of the 24 response stations in turn, sweeping all units once within a few milliseconds, then 4) the "hold-in" voltage to the coils on the Micro-Switch units is interrupted momentarily, at which time all stations are re-set, and the next message, concept and scales are selected and presented for judgment. Ambiguous responses are excluded by means of the electrical bail circuit through which the MicroSwitch units are interconnected. This circuit simulates inter-lock latching and "lockout" features of our previous banks of mechanical switches, whereby pressing one switch releases any other, and it is impossible for two or more switches to be activated at the same time.

Conclusion
There are a number of advantageous features of the system presently in use and described here which have made the time and effort of design and construction worthwhile to us. Despite what may seem to be a rather kluge-like contraption (the innards are pictured in Figure 3), it works! The case was fabricated of aluminum and designed to clamp onto a standard classroom desk -- when so attached, it is so sturdy that the desk can be lifted by the response box. Data lines, power and control lines number 30, contained in a 36 conductor cable from each station to an interconnect mainframe; the large 200 pin socket seen in the photographs allows testing and access to various circuit points without taking apart the two halves of the case. The advantages of this design, aside from meeting the cautions and considerations previously discussed, make the unit convenient, easy and fun to use. This last quality is of no small importance when subjects are asked to supply a number of responses at one sitting. Paper-and-pencil work is dull and tiring, but units such as these, while not as fully entertaining as Artoo Detoo, are nonetheless somewhat responsive, and thus "alive." For example: the buttons are not only large and easy to hit, they are differentially lighted, and when any one of them is pressed lightly, a special circuit activates the "pull-in" coils of the Micro-

Switch units [4] thus completing the action with a slightly perceptible tactile feedback. The same coils also serve a "hold-in" function, by means of a modified electrical bail circuit, enabling the latching and lock-out features, and allowing all activated switches to be re-set simply by opening the common 28 volt line to all units. Simultaneously with the self-completion action of the pull-in coils, the response unit signals to the subject (and continues to remind him until the data is recorded and all units are re-set) which scale division he chose. This we accomplished, not by the usual and obvious alternative of turning on a lamp under the chosen switch cap, but by turning off all the other six lamps (see Figure 4 for an example).

While our experience with the design described here has been gratifying, it is only one of many possible hardware implementations for input devices with which to conduct microcomputer-based consumer attitude research. Advancing electronic technology challenges us to develop response units which would be equally or more sophisticated in their functions, but simpler internally and with a smaller component count. As such devices emerge, small system computerists will be increasingly enabled to conduct profitable audience and market research.

References

1. More information on this facility is available in "The Kent State University Laboratory for Research and Instruction in Rhetoric and Communication: A Report on the Rationale of its Design, State of Development, Capabilities and Potential for Functional Growth" (1972, 1976). This report is available from H. P. Munro, School of Speech, Kent State University, Kent OH, 44242.
2. Charles E. Osgood, George J. Suci, Percy H. Tannenbaum. The Measurement of Meaning (Urbana: University of Illinois Press, 1957).
3. Jay Sharbutt. "Guinea Pig: TV 'Tester' Rates Evening 'Very Dull.'" Akron Beacon Journal, January 2, 1978
4. The switch units which we managed to obtain (as electronic "surplus" material) are from MicroSwitch, Series 2 Rectangular Display Modular Lighted Pushbutton Switch/

Indicators. The Operator/Indicator Housing type is #2C69, and the switch module was #2D172, modified to momentary rather then alternate action and using only DPDT contacts. Mounting barriers (2B-) were not used so as to provide the better appearance of a continuous scale with closely segmented intervals; there is enough clearance between display screens (caps) to permit easy operation without fouling, if the mounting cutout (in our case, for a bank of seven switches) is carefully measured.
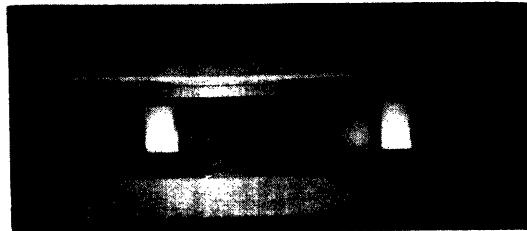
Figure 1
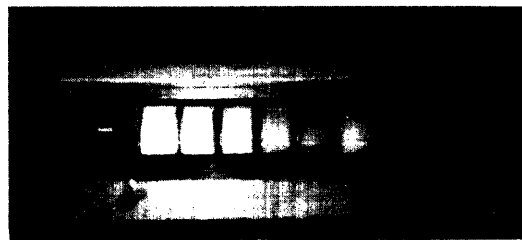
Figure 2
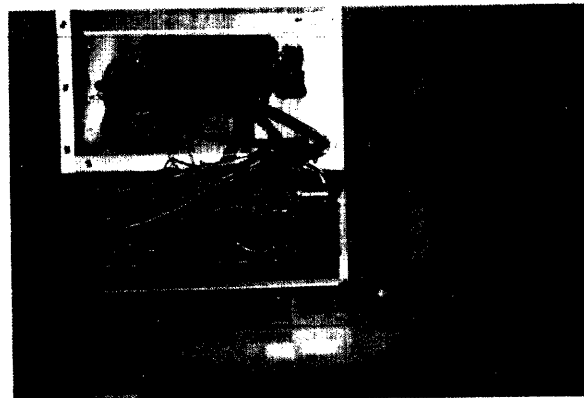
Figure 3
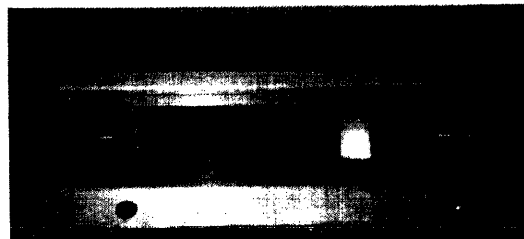
Figure 4

IMPROVING NAME RECOGNITION AND COORDINATION IN VIDEO CONFERENCING

David Stodolsky, PhD, Center for Educational Research
Stanford University, Stanford, CA 94305   415 494-2106

Abstract

Current techniques for coordination of group video conferencing are derived from broadcast applications. Conferencing typically occurs between two locations, and often a number of support personnel are required for equipment set-up and operation. A computer-based mediation system can be used to resolve problems of technical and social coordination while reducing costs and improving name recognition.

Conferees indicate their wish to speak by pressing a switch, and the computer at their location transmits this request to the location chairing the conference. The request is queued by a mediation program. When the previous speaker terminates, the new speaker's camera is activated, and the speaker's name is automatically displayed below the image. Conflicting requests are typically resolved according to preprogrammed rules; however, the chairperson may override the automatic resolution mechanism at will. Computers at each location are responsible for set-up and self-testing as well as camera activation and name display. The resultant system permits flexible interaction among participants while balancing participation and improving the quality of deliberative decisions.

Video conferencing opens the prospect for a new era in representative democracy. The full potential of this technology, however, requires careful analysis of its current limitations. These limitations are not technological, but stem from the non-interactive mode of operation typical of the video medium.

While the current video techniques are relatively easily extended to interaction between two locations, multiple location conferences have proved problematic in past experiments. If the full capabilities of the media are to be realized, coordination techniques for multiple location conferences must be improved.

Automation of many functions provides an ease of use and flexibility which could facilitate acceptance of the technology. For instance, it would make practical the placement of interactive stations within or near representatives' offices. This distribution of stations and the ability to activate the system on short notice, since few support persons would be necessary, could be crucial in increasing utilization of video conferencing technology.

Increased availability of communication satellite channels, perceived need for a closer relationship between representatives and their constituents, and possibility of substantial economic savings have resulted in experimental use of video conferencing by Congress (Carter, 1977). The technique to be described here could be used to improve name recognition and coordination in such conferences. The method could reduce the costs of staff support by automating certain functions normally performed by camera crews and their backup persons. Previous research in teleconferencing has indicated that automatic switching may be preferred in teleconferencing (Bretz & Dougharty, 1974; Stodolsky, 1976). The operation of a Resolving Bridge for teleconferencing is probably best explained from the user's viewpoint. Prior to teleconferencing, each user's name is typed onto the system control keyboard. (With an appropriate card reader connected to the system, members of the House of Representatives could merely insert the card used with the House voting system to enter their names.) When the conference is called to order, the chairperson triggers the system initialization procedure. Initialization is equivalent to a roll call. A person's name appears on the video display, and that person presses a switch at his position. The person could also state his name to indicate proper pronunciation and confirm correct name assignment. (Stodolsky (1976) employed an error-checking procedure to ensure a one-to-one correspondence between names and conferencing positions.)

During conferencing, the same switch is used to request the floor. When a speaker is selected, his name appears on the bottom of the screen, and his microphone is simultaneously activated. Camera pointing could also be a function of speaker selection if the necessary procedures were followed during initialization.

With the Resolving Bridge, speaker selection is normally automatic but manual override is available to the chair at any time (Cohen, 1976; Stodolsky, 1976). The chair may also choose the rule for speaker selection. The selection rule may be based upon principles of priority, precedence, or equal time (Stodolsky, in press).

In a previous experiment (Stodolsky, 1976) the rule came into action only when two or more persons were simultaneously requesting the floor. For example, the Equal Time Resolution rule would always select the person who had spoken the least time thus far in the conference. Rules of this sort are most helpful when there are many conference participants.

The many-participant conference can be of two types, one in which participants are each at different locations and one in which they are grouped at remote locations. When each participant is at a different location, the ability to switch rapidly from one image source to another is essential. When participants are grouped at a remote location, the video image can display a panel, thereby sacrificing image resolution for technical simplicity. With appropriate initialization, the Resolving Bridge mechanism can improve effective resolution in this case. The full size image and name display continuously available during conferencing improves name identification and recognition. The automatically triggered switching improves coordination in conferencing by eliminating delays and permitting rapid interchanges among participants.

The Resolving Bridge may also have an important impact upon the emotional tone and the performance of a deliberative body. The above mentioned Equal Time Resolution rule, for instance, tends to balance participation and assist more reticent individuals in gaining the floor. This influences both the perceived fairness of the proceedings and the likelihood that relevant information will be contributed by such persons. Even a simple rule restricting freedom to interrupt has been found to improve the quality of deliberative decisions (Morley & Stephenson, 1969). Short (1976, p. 93) confirms that greater formality in deliberative procedure tends to improve performance. The Resolving Bridge for teleconferencing is a powerful method for structuring deliberations by increasing formality without sacrificing flexibility.

Acknowledgments

References

Bretz, R., & Dougharty, L. A. Two-way TV conferencing for government: The MRC-TV system. (R-1489-MRC). Santa Monica, Ca: Rand, April 1974.

Carter, L. J. Videoconferences via satellite: Opening congress to the people? Science, July 1977, 197(4298), 31-33.

Cohen, D., and ISI Research Staff. Network Secure Communication. In ISI Research Staff, A Research program in computer technology, annual technical report, July 1975-June 1976. Marina del Rey, Ca.: University of Southern California, Information Sciences Institute, July 1976, (ISI/SR-76-6).

Morley, I. E., & Stephenson, G. M. Interpersonal and inter-party exchange: A laboratory simulation of an industrial negotiation at the plant level. British Journal of Psychology, 1969, 60(4), 543-545.

Short, J., Williams, E., & Christie, B. The sociology of telecommunications. London: Wiley, 1976.

Stodolsky, D. Machine-mediated group problem-solving: Therapy, learning, performance (Doctoral dissertation, University of California, 1976). Dissertation Abstracts International, 1976, 37, 1949-B. (University Microfilms No. 76-19,633).

Stodolsky, D. Group conferencing with automatic mediation. In J. Belzer, A. G. Holtzman, and A. Kent (Eds), The encyclopedia of computer science and technology. New York: Marcel Dekker, in press.

THE BEDSIDE MICROCOMPUTER IN THE INTENSIVE CARE NURSERY

Robert C. A. Goff, M.D.   Fellow in Neonatology
Children's Hospital Medical Center of Northern California
51st and Grove, Oakland, California   (415) 654-5600 ext
431

Abstract:

Software has been developed to enable Pediatricians and Neonatologists to maintain bedside microcomputers in the Intensive Care Nursery, providing instant processing of, and access to the voluminous laboratory data and evnt summaries generated by each infant. The data is stored in a problem-oriented format, and may be accessed with an inquiry to any particular problem. The program is written in North Star extended disc BASIC (version 6, release 3, -- which utilizes random disc file access) and is implemented on a SOL/20 Terminal Computer with 48K RAM, and triple North Star Microdisk drives.

Introduction

The use of computers in the Intensive Care Nursery is not a new idea, but I am unaware of previous attempts to use a microcomputer in such an application. This paper will discuss the structure of the software, and the reasons for both the programming language used, and the selection of the hardware configuration.

As a general background, one must realize, first of all, that infants in an Intensive Care Nursery (ICN) are usually highly unstable patients with multiple, complex medical problems, when compared to older patients in other hospital settings. Secondly, and as a result of this first consideration, infants in the ICN generate an overwhelming quantity of both data and narrative description of clinical events, conditions, and procedures. As an example, the daily progress note written in a typical patient's chart (in most areas of the hospital) will require, perhaps 8 or 10 lines on one page of his chart, and will include all new laboratory results and procedures, as well as the patient's clinical condition for that day. In the ICN, however, it is not at all uncommon to find a progress note for one day requiring as many as 2 to 3 full pages of discussion and laboratory

values. In addition to this voluminous daily documentation, there is at the bedside of each infant a flow chart of all the daily laboratory results, and the daily computations of fluid intake, fluid intake per kilogram per day, calories per day, calories per kilogram per day, urine output per kilogram per hour, etc.

While today, with no less than a Herculean effort, we are still capable of managing and reacting appropriately to the reams of information generated by our ICN patients (up to about 40, at full census), we are realizing that the current trend in Neonatology is toward more laboratory tests per infant and more documentation of the increasing number of proceures required by each infant. If this trend continues, then it will be virtually impossible to keep apace of this information deluge. Future trends aside, it is currently a major task, each time one of our infants is discharged from the hospital, to review his records (often consisting of 3 to 8 volumes of hospital chart), understand his 2 to 3 month hospitalization, and then dictate a usefully concise and accurate summary. Our present practice is to dedicate 1 to 3 hours in preparing each summary, with our nursery requiring, on the average, 2.5 summaries per day.

The solution to this porblem is obvious--to utilize some form of computer processing and synthesis of both laboratory data and event description to maintain instant access to any past information, to accept daily input of data, and, at the time of discharge, to abstract from the patient's file those pertinent items appropriate for inclusion in a discharge summary.

Software

The program, written in North Star BASIC, is fairly simple in structure, but, because of the multiplicity of types of data manipulations required, and the extensive text capability

required of a narrative summary, the program length is projected to run to about 60K bytes, exclusive of the space needed for variable manipulation. As a result of its length, the program is structured in the form of disc files, directed by an executive program which chains into RAM whichever sub-program is called. The program resides in one disc drive, and stores all patient data on the other disc (a triple drive will enable the use of one microcomputer for every two infants). The organization of the sub-programs is a modified version of the Problem Oriented Format, now popular among physicians, and extensively used in Intensive Care Nurseries across the country. Rather than being oriented strictly to patient problems, the sub-programs are representative of body systems. This approach allows the formatting of virtually every type of clinical data or problem, and will permit easy access, at a later date, to any particular information required by patient follow-up or retrospective data analysis.

It should be noted that, because of the relatively stereotyped set of clinical problems common to premature and sick infants, an estimated 90 to 95% of all data and event summary information can be specifically encoded for later search and retrieval. The remaining 5 to 10% of the information would be accessable only by manually searching a category such as "OTHER", included under each body system. This situation is not the case with general medicine or, for that matter, general pediatrics, in which the multitude of commonly encountered clinical entities would require a much more complex structure of software in order to accomplish a comparable textual product with comparable data access capability.

Executive Routine: This is a short program which displays on the CRT a menu of the major sub-routines of the system. If the hardware consists of three disc drives, for use on two different patients, then the patient is selected within the executive routine. At this point, the physician chooses the particular body system of interest, and the executive chains in the sub-routine which has been called. With the sub-routine loaded, the physician is presented with a menu of routines included, such as:

1. Enter Data
2. Enter Events
3. Review Data and Events
4. Plot Data
5. Print Textual Summary
6. EXIT

On selecting, for example, #2, the CRT displays a menu of events related to the chosen body system. When an event has been selected, the computer will then request the time and date of the event, and then compute the infant's age at the time of the event. This is now displayed for confirmation, and if approved, will be recorded in the appropriate file on the patient's disc. An opportunity is given to record additional events, then control is returned to the initial sub-system menu. On exiting the sub-system, the executive routine is chained into RAM, and it is then used to access further sub-routines.

Sub-routines: The major sub-routines serve to format the data and events into either random or serial disc files, whichever is most easily manipulated for the given type of information. The plotting functions are capable of producing graphs of data, plotted simultaneously with certain event markers, as well as standardized curves for reference. For example, the plot of the growth chart will, on a single page, plot three graphs: weight, length, and head circumference, each with appropriate standard percentile curves, and each in the format of the Babson growth chart. Because of the ease of generating these curves, and their usefulness to the physician who follows the infant after discharge from the ICN, they may be included in the final discharge summary, and available to the referring physician immediately. Additionally, attempts are being made to represent complex data, such as arterial blood gases and ventilator settings in easily interpretable graphic form.

Text: Most of the infant's admission history (primarily prenatal and maternal history) is encoded, and at the time of review decoded, by the History sub-routine, so that most of this textual material is confined to the program disc, and does not require space on the patient's disc. However, uncommon items of history can be typed in as text, and are stored as strings on the patient's disc. This is also the case with each of the other sub-routines. The finished discharge

summary will be in the form of a standard textual discharge summary, and may optionally be formatted as a letter.

An additional feature of the output capability of the software is that it can print the forms (presently filled out by hand) which are required by the State of California for each infant who is transported from a referring hospital to an Intensive Care Nursery.

Diagnoses: Each sub-routine posesses considerable diagnostic capability. Any diagnoses which can be made solely on the basis of laboratory data and encoded events or encoded history, will automatically appear in the summary as discharge diagnoses. While the attending physician has the option of deleting any of these diagnoses, or adding other diagnoses to the list, it is anticipated that by far the majority of diagnoses will be accurately made by the program, and will maximize future access for statistical study of patient care information. An additional feature of the diagnostic algorithms is that any suggestive (but not conclusive) diagnoses will be pointed out to the physician as possibilities which may warrant further clinical or laboratory investigation. (Once again, the rather circumscribed nature of neonatology allows this capability to be implemented on so small a system.)

Choice of Programming Language

Only two languages were considered in setting about this project: 8080 Assembly Language, and BASIC. The former would allow for a much more concise program structure, considerably less RAM, and more rapid program execution. BASIC was chosen instead, for several reasons. Most importantly, it would allow the program to be quickly modified to run on just about any hardware system, including time-shared systems and the large systems available at most university hospitals. A second advantage to BASIC is that it would allow other users to easily modify any of the graphic, textual, or diagnostic routines to meet their exact needs or preferences.

North Star extended disc BASIC (version 6, release 3) was chosen in particular, for three reasons. First, it is an extremely powerful and easy to use BASIC. Second, it posesses the CHAIN function and extensive string manipulation capability. Third, this was a natural choice to use with the North Star disc drives, discussed below.

Hardware

The hardware chosen consists of a SOL/20 Terminal Computer with 48K bytes of RAM, a three-drive North Star Microdisk drive combination, a Sanyo CRT monitor, and a Diablo 1610-3 receive-only "daisy wheel" printer. If the system is to be used for only one patient at a time, or by several patients, changing discs for each patient, then a dual disc drive will suffice.

The choice of the SOL/20 was based on two major factors. The first is size. The Sol will fit comfortably at the bedside in the ICN, whereas most other microcomputers are simply too bulky. The enormous backplane capacity of the larger units is not needed. If the 48K RAM is all on one high density memory board, then the SOL will have 3 empty slots for further hardware development. The second major factor in the choice of a SOL is its user-oriented keyboard, and lack of a front panel. The optional numeric keypad is a tremendous advantage when entering large volumes of laboratory data. Perhaps a third consideration is the ease with which the SOL may be set in a "terminal mode" and networked to a laboratory mini-computer, for direct data acquisition. (A network of micro-computers is certainly in the near future for most hospitals.)

In considering the various disc drives available, again two facotrs were of greatest importance. First, once again, was size. The large disc drives simply require more room than is available presently at the bedsides of ICNs. The North Star drives can be tucked away just about anywhere. Second, was flexibility in interchanging one patient's data, at the time of his discharge, for that of another patient. A large disc would be wasted if it contained the information of only one patient, and flexibility would be lost if a large disc were used for more than one patient. The 90K byte capacity of the 5 inch disc seemed to be only a slight overkill, and could easily justify the use of one disc per patient.

The software was originally developed using a DECWRITER as the hard copy output, and all graphics were implemented so that any serial printer would be capable of generating entirely adequate graphs. Chosing a Diablo "daisy wheel" type printer was prompted by not only the desire for more precise graphics, but also the preference of most physicians for reading a solid type font, rather than dot-matrix. The printer, whatever the type, is not intended to be at the bedside in the ICN. It should ultimately be a part of an additional system located in some other area of the nursery or nursery offices, and would be used solely for printing the discharge summaries. This additional system can easily be cost justified by using it the remainder of the time for inventory, scheduling, accounting, and numerous other tasks. Alternatively, the printer may be placed on a mobile cart, and rolled to the bedside unit for use at the time of discharge.

Costs

The system described, including the Diablo printer, should cost approximately $9000.00 with all necessary supplies and sales tax. Substituting a DECWRITER for the Diablo 1610-3 will drop the cost by about $1200.00. These prices include the cost of 100 floppy diskettes--$450.00 (for a census of 40 patients).

Justifying such an expense should be in the light of the cost of typical monitoring electronics used in the ICN. As an example, the PSI infant monitor (which monitors heart rate, respiratory rate, blood pressure, heart rate trend, respiratory trend, and blood pressure trend, along with appropriate alarms) runs in the neighborhood of $10,000.00 per bed. Less expensive monitors are still in the $5,000.00 range. By using one triple disc drive SOL system for every two beds, the cost is about $2,200.00 per bed, plus the cost of one printer spread over the entire nursery. These figures, of course, do not measure the improvement in patient care that would result from instant data access at the bedside, as well as increased physician time attending to matters other than a dictaphone. There is also a significant savings in medical transcription costs, by eliminating the need to transcribe lengthy ICN discharge summaries. Perhaps the greatest cost justification for a large referral center, such a

Children's Hospital in Oakland, is that by generating discharge summaries at the instant of discharge, the hospital will render better service to referring physicians, and certainly thereby improve community-hospital relations and physician-hospital realtions.

Summary

A general description of a micro-computer implementation of a bedside computer for the Intensive Care Nursery is presented, with some of its basic features, and the authors justification for selecting North Star BASIC, and the SOL/20-North Star Disc combination. Costs, and cost justifications are also discussed.

AN AUTOMATED CONFERENCE MEDIATOR

David Stodolsky, PhD, Center for Educational Research
Stanford University, Stanford, CA 94305   415 494-2106

Abstract

A portable mini-computer based mediator, which facilitates speaker selection in conferencing is described. The system is a two-level communication network, permitting both anonymous and public responses. Up to 128 response units permit anonymous inputs on short travel switches with positive confirmation by tone signals. Each response unit contains a microphone which feeds a voice-activated switch in the mini-computer interface. The program samples six bits of information from each unit, five bits from short travel switches, and one from the voice-activated switch. The microphone of the current speaker is selected by a BASIC program responding to switch inputs.

The central controller consists of a PDP-8E computer with 4,000 words of memory, floppy disk storage, and a video compatible display. A crystal controlled clock with storage provides the system with a millisecond timebase. All facilities are accessible from BASIC programs.

The design objective was the creation of a tool for the study of automatically facilitated group interaction, which would not by its form induce limitations on group process. Precise delivery of tone and video displays and rigorous measurements of speech sequencing and voting behavior permit utilization of techniques previously limited to the psychology laboratory to be used routinely in the classroom.

Technological Advances in the Study of Group Interaction

In recent years, three different types of electronic systems for the study of group interaction have been presented in the social-psychological literature. Each of these systems represents a limited but significant technological advance in the study of group interaction. One class of systems is devoted to the measurement of speech sequencing, another to the collection of votes, and a third to delivery of stimuli to group members.

The speech sequencing measurement systems (Cassotta, Feldstein & Jaffe, 1964; Haley, 1964) are directed, as the name implies, to the measurement of vocal behavior of group members. Their development was motivated by a need to improve the accuracy of data collection and to cope with the mass of data generated by detailed recording of speaker turn-taking behaviors. These methods do not attempt to analyze the fine structure of vocal production such as tone, stress, or emotive quality of the voice. Their implementation is off-line so that no immediate feedback to the group members is possible as discussion proceeds.

The student response or electronic voting system represents another avenue of development in the study of group behavior. The depression of a switch or turning of a knob is the measured response in these systems. They have often been used as data collection devices (e.g., for the administration of tests) in the off-line mode. The more recent research attempts to use them in the on-line interactive mode to facilitate group dialog (Sheridan, 1973), problem solving (Chu, 1972), and learning (Rubin, 1970). They are limited in that they are broadcast information systems; there is only one channel, a television type display or verbal announcement to feed messages back to the group.

The stimuli presentation system is the third direction of research on group interaction (Aiken, 1965; Hastorf, 1966; Shapiro, 1963). In these studies, the type and quantity of verbal production of group members is manipulated by private presentation of information (e.g., RED light, GREEN light) to each person in the group. The group is observed by raters from behind a one-way mirror and differentially reinforced by the lights, which could denote "good insight-poor insight" or other dimensions of interest.

An integration of these types of development in a computer-based laboratory for the study of marital therapy has been reported (Thomas et al., 1973). This system accommodates only a few persons at once; lights are used for stimulus presentation and only a few buttons are available for responding. Vocal analysis is quite flexible using voice-operated switches to collect sequencing data and raters to determine "faulting," "positive talk," and other parameters in conversation.

Design Objectives

The primary objective in the design of this system was the creation of a tool for the study of automatically mediated group interaction which would not by its form induce limitations on group process.

Classroom Environment. The design attempts to bring the advantages of the automated laboratory to the classroom. To the researcher, the classroom setting is advantageous in that it provides groups of people in an environment which is less artificial than that found in the experimental psychology laboratory. To the instructor, it provides a welcome relief to the logistical problems of the large classroom. Pre-programmed modules can administer tests with immediate feedback to the student and summary statistics to the instructor for the evaluation of test questions and identification of problem students. In the response system mode, the instructor can obtain immediate feedback to lecture material. In the conversation mediation mode, classroom discussion can be facilitated.

Response Units. Because of the expense involved in such a computer-based system, its use has previously been limited to a few people at any given time. The approach described in this paper uses standard computer components for a majority of its implementation. A substantial economy is achieved, however, by the use of simple, hand-held response units, which can be easily constructed. The objective of 64 response units was selected to meet classroom needs, where the system could serve as a conversation mediator and electronic voting machine and could aid in the administration of tests or experiments. The fully expanded interface accommodates 128 response units.

Portability. The objective of a portable design is to permit use in different classrooms and in the laboratory. This feature combats under-utilization, a chronic problem with class-room response systems, in two ways. First, it permits the system to be placed at a location where it is likely to be used. Second, it permits easy rearrangement of seating patterns, an important variable in both teaching and research which greatly improves the system's flexibility.

Voice Capability. The response unit was designed to permit voice communication and thus experiments in which people are visually and acoustically isolated. This feature is also useful in face-to-face groups and combats the problem of under-utilization of electronic response systems. A microphone and voting system is more likely to be used than just a voting system. The benefit of voice amplification in large groups is obvious even to those

uninterested in voting. In some previous response systems, instructor/student dialog was limited by acoustical problems (Instructional Industries, Inc.).

Design for Equal Participation. Most previous systems for the mediation of group interaction have, by their form, limited equality of participation. A primary promoter of unequal participation in the large classroom is the public address system, which amplifies the instructor's voice. Not only does the instructor have a central position and authority, but his voice is enormously louder. Any system seeking to avoid imbalance in participation should provide each person with access to the public address system.

Anonymous response capability has been demonstrated to facilitate participation in both voting and speaking (Sheridan, 1973; Dybvig, 1972). The design objectives for the response collection unit specified characteristics rarely encountered in a typical computer terminal or voting system. The unit had to permit private voting responses and speech transmission capabilities.

Another limitation electronic voting systems often foster is unequal participation via centralized control or private presentation of aggregated response information. Centralized control is countered by use of a computer which can provide centralized coordination and distributed control. Private presentation is avoided by simultaneously presenting the aggregated information to all participants. The system can be configured for centralized control and private presentation of aggregated responses but this mode of operation is not forced upon the user by the hardware design.

Confirmation. There is no fail-safe capability in most electronic voting systems, and errors can accumulate without the knowledge of the operator or of the users. Private confirmation of responses in an electronic voting system is a necessity. This feature is important both in order to control errors and to maintain confidence of users during voting. For full flexibility, maximal response rate, and tight error control, the system should be designed to simultaneously receive and confirm response, that is, have a full-duplex capability.

Response Collection. Previous systems introduced limitations upon the rate users could transmit information because of the low response collection rate, limited alternatives, or lack of capability to transmit information simultaneously on the public and anonymous levels. In

the system proposed below, the limitations on information transmission are due only to the inherent information processing capabilities of the users.

One of the critical factors in the design of an electronic system is the selection of the total number of parallel response alternatives available to the users. Previous studies (Miller, 1956) have indicated that the amount of information transmitted for stimuli which vary on a single dimension is about 3.25 bits. As the number of stimulus dimensions is increased, the amount of information which can be transmitted in a single judgment also increases. A four-dimensional stimulus can be transmitted with 5 bits of information. For a stimulus with seven dimensions, a maximum of 7 bits of information can be transmitted by the user.

In use, the number of response alternatives is likely to be limited by memory for alternative codes rather than by judgmental capacity. A response indicating one of 32 categories is learned relatively easily. Englebart (1965) found that personnel can readily learn an alphabetic code on a five-bit device.

Latency measurement may have application in the classroom, as the following computer-assisted instruction experiment suggests. Judd and Glaser (1969) found that there is some increase in latency if an item remains unlearned over a considerable number of trials, while there is a large and significant drop in latency from the trial of the last error to succeeding trials. "There is also an indication that latency of the correct response begins to drop, while incorrect response latencies remain constant, or increase slightly on the last one or two trials prior to the trial of the last error" (Judd & Glaser, 1969). The study suggests that as the measure of correct response approaches its asymptote, the latency measure becomes relatively more sensitive. On the basis of latency, it appears that overlearning continues to have an effect beyond 10 trials past the TLE. "Post trial of the last error response latencies may provide means of determining the amount of over-learning practice which would be required for a particular subject to assume to assure a given degree of retention" (Judd & Glaser, 1969). To summarize, there are two major findings of the experiment: first, prior to the trial's last error, latency was a measure of test complexity but did not measure the development of associative strength. Second, during overlearning, response latency did appear to measure the continued development of associative strength. "Latencies may be quite useful in a situation in which instructional materials have been carefully programmed, so that correct response probability is always relatively high, but they would seem to be least useful in situations in

which the probability of correct response is very low. . . . These findings have definite implications for instructional decision making. . . . While it is known that overlearning increases retention, the amount of overlearning to be provided has always been a relatively arbitrary decision. The capability of measuring response latency may provide a means of determining the optimal amount of overlearning . . . for a particular student. . . . If it were found that response latencies were not shortened by instructional programs designed to bring a student to a high level of proficiency in a certain skill, the utility of instructional procedures might be questionable" (Judd & Glaser). The instructor using an electronic mediation system which can provide him with latency data would be in a superior position to evaluate instructional program effectiveness as compared to an instructor who had a system providing only correct response information.

What degree of time resolution must the apparatus have to be limited in accuracy only by the user? In the studies by Judd and Glaser (1969), response latencies were measured with an accuracy of + or - one millisecond. One millisecond is near the minimal repetition time of the neuron and thus one the basic response subcomponents which would have to be measured in a psychological experiment.

Voice responses contain a vast amount of information. They can be analyzed in terms of their duration and sequence, amplitude, tonal qualities, syntax, and semantics. Each of these possible measures overlaps the others in terms of the information it can yield. The above measurements are progressively more complicated to perform automatically. The minimal system requirement is the capability to determine whether vocalization is occurring at a given moment. This function is necessary if vocalization is to be used as an implicit request for the group's attention, as is the case in unassisted conferencing. Similarly, the cessation of vocalization is useful as a signal of relinquishing the group's attention.

An unobtrusive system must identify a speaker and enable the speaker's microphone within a psychological moment (approximately 1/10 second) so that it appears that the speaker's line is open whenever the person shes to address the group and is permitted to so by the rules of discussion.

The ability to detect voice onset and offset yields the ability to collect patterns of turn-taking within a group. These patterns can yield valuable information about the group's emotional tone (Haley, 1964). Accurate determination of vocal patterns within a single

speaker's comments can yield information as to emotional tone of that individual. Time resolution of 1/10 second is adequate to determine vocal patterns both within a given individual's comments and between vocalizations of different individuals.

The basic device I have specified thus far would accept 5 bits of information simultaneously and resolve the input to within a thousandth of a second. No human, however, can transmit 5,000 bits of information per second, though this is not considered a high data rate for a computer. The reaction time of a person on a task is related to the amount of information transmitted. The currently accepted function for disjunctive reaction time is $T = A + BH$, where H is the amount of information expressed in bits, transmitted by the subject per past response (Judd & Glaser, 1969). Nominal values are: $A = .2$ sec., $B = .2$ sec/bit.

The trade-off between reaction time and information transmitted results in a upper limit of 50 bits of information per second transmitted from stimulus to response (Hyman, 1953). In case of user errors, less information is transmitted; thus, the keypress rate may exceed this level. In one experiment, even cases where subjects transmitted well below the potential information transmitted by errorless responding, the reaction time was linearly related to the potential information in the task (Judd & Glaser, 1969). From the reaction time literature, it would be expected that response latencies would be a linear function of the amount of information actually being transmitted, but the latency data appeared to be more a linear function of the amount of potential information in the task.

The response rate may also diverge from the information transmission rate when previously learned sequences of user keypresses are delivered at a high rate as a result of a single judgment. In no case has rate of actual responses exceeded 50 bits/second, even for short periods (Flanagan, 1965; Posner, 1967). Thus a device capable of transmitting 50 bits/second with a resolution of 1 millisecond should not add any limitations to the rate of information processing in our man-machine system. The device described should accept 5 bits of information in parallel and transmit this data every tenth of a second.

Software. A crucial human factors decision in any automated system is the trade-off between complexity and flexibility in operation. In many computer systems, the user has to master many levels of complexity before he can effectively make use of the apparatus. Typically, the user must master the operating system, an editor, a programming language, and often the machine language and even hardware details. If the system cannot operate independently,

the user must learn the communication protocol of yet another machine. These complexities limit effective utilization of a system.

In order to minimize these problems, a high level computer language is specified for implementation of the mediation system. This language must be easy to learn and must have as built-in features a simple operating system and a rudimentary editing procedure. Thus, once the user has mastered the language, he can utilize the machine effectively with only a few auxiliary commands.

Response Units

Switch Arrangement. The switch arrangement used was motivated by research at the Augmentation Research Center, Stanford Research Institute (Englebart, 1965). This research group investigated stenotypewriter, standard typewriter, telegraph key, and the computer light pen as the input devices for man-machine communication. They eventually settled on the chord-keyset. This device uses the five fingers to generate 31 possible chords. The condition when no keys are depressed is reserved to indicate the previous chord is done. Englebart found that a clerk could learn to transmit about 20 words per minute after 20 hours of unsupervised practice with this keyset. A speed of 35 words per minute was attained with additional practice. This chord-keyset offers a good trade-off between speed of input and portability. In the conference mediation application, the chord-keyset offers a substantial margin of capacity, while maintaining portability by permitting participants to be unencumbered by fixed keyboards. In addition, it gives them the advantage of a free hand for taking notes or for other purposes.

A critical specification in an electronic vote collecting application is privacy of inputs. The switch characteristics in an isolated group situation are not critical. If one wishes to maintain privacy of a vote in a face-to-face situation such as that of the classroom, some precautions must be taken. One approach is the placement of the switch underneath a table to insure it is not visible to other members of the group. This switch may be activated by either the hand, or, if only a single switch is necessary, by the foot. Another method is a physical barrier, which covers the hand, isolating the movements of voting from the visual sphere of group members.

The keyset configuration selected does not require visual attention, and may be kept out of sight of the user and others present. Concealment would, however, eliminate the response unit's multi-function capacity as a

voting device and voice communication instrument. The use of a minimal travel or pressure sensitive switch dispenses with the need for concealment.

Switch Characteristics. Most switches can be obtained in either a momentary or a latching version. Latching switches are available as toggle or push-button types. The toggle type is inapplicable because of privacy considerations (Sheridan, 1971). A complete cycle with the push-button latching switch involves a press-release cycle to turn it on and a press-release cycle to turn it off. This is considerably slower than a momentary switch which is on when pressed and immediately off when released. Storage of information is done electronically. The momentary switch can also function as an automatic key which has fixed repeat time and senses a static state to determine an output. Examples are automatic keyers used to transmit Morse code and repeat keys on electric typewriters.

After a comprehensive survey of current switch technology, the electrical grating mechanical switch (Model TC-1) was selected for use in this application. The TC-1 is an economical, low force, low travel switch. "The multi-element, parallel contact design concept provides an extended conduction region. Its principle constitutes the first major advance in the principles of mechanical switching in nearly half a century" (Wild Rover Corp.). The switch is heat and shock resistant and has a life in excess of 10 million cycles at low load. In the handsets, I use the TC-1, series C switch capsule, .615 inches in diameter by .16 inch depth, weight 1.4 grams and the TC-1, M7, a square shaped, plastic packaged switch with the numbers 0 through 4 (.75 inches by .75 inches, 3 grams). The packaged switches were mounted with switch 0 under the thumb and switches 1 through 4 under the fingers 1 through 4. The switch travel for contact is approximately 10 thousandths of an inch, and the force required is minimal. Thus it is possible for the exposed finger to activate the switch without apparent movement. This feature permits completely private voting in the small group setting without the necessity for concealment. In operation, the person can rest his finger lightly on the switch without triggering a response, and then can trigger an input without apparent movement. This type of switch should not be confused with the "Contact" or capacitive relay switch. The capacitive relay switch is activated by proximity or touch; therefore, a person could not rest a finger on the switch without triggering it, a necessary requirement for this application.

Confirmation. In the previous sections, a switch configuration permitting anonymous or private response without the need for visual attention was developed. The needed confirmation should have similar features for full flexibility. The alternatives are tactile feedback or auditory feedback with provisions for privacy.

Tactile feedback can be given by either an inertial transducer (Sherrick, 1965) or a low level electronic stimulus delivered directly to a person's skin. The electronic stimulus would permit simultaneous sensing of Galvanic Skin Response, but the signal would be degraded by hand placement and movement artifacts. The ratio between absolute threshold and painful stimulation is 1 to 8 and permits 59 distinguishable steps or just noticeable differences (Saunders & Collins, 1971).

Conversion of the feedback signals into low frequency tones offers the best solution for tactile feedback. Most people have had at least minimal experience with tone-producing instruments. Complexity is considerably reduced by using an inertial transducer.

Auditory feedback can also be accomplished with tone signals, if their frequency is a magnitude greater than that required for tactile feedback. The major disadvantage of this mode of delivery of acknowledgment is the precautions which must be taken to insure privacy. If the device produces an airborn signal, its chamber must be sealed to the ear if privacy is to be preserved. An inertial transducer mounted on the mastoid bone which transmits vibration directly to the inner ear by bone vibration is commercially available (Radio Ear Corp., Cannonsberg, Pa.). This device, however, can become uncomfortable after extended use, and its placement is probably too critical for use without individualized instruction. An earphone type of device was selected for this application. When the user hears a tone signal in response to his keypress, he can be certain that the computer has received and acted upon his input. Tone response is a software function.

The current response unit is a full-duplex communication terminal, permitting simultaneous transmission of keypress information to the computer and tone signals from the computer to the terminal.

Handset. The design finally adopted was a modified telephone handset. The modification consisted of the installation of six switches, four placed between the mouthpiece and earpiece of the telephone handset so that the fingers of the hand could rest upon them. In order to permit the handset to be held in either hand, and therefore accommodate left-handed people, two thumb switches are mounted on opposite sides of the handset, just below

the earphone housing. When the handset is held in its typical fashion, the user's fingers and thumb rest on the appropriate switches. A substantial advantage in using the telephone handset is its familiarity to almost every participant.

This arrangement meets a number of restrictive criteria necessary for the flexible use of an electronic response collection system in the group environment. The telephone handset, which is readily available, can serve effectively as a two-way voice communication instrument, thereby permitting experimentation in which the group members are separated. In the face-to-face situation, the microphone element serves as an effective voice pickup unit and permits the amplification of the speaker's voice so it is easily heard by other group members.

Each handset is connected to the computer interface via a nine-conductor cable. Five conductors return switch depressions, one selector line activates that handset's switches, one line goes to the earphone, one to the microphone, and a common ground line is used by the earphone and microphone.

Voice Input. The telephone handsets used in this implementation were of the sound-powered type. The sound-powered telephone handset uses a dynamic element for its microphone, and thus permits a higher quality voice signal than that produced by the carbon button transmitter used in the conventional telephone.

An advantage of the telephone handset in the classroom type of situation is the maintenance of an appropriate mouth to microphone element distance. An alternative in the group situation is a conventional microphone. The conventional microphone public address type of system is, however, prone to acoustic feedback which can cause severe oscillation, resulting in an overload of the various components of the system.

A further problem with the conventional microphone system is spill-over, which occurs when one person speaks exceptionally loud. Under this condition, a number of microphones can appear to be on, to the computer, thus causing confusion in the determination of the actual speaker. This is a problem even in groups of three or four people. Attempts have been made to design systems which cancel this interference by electronic summation and cancellation (Cassotta, Feldstein, & Jaffey, 1964). This cancellation technique is only effective for a small number of speakers and a fixed placement of microphones. A substantial increase in complexity would be required for a large group of people in which microphone position was not preset or was permitted to change during group

discussion. A second approach to facilitate the identification of the actual speaker is the use of the throat microphone (Haley, 1964). The throat microphone, however, produces an unnatural voice sound of lowered intelligibility. It can best serve as an adjunct to a primary voice transmission channel.

Computer System

Selection Factors. The selection of a mini-computer was based on three major factors: 1) computational needs, 2) software availability, and 3) cost. Since the primary function of the mini-computer is to collect and respond to switch depressions by group members, the computational load is quite light. The voice-operated switches transmit data to the computer at the highest rate. To continuously collect data on vocalization of all group members at one tenth of a second intervals would require the computer to sample 320 times per second, since two units are selected simultaneously. This is the maximum sampling rate for vocal inputs and switch inputs, since both of these inputs are sampled by a single operation. A rate of 320 samples per second is quite low for any modern mini-computer.

The second major factor in computer selection was availability of software. The computer selected should have a large number of installations, preferably in educational institutions, and should have an active user society, which maintains and updates an extensive software library. Software support by the manufacturer and a selection of appropriate peripheral equipment are also important factors in mini-computer selection. As a result of these considerations, the PDP-8/E was selected as the main frame for this system.

The availability of a Digital Equipment Corp. PDP-10 on campus also contributed to this decision. Programs can be completely prepared on the PDP-10, which has available a PDP-8 simulation program and cross-assembler. Thus the advantages of a large time-sharing system are available for program preparation, but the system is not needed for operational use. All programs run in stand-alone mode on the mini-computer system.

Peripherals. A serial input/output board was used to drive a video compatible terminal (Ann Arbor terminal, Model 204). This terminal was selected because of its low price and a format appropriate for the display of information to a large group. The unit generates a television type display with 16 lines of 32 characters each.

The Ann Arbor terminal 204 disp y unit is driven at a 1200 baud rate. This permits 120

characters per second to be transmitted to the screen. The terminal permits absolute character addressing; thus, any character on the screen can be changed in one fortieth of a second. The system is equipped with eight small video monitors. For larger groups, standard television receivers tuned to channel four can be used, since the terminal simultaneously generates a radio frequency signal on this channel and a direct video output to the eight nine-inch monitors. This display unit is also equipped with a teletype-like keyboard, which serves as the operator's console.

A secondary storage medium in a mini-computer installation is often critical for effective operation of the system. The storage device should be fast and flexible in order to act as an extension of the minimal memory available on the mini-computer. Unless the medium is removable, the system is limited in use to only a few users, for storage space can become rapidly exhausted. The removable diskette makes the system much easier to use since it makes possible stand-alone operation and provides an instructor with a medium which, when mounted, provides a memory of previous class sessions. Program selection by simply inserting an appropriate diskette is also possible.

In many cases, the cost of peripheral storage devices exceeds the cost of the computer itself. The only device capable of meeting the above specifications at minimum cost was a movable head diskette or "floppy disk" system. The unit selected permits the storage of 131,072 12-bit words per disk unit. The maximum transfer rate is 15,000 words per second, and transfers are made in blocks of 128 words. This configuration permits the transfer of data to and from the main memory of the computer to be time-sequenced with data collection operations. In the current configuration, the longest possible transfer from disk would take approximately 30 milliseconds. Thus, even when sampling data 10 times per second, the computer can transfer information to disk memory without the loss of any data samples. The alternative, a direct memory access module, is an expensive peripheral controller which permits data transfer to be interleaved with computing operations.

The computer configuration included a teletype interface. The teletype permits the loading of paper tape when building a software system, running diagnostic programs, and the transfer of information from other machines. It also functions as a hard copy listing device.

Interface Hardware

Clock Board. Because of the high rate of information transfer to and from the disk, the computer cannot perform any other operation while a disk transfer is in progress. In order to maintain accurate timing, a special cumulating clock was constructed. A foundation module (11-DE-8) was purchased from Douglas Electronics Inc., San Leandro, Ca. The fundamental time source for this clock was a crystal oscillator producing a square wave output at 10 megahertz. This signal was divided by 10,000 to yield a pulse per millisecond. Each pulse causes a 12-bit counter to increment. Periodically, the counter is read by the computer and cleared. Thus, the clock can accumulate 2,048 ticks, or milliseconds, before overflowing. The most significant bit of the 12-bit counter is reserved to indicate overflow. If this bit is set on, it cannot be turned off, except by the operation which reads the counter into the accumulator of the central processing unit. This positive indication of overflow conforms with the standards specified by Creelman (1974).

The same foundation module serves to transmit 12 bits of information to the interface. These outputs control microphone selection and tone selection for feedback.

Response Unit Interface. The majority of the hardware design effort was directed toward the interface for the electronic voting units. The interface can be divided into five major sectors.

The switch input lines are connected to a common node through diodes. Thus, switch #1 from all of the odd numbered handsets connects to the same input terminal on the 12-bit I/O board. At any given moment, one pair of hand units is selected. Each unit delivers five finger switch closures and one voice-operated switch closure to its half of the 12-bit input board. The input interface is jumpered to indicate a transition on an input line. The use of the input lines to indicate transitions eliminates the problem of switch bounce by storing any on-transition that occurs during selection of a unit.

The 12-bit output section of the parallel I/O interface board is used for switch input and for tone feedback. The four least significant bits are bused to one-of-16 selector integrated circuits. The fifth bit is used to enable one of two selector circuits.

The six most significant bits are similarly arranged to feed a total of up to eight selector circuits. The sixth bit of the 12-bit parallel output is reserved for system expansion. It can be used to select a high or low bank of selectors. It would operate simultaneously on the unit voting selectors and on the feedback selectors.

The output register on the clock board provides the microphone selection information. The seven least significant bits are reserved for microphone selection via up to eight one-of-16 selectors. Any one of a possible 128 microphones can be on at a given moment.

The five most significant bits select any combination of five tone generator outputs to be transmitted to a given earphone via the feedback selectors. Thus, the feedback tones can occur in any combination, as is necessary with a chord-handset approach. The actual tones are derived from the timing chain of the clock. Tones derived from the clock chain are accurate to five digits, which is well in excess of the accuracy needed for this feedback application.

Voice Signal Processing. The voice-operated switches are integrated with the speaker selection circuitry. A novel MOS gating method is used to minimize system cost and response time and to maximize reliability. The microphone signal from a given hand unit feeds into an operational amplifier, which is set for a gain of 1,000. This amplified signal is then fed into the voice gating circuit and the voice-operated switch circuit.

The voice gating circuit is comprised of a three input nand C-MOS gate. The audio signal is fed to one input. The other two inputs are tied together and are fed from the microphone selector lines. The outputs from these microphone selector gates are summed and amplified before being sent out along the earphone bus. The earphone bus feeds to the PA amplifier in a face-to-face group and into the individual earphone transducers for use in the isolated group condition. The use of a C-MOS gate in this application is dependent upon the device's capability to function as an analog or digital circuit element. When the device is selected, it serves as an amplification stage. When it is not selected, it is completely cut off, thus not permitting the transmission of audio information.

The voice-operated switch circuit contains two operational amplifiers. The first amplifier drives a rectifier. The derived DC potential, filtered with a one tenth of a second time constant, is in turn amplified and used to trigger a one-shot. The one-shot drives a two input AND gate. The other input to the gate is the unit selector line. The gate output is treated as the sixth switch input from a given unit.

Connectors and Mounting. Cables provided by the interface manufacturer are used to bring parallel I/O signals to the interface. DEC compatible sockets are used on the interface board itself (Douglas 26-DE8 wire wrap sockets).

The electronic voting units plug into the interface via 36 pin Molex sockets. Each unit uses one nine-pin column. Thus, a group of four units connects simultaneously to the interface. The back of the connectors is used for mounting the busing diodes. All circuit components are mounted on a perforated vector board. This board is in turn mounted to a standard 8½ inch relay-rack panel using two inch standoffs. In the mounted position, all circuitry is between the perforated panel and the rack panel. Individual potentiometers are supplied to adjust the level of input needed to trigger the voice-operated switch. This permits different types of microphone elements to be used in the voting units. Wire wrapping was selected as the method of interconnection of circuit elements.

Software

BASIC Language. The BASIC computer language was selected based on criteria of ease of use, compatibility with other systems, and availability for uses with the current memory limitation. The 4K memory made it imperative that the software system make effective use of the diskette secondary storage media. POLY-BASIC (DECUS 8-195), a BASIC dialect, fits these requirements. It is almost identical to BASIC as described in Kemeny and Kurtz (1967). BASIC in this software package is implemented by compilation, and programs can be larger than core memory. In addition, the CHAIN command permits one program to call another. Thus, the sequence of programs limited only by size of diskette storage can be executed under program control.

Language Additions. POLY-BASIC had to be modified to accommodate the diskette secondary store. The modification is transparent to the user. The POLY-BASIC READ and WRITE commands were modified to handle a "DATA" file on disk instead of in the temporary storage area from which programs are executed. The change permits as many as 40,000 numbers of characters to be reserved for data storage by use of the added OPN function.

The 12-bit parallel I/O interface is controlled by the new function SRD, which interrogates a specified handset pair. The function is called with the 12 least significant bits of the argument specifying which output line to select and returns with a 12-bit word containing a pattern with the ones indicating switches closed.

The hardware clock is interrogated by the pseudo-interrupt routine which updates a 24-bit memory storage location indicating elapsed time to the millisecond. The CLK function interrogates this location. The output register

on the clock board which controls the micro-phone select and tone select lines is loaded with a 12-bit word by the MIC function.

Undocumented Features. The switch re-gister on the front panel can be interroga-ted by the KEY function.

Pseudo-Interrupts. The complexities of interrupt handling are avoided in the POLY-BASIC software by the use of pseudo-interrupt software. After each line of code is executed, or block of data is transferred, the system polls all peripherals to deter-mine whether service is required. This pro-cess updates the internal clock time and im-plements buffered terminal input and output. Thus, a line of up to 32 characters can be buffered to the video display while proces-sing continues.

Comparison with Other Systems

A computer system for group instruction and research has been presented by Scholz and Holly (1975). This system has a subs-tantially different orientation; it also re-presents a different hardware approach to vote collection than that presented here. Their unit uses an input multiplexer, as opposed to the selector system described here. The input multiplexer has the advan-tage that it permits more rapid scanning of the input lines in situations where line ca-pacitance could cause slow response times. This has not been a problem in the system described in this paper, however. The input multiplexer approach must use a selector circuit for each bit of the input signal. In our system, six times as many selector circuits would have been necessary if the in-put selector approach had been taken. In the Scholz and Holly system, only three se-lector circuits per unit are required; thus, the difference is not as significant. The useful feature in their system not used in the currently described vote collection sys-tem is an auto-polling circuit. This permits the interface to autonomously scan the voting units, while the computer is performing other tasks. In this condition, their system gen-erates an interrupt when a vote is received. In the system described in this paper, the use of the interrupt system is reserved. All polling proceeds under program control.

A number of electronic voting systems are available commercially. These computer controlled voting systems have in most cases evolved from previous systems which did not use the computer for their control. Therefore, the configurations are somewhat redundant, and, in many cases, the computer is not utilized effec-tively. Most of the electronic voting systems in current use are simple tallying mechanisms. Even when the systems are controlled by compu-ter, the operations they perform are extremely simple.

References

Aiken, E. G. Changes in interpersonal descriptions accompanying the operant conditioning of verbal frequency in groups. Journal of Verbal Learning and Verbal Behavior, 1965, 4, 243-247.

Cassotta, L., Feldstein, S., & Jaffe, J. AVTA: A device for automatic vocal transaction analysis. Journal of Experimental Analysis of Behavior, 1964, 7, 99-104.

Chu, Y. Study and evaluation of the student response system in undergraduate instruction at Skidmore College. Educational Support Project of the General Electric Company, Schenectady, New York, 1972.

Creelman, C. D. Software management of timing in computer-controlled on-line experiments. Behavior Research Methods & Instrumentation, 1974, 6(5), 488-492.

Dybvig, H. E. Some observations on the use of the student response system in teaching radio and television classes. Educators Review, Instructional Industries, Inc., New York, 1972.

Englebart, D. C. Augmenting human intellect: Experiments, concepts, and possibilities. (Contract AF 49(638)-1024). Menlo Park, California: Stanford Research Institute, March 1965.

Flanagan, J. L. Speech analysis, synthesis and perception. Berlin: Springer-Verlag, 1965.

Haley, J. Research on family patterns: An instrument measurement. Family Process, 1964, 3, 41-65.

Hastorf, A. H. The "reinforcement" of individual actions in a group situation. In Krasner and Ullman (Eds.), Research in behavior modification. New York: Holt, Rinehart, and Winston Inc., 1966.

Hyman, R. Stimulus information as a determinant of reaction time. Journal of Experimental Psychology, 1953, 45, 188-196.

Instructional Industries Inc. Personalized instruction and mass-lecture planning at Monroe Community College. Educators Feedback, Ballston Lake, New York.

Judd, W. A., & Glaser, R. Response latency as a function of training method, information level, acquisition, and overlearning. Journal of Educational Psychology, 1969, 60(4), 1-30.

Kemeny, J. G., & Kurtz, T. E. BASIC programming. New York: John Wiley and Sons, Inc., 1967.

Miller, G. A. The magical number seven plus or minus two: Some limits on our capacity for processing information. Psychological Review, 1956, 63, 81-97.

Posner, M. I., & Fitts, P. M. Human performance. Belmont, California: Brooks/Cole Publishing Company, 1967.

Rubin, S. Student personalities, classroom interactions and the evaluation of an anonymous feedback system in college classrooms. Thesis, Purdue University, 1970.

Saunders, F. A., & Collins, C. C. Electrical stimulation of the sense of touch. The Journal of Biomedical Systems, 1971, 2(7), 27-37.

Scholz, K. W., & Holly, B. A 64-station computer-assisted teaching and research facility. Behavior Research Methods & Instrumentation, 1975, 7(3), 301-304.

Shapiro, D. The reinforcement of disagreement in a small group. Behavior Research and Therapy, 1963, 1, 267-272.

Sheridan, T. B. Technology for group dialogue and social choice. Proceedings of the Fall Joint Computer Conference, 1971, 327-335.

Sheridan, T. B. Progress report of the M.I.T. Community Dialog Project, 1973.

Thomas, E. J., Walter, C. L., & O'Flaherty. Computer assisted assessment and modification: Possibilities and illustrative data. 100th National Conference of Social Welfare, May 30, 1973, Atlantic City, New Jersey.

Wild Rover Corporation. The TC-1 principle. TC-1 Touch activated switches, keyboards, & moving key systems. Norwood, New Jersey.

SYNTHETIC SPEECH FROM ENGLISH TEXT

D. Lloyd Rice
Computalker Consultants
Box 1951
Santa Monica CA 90406

A flexible rule processor and a set of rules is described
which converts normally spelled English text into the phonetically
spelled strings needed for a speech synthesizer system such as the
Computalker CT-1/CSR1. String match-and-replace type rules locate
common letter sequences of normal spelling and substitute the appro-
priate phonetic sequences. The rules are stored in a way that allows
easy modification with a standard text editor and open-ended upward
expansion of the system to a general phonetic dictionary. Problems
of such a system are discussed, including the proper assignment of
stress patterns and ways of handling ambiguous pronunciations.

MACHINE RECOGNITION OF SPEECH

M.H.Hitchcock, President, Phonics Incorporated, POB 62275, Sunnyvale CA, 94086

Introduction

Computer recognition of spoken words and phrases has received a good deal of interest in the last few years as a sort of "ultimate" form of man-machine communication. The available technology however has fallen considerably short of the dreams of automatic dictation equipment, natural language voice programming and a myriad of other applications requiring a degree of technical sophistication not yet widely available. Is voice control of machines realistic today, and if so, what kind of applications lend themselves to today's technology? How well do they have to work and how much should they cost - these are the questions to be addressed by this paper.

Is voice control of machines a realistic goal given the type of speech recognition equipment currently available? First of all, what type of equipment is available. All stand alone speech recognition systems commercially available today share some attributes. All systems operate on discrete words or short phrases varying in maximum length from ½ to 2½ seconds depending on the specific system. None can handle continuously or normally spoken text. The practical limits of vocabulary size are from 16 to several hundred words depending on price/performance requirements. All systems require that the user train or teach the system a set of vocabulary words. This training most often consists of letting the system "hear" several examples of each word, forming a prototype pattern of each item. All systems operate in real time. Most systems are relatively insensitive to extraneous noises and have the ability to reject words which they have not been taught. Output is available as either parallel or serial data identifying the index of the word spoken. From a technical standpoint, speech recognition systems fall into two main categories. Parametric systems rely on extracting specific features of the speech signal which relate to linguistic units such as phonemes. These features are then concatenated and word recognition attempted by analyzing the strings of these features. The main advantage of this technique is that a relatively small number of features may be used to identify a large number of spoken words since it is essentially a sequential process. For example, let us assume that using some sort of feature extraction process, the english phonemes, only 46 in number, can be correctly identified. Any word may then be identified simply by recording the recognized sequence of phonemes and using a table-lookup to retrieve the result. Features which might be used to identify phonemes or characterize whole words include fundamental voice frequency or pitch, frequency and amplitude of vocal resonances called formants, noise-like segments called fricatives and many other well known linguistic units. The biggest problem with this technique is that many of these voice characteristics, being somewhat subjective in nature, may be quite difficult and at times impossible to identify in the acoustic signal.

Non-parametric speech recognition systems on the other hand look at the speech signal as simply a complex pattern in the time and/or frequency domain. These signals may be reduced to a manageable number of data points by using information theoretic techniques to remove redundant information. The resulting data sets are then associated with spoken words by applying any of several well known pattern recognition algorithms. These systems would work equally well for other acoustic or electrical inputs since no specific features of speech are looked for. The result in either case is a system which can identify a limited number of spoken words with a reasonably high degree of accuracy. Such automatic speech recognition systems are available for applications including industrial control, aids for the handicapped, remote data entry/retrieval, simple machine control and game playing. Specific devices currently being marketed include: VIP500 - Threshold Technology Incorporated. A high quality system capable of 99% recognition rates with a 20-40 word vocabulary. This is a parametric system used mostly in industrial applications such as sorting, data input for quality control, or in hands-busy machine control situations. Its price of $10-30,000 puts it out of the

range of most home computer users.

VDETS - Interstate Electronics Corporation. This is a non-parametric based system roughly equivalent to the Threshold Technology unit in price and performance. It was originally designed and marketed by Scope Electronics Incorporated. It is unique due to a high level voice oriented software package that allows the customer to fairly easily adapt the system to specific application requirements.

CGM16 - Centigram Corporation. The CGM16 is a non-parametric system capable of recognizing a 16 word vocabulary with an accuracy of about 95% for a practiced user. Priced at $3800, it will probably find use in simple industrial machine control applications and perhaps among the wealthier home computer users.

SR/8 - Phonics Incorporated. Another non-parametric system with about the same performance as the CGM16. A useful feature of this system is a user controlled threshold which allows the recognizer to reject words or sounds it has not been trained to respond to. Additionally, it utilizes a dynamic programming algorithm which makes it relatively insensitive to word duration changes which might be experienced over long term usage. It's price of $995 for a complete unit or $550 for the OEM/ hobbyist makes it the lowest priced stand-alone speech recognizer available today.

What kinds of problems may be solved by systems such as these and how well do they have to work? Perhaps the best insight into this question can be gained by looking at specific examples of speech input systems currently in use.

One of the products manufactured by Owen-Illinois Corporation in Pennsylvania is glass television faceplates. As part of a quality control process, dimensional measurements must be taken requiring physical manipulation of both the faceplate and measuring instruments. The resulting measurement data was then recorded, keypunched, and later analyzed for any measurements out of tolerance. This was a labor intensive operation, prone to error and with a built in time delay before information could be sent back to the manufacturing line to correct out of tolerance runs. The system was streamlined by employing a speech recognition system, a Threshold Technology Inc. Voice Data Entry System. With the speech system, which includes a minicomputer, an inspector enters the mea-

surement data as quickly as measurements are taken. Reports of out of tolerance conditions are available immediately and thus manufacturing line changes may be implemented before long runs of defective parts are made. In operation, the unit is pre-programmed with the required measurement sequence and tolerance limits for all products to be inspected. After manually entering via TTY heading information, including product to be inspected, the inspector's name and any changes to be made in the pre-stored tolerances, the inspector is ready to start an inspection run. The voice system leads him through the process, displaying received measurements to allow for their correction if necessary and immediately flagging any out of tolerance events. When the inspection run is complete, a report is provided on the TTY including all necessary heading information and a statistical analysis of all measurements taken. The entire operation is performed quickly and accurately by a single inspector, justifying the pricetag of $30,000+.

The sorting of baggage at the United Airlines Terminal at Chicago's O'Hare Airport was a two man job. One man physically manipulated pieces of luggage, reading the destination tag and sliding it onto a loading mechanism to a carrousel sorter. As he did this he called the destination to a second man who entered it via a numerical keypad, providing information to the sorter to allow it to offload the luggage at the appropriate place for shuttling to the aircraft. Using a voice input system, again a Threshold Technology Incorporated system, the same man who handles the baggage speaks the destination, verifies its accuracy and causes it to be entered into the sorter's memory. Additionally, instead of entering the destination by two digit numeric code as was done manually, destinations may be entered by airport name such as LAX, SFO or IAD. This further reduces the error rate and makes the man-machine communications aspect of the job more natural. Since this is a multi-shift operation and one man is eliminated per shift, the high cost of the system is acceptable.

A sophisticated system called VOTECS for Voice Operated Teletypewriter and Environmental Control System, produced by Scope Electronics Incorporated has been used by the Veterans Administration Hospital in Richmond VA to allow handicapped persons access to a timeshared computer. Using this system, a spastic quadriplegic student has demonstrated

the ability to write, debug and operate computer programs communicating entirely by spoken commands. While each key of the typewriter may be activated by speaking the appropriate character or symbol, not all words or commands must be spelled out. Certain commonly used "BASIC" commands such as DATA, READ, PRINT, NEXT and others may be spoken as complete words. The resulting throughput rates compare favorably with manual entry. The individual using this system may also voice control certain functions in his environment such as lights, radio, TV or bed position. For persons temporarily or permanently disabled, voice data entry and control provides a degree of freedom not previously available.

How much should speech recognitions systems cost? In nearly every case of systems performing some real function in industry, the only acceptable criterion has been, does it significantly reduce labor cost? If a man may be replaced by a machine, the machine's cost, including maintenance must be such that it will pay for itself in 1-2 years. In some cases, increased performance is enough of a factor to justify cost but this simply means increased output with no increase in labor. This has been true because prior to last year, speech recognition systems typically cost $20-40,000. It is hard to justify the use of a system like this simply because it is easier or more natural to use. This situation is rapidly changing however as prices for complete recognition units continue to drop dramatically. Technology that was available 8-10 years ago for $20,000 is now available for less than $1000. The question of speech recognition system cost is tied fairly closely to system performance. For applications requiring large vocabularies and/or very low error rates, system cost can be expected to range from $10-50,000. These applications include most industrial requirements such as remote data entry, computer aided design and sortation. For applications such as equipment control, aids to the handicapped, and game playing, smaller vocabularies are typically used and higher error rates tolerated. However, once the error rate exceeds 6-8%, the system becomes quite unworkable, regardless of cost. Fortunately, the lower priced recognition systems, such as the Phonics SR/8 and Centigram CGM16, offer adequate performance for many simple control tasks such as environment control, control of

simple devices such as timers or locks, and noncritical data entry tasks as might be encountered by the amateur astronomer, photographer or radio operator.

What of the future of machine recognition of speech? Much of the work done in universities and corporate research labs indicate significant improvements to speech recognition technology. "Speech Understanding Systems" allow the user to retrieve information from a computer via his natural language, speaking normally rather than having to use one of a small number of commands spoken very concisely. Vocabulary sizes have been pushed to over 1000 items and teaching the system initially has been simplified and in some cases eliminated altogether. Systems have been designed which adapt to the users voice changes constantly, thus keeping recognition performance high for very long periods of time. A great deal of work remains to be done before systems such as these may be brought to bear in real applications. Not only are they very expensive, requiring large amounts of memory, but most do not yet operate in real time. It will not be long before the semi-conductor industry provides the hardware necessary to allow practical implementation of some of these advanced systems. In the meantime, as microprocessor speeds increase and memory costs go down, both the recognition accuracy and vocabulary size may be increased. Certainly, the technology is available today to let people verbally control machines when their hands or eyes are busy, as a way to simplify communication with computers or simply because it is fun.

SSTV GENERATION BY MICROPROCESSORS

Clayton W. Abrams K6AEP
1758 Comstock Lane, San Jose, CA 95124

Slow Scan Amateur Television (SSTV) is a low resoltuion/bandwidth video communications method.

SSTV is a 1000:1 reduction in bandwidth from normal TV. This means that an SSTV picture will have a bandwidth of 3 KHz, which is compatible with low-cost tape recorders and amateur radio transmitters.

SSTV was first transmitted over the air in 1958 on the old 11-meter band. After numerous experimental contacts, the FCC authorized HF transmission of SSTV in 1967. Since that time numerous amateur radio recievers and TV camera converters have appeared on the commercial market. Now that commercial microprocessors are becomming available for reasonable prices, there use for amateur radio applications is a natural evolution.

Obviously, the use of microprocessors requires a merge of hardware and software in a relatively complex manner. Today, I would like to talk about three 6800 computer programs which I have written for my SWTPC 6800 computer system for amateur radio SSTV.

SSTV Character Generator Program

The easiest method of microprocessor SSTV generation is a character generator. Last year at the Computer Faire, I spoke about my software approach to SSTV character generation. The details on this approach were published in June 1977 "73" Magazine, and in the Proceedings of The First Computer Faire.

SSTV Titler Program

The second method of character generation was described in detail in October 1977, "73" magazine. In this method I inserted SSTV character lines into SSTV pictures. This method was entirely a software approach and attached a SWTPC 6800 to a SSTV Scan Converter. The only trick to this approach is to slave the microprocessor to scan converters horizontal and vertical sync pulses For those of you who are not familiar with scan converters, they are devices which convert a fast scan TV camera directly to SSTV by analog and digital techniques. See Fig. 1.

The computer software of the Titler program allows for the insertion of up to 9 characters on anyone of 10 locations of a SSTV picture. A total of 10 character lines can be stored in computer memory and inserted in the SSTV picture in any order on up to 10 successive picture frames.

Figure 2 is a block diagram of the SWTPC 6800 Computer/SSTV Scan Converter interface, which consists of only 4 wires.

The vertical and horizontal pulses provide the proper inputs to the PIA, which allows the computer program to count SSTV lines and insert the SSTV characters at the correct location in the picture. The black and white control lines are attached to the D/A converter in the scan converter and allows the selection of black on white or white on black characters with background.
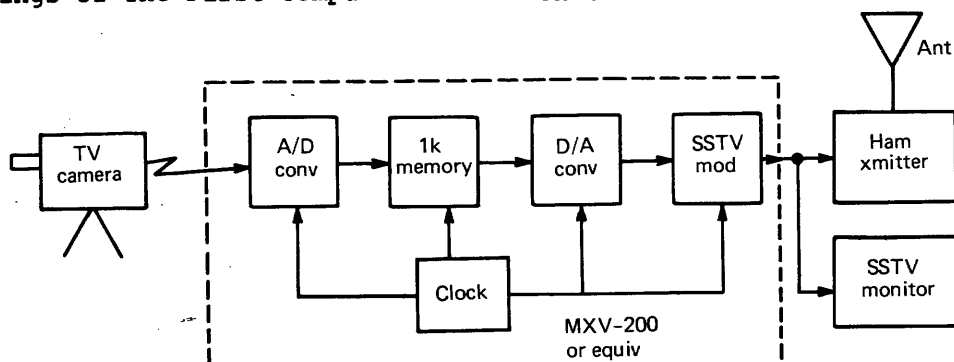
Fig.1 Scan converter
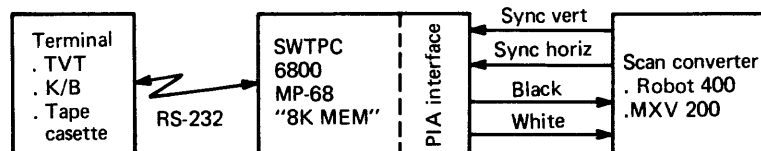
Fig.2 SSTV titler program computer interface

An interesting part of both of my character generator programs is that the character dots are in computer memory, which means total flexibility. Character sets can be exchanged easily or even dynamically with slight program changes to the software.

SSTV Picture Enhancement Program

A new and exciting program which I developed last summer, also to be discussed today, is my SSTV picture enhancement program. This program will be published in 73 Magazine this spring in two parts due to its size and complexity. However, today I would like to discuss its highlites and explain the computer techniques used.

The field of image enhancement is not a new one. Numerous commercial TV firms have used these techniques which I will present today. My main intention in this program was to apply these techniques in a simple straightforward way toward amateur radio SSTV generation.

I'll start by listing some of the highlites of my software package:
1. Allows the reception of an SSTV picture over ham format, which can be connected directly to a ham transmitter.
2. Transmits the picture in computer memory in a SSTV format, which can be connected directly to a ham transmitter.
3. Prints a hard copy picture of the picture in computer memory on a SWTPC PR-40 printer. ASCII characters are substituted for the picture gray levels and printed.
4. The picture in computer memory can be modified or transmitted with the following enhancements:
   - remove noise from successive pictures received
   - add contrast to the picture
   - zoom in on 5 locations on the picture
   - reduce the picture gray level content from 16 to two
   - produce negative or inverted pictures

Most of the above enhancements were accomplished by computer software

and a special interface card which cost less than $75 to construct. However, to accomplish these tricks you must first obtain a 6800 computer.

I would like to first discuss the computer interface hardware. The first piece of hardware required is a SSTV received. Many units are available at moderate costs. These units are required to demonstrate the SSTV video to a varying DC level, and to obtain horizonatal and vertical sync pulses. I installed a pre-amp board in my MXV-100 monitor to scale the video DC voltage swing to 0-5 volts, which was necessary for my computer interface card.

The next piece of hardware required is the special interface card (Fig. 3). It consists of an analog-to-digital converter (A/D), digital-to-analog (D/A) converter, and a SSTV modulator, the cards block diagram is as follows.

The card uses off-the-shelf Datel modules and standard IC's. This interface card can be easily duplicated.

The modules are used to input/output the analog and digital to the outside world. The analog modules are controlled by the PIA chip which is under program control.

It is now approrpiate to discuss how the TV picture is formatted in computer memory and how the enhancement techniques are achieved. The TV picture is divided into small picture elements which are called pixels. A horizontal scan line was defined to contain 128 of these pixels. Since an SSTV picture contains 128 lines, the digitized picture in computer memory will contain approximately 16 K picture elements. Each picture element was defined to have 16 gray levels or 4 binary bits. If the 4 bits or nibbles were packed into bytes by the computer software, a TV picture could be contained in 8 K of computer memory.

The biggest trick of the entire project was to place the SSTV picture in computer memory. I took a software approach to the problem by sampling the A/D converter every 520 microseconds. Since the A/D converter had a conversion rate of less than 50 microseconds, I had sufficient time to format the
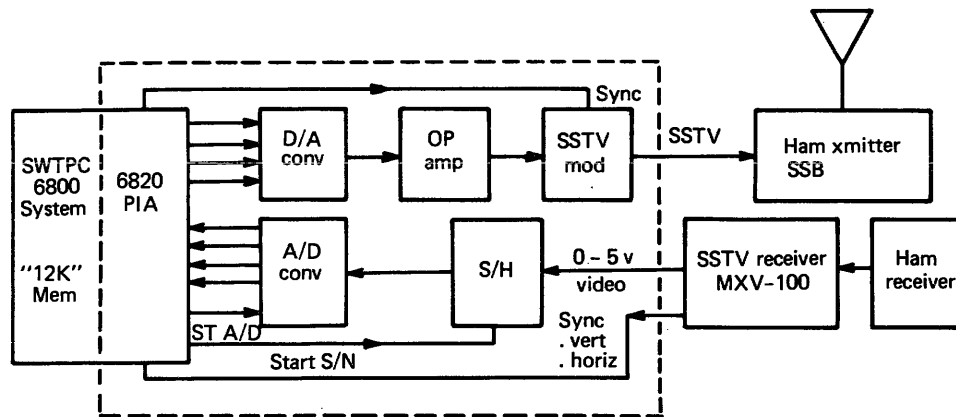
Fig.3 SSTV interface card

pixels and place them into memory before the next pixel time.

The next project was to convert the digital pixels to analog SSTV. The pixels were unpacked and shipped to the D/A. This caused the SSTV frequency to shift between 1500 and 2300 Hz which is the SSTV black and white frequencies. The PIA was additionally connected to the SSTV modulator FSK control line which creates sync pulses of 1200 Hz. All pulses and timings are controlled by software delays. It is now appropriate to discuss some of the enhamcement techniques used.

Noise Reduction

This computer programming routine averages received pixels with those in computer memory. The averaging is accomplished by software on a real time basis. The resultant effect is to reduce the random noise received by the square root of the number of pictures received.

Hard Copy Printing

This programming routine prints hard copies of SSTV pictures on a PR-40. Seven ASCII characters are substituted for each gray level printed.

Contrast Enhancement

This routine of the program adds contrast to dark SSTV pictures received. The following algorithm was used:

original pixel	darkest pixel	original - darkest	X2 enhancement in contrast
9	2	7	E

The routine first finds the darkest pixel in the center of the picture's center 10 scan lines. This value is

then used to modify all pixels in the computer memory.

Zoom Enhancement

This routine zooms in on pictures in computer memory with a 2 X magnification. The zoomed picture is then transmitted over the air. The process is quite simple as follows:

original byte

A | 4

SSTV line 1   A | A   4 | 4

SSTV line 2   A | A   4 | 4
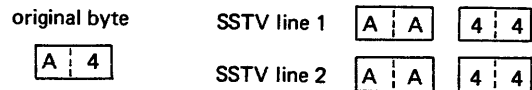
Five locations can be selected in the picture for zooming by the program.

Other Enhancements

The negative pictures were produced by complementing each pixel before transmission. Additionally, a binary effect was achieved by converting the picture elements to either black or white before transmission. The binary clip level was selected by a program constant in computer memory.

The preceeding techniques show how powerful microprocessors are and how naturally they fit into the SSTV applications.

I would like to encourage other radio amateurs to explore the use of computers for other ham radio applications. My experience has been very rewarding and I have never had so much fun with the hobby before.

I'm sure the applications are just starting and the future will be very exciting.

A REAL TIME TRACKING SYSTEM FOR AMATEUR
RADIO SATELLITE COMMUNICATION ANTENNAS

John L. DuBois,Dytron Inc. 241 Crescent St.
Waltham Massachusetts   02154

Introduction:

This paper describes a hardware-software system for pointing an amateur antenna at a polar orbiting satellite such as OSCAR-7 and automatically tracking it during a pass.

The program is written in BASIC and performs all computations necessary for tracking once given the pass equator crossing time and longitude.

The system described uses a S-100 bus microcomputer operating with BASIC. Specific hardware is described consisting of A/D conversion and parallel output for interface with antenna azimuth and elevation rotators. the system requires a real time clock in the computer and a commercially available S-100 buss clock board is used in the example.

Background and Objectives:

There have been a number of good articles (1,2,3) written on the subject of computing the track of low altitude polar orbiting satellites such as OSCAR-7 and NOAA-5 for amateur radio purposes. There have also been several ingenious circuits published (4,5) for pointing appropriate antennas at the satellite track from pre-programmed media such as tape cassettes or paper tape.

The obvious combination, however, in view of the current explosion of microcomputer applications between doing the track computations and managing the pointing hardware all by microcomputer has not yet, to the authors knowledge, appeared in the amateur literature. This is such an appealing application after one has experienced the need for six arms in trying to track an OSCAR pass, tune the receiver, spot a desired frequency with the transmitter, and log the last QSO that it was tackled very shortly after getting a microcomputer system in operation.

A review of literature in amateur publications quickly turned up a wealth of ideas for software. The article by Henson in February 1977 73 Magazine is an excellent reference and in fact forms the basis for most of the orbital calculations in this program.

The material available for hardware ,on the other hand, is not very helpful unless one intends to exactly duplicate a particular pre-programmed "tracking" circuit. Since the effort and cost involved in building one of these devices is a significant fraction of that necessary to assemble a small microcomputer and the result is specialized to only one purpose, it seems more desirable to put the labor and money into a "micro".

At this point it was necessary to make some choices relating to the specific hardware to be involved at the antenna end of the system. The data transfer necessary between the antenna-rotator system and the microcomputer is the current position and the movement commands. In order to simplify the interface hardware a type of azimuth and elevation rotator was chosen which controlled the motor by independent SPST switches (although one side of each switch is common) and which indicated position with an isolated potentiometer coupled to the rotator shaft. These are the Kenpro Model KR-400 and KR-500 for azimuth and elevation respectively. The essential part of the rotator schematic is shown in Figure A-1.(See the appendix for figures) Other types can of course be used with the interface board described but in some cases a little ingenuity may be necessary to obtain the variable DC voltage output from the indicator mechanism.

The signal, then, which is read to indicate azimuth (and elevation) is a DC voltage derived from the shaft coupled potentiometer. Control commands to the rotators are issued by simple

contact closures for cw,ccw,up,and down connected in parallel with the rotator's manual controls. This provides a convenient method for overriding the computer position commands.

A similar set of choices was necessary for the software. (The microcomputer was not open for choice, it was already in operation and not likely to be replaced!). The general specification for the program was that it require as input only equator crossing time and longitude for a desired pass and that it perform all other necessary calculations internally, commanding the rotators to point the antenna appropriately.

This left two loose ends, real time and orbital constants. It was decided to write separate programs for each satellite of interest, differing only in the fixed orbital constants.

Real time was a little stickier. It was finally decided to put a hardware clock in the computer because it would simplify applications in other amateur radio programs such as RTTY. Although the BASIC program presented references this hardware clock (Comptek Model CL-2400) other schemes of deriving real time can of course be used.

At this point it was necessary to decide on a hardware interface board to "read" the rotators and issue direction commands. The approach chosen called for a combination multiplexed A/D input and relay contact parallel output board. Since this exact combination does not exist among the many S-100 bus accessory boards, it was decided to design one to do the job efficiently rather than use up two chassis slots with separate boards or modify an "almost right" board.

Interface Board:
The interface board (6) was designed specifically for position readout from antenna rotators and for issuing direction commands to those rotators. Several features, however, give it wider application while serving as an antenna controller. The schematic appears in Figure A- 2.

A 3 1/2 digit BCD A/D converter with full scale reading of +/- 1.999 volts is provided. Input to the converter comes from an 8 position multiplexer under program control. Input to multiplexer channels 0 thru 7 appears on pins 1-6 and 19 of the 24

pin I/O socket with ground at pin 24. As an aid to troubleshooting, the multiplexer output is also brought to this socket, pin 23. Up to 8 separate analog inputs may be read but two of these will normally be antenna position, leaving 6 for signal strength, transmitter output, etc.

There are 6 parallel output relays which may be individually latched on or off under program control. The isolated SPST-NO relay contacts appear on pins 7-18 of the I/O socket. Normally, 4 of these will be used for rotator commands: up, down, cw and ccw. This leaves 2 free for transmit-receive switching, etc. The relay contacts are rated at 28 VDC, 250 Ma. maximum and external slave relays should be used if the rotator switch requirements exceed this.

In addition there are 2 direct outputs and 1 direct input to the 8255 PPI available. The outputs will only drive 1MA, however, and must be buffered for TTL compatibility. The input is directly TTL compatible. These lines appear on the I/O socket at pins 20 and 21 (output) and pin 22 (input).

The interface board uses an Intel 8255 PPI which is an extremely versatile parallel I/O device with 24 lines which can be programmed to be input or output. This board expects these lines to be programmed in a specific arrangement of 12 lines in and 12 lines out.

The 8255 responds to 4 sequential port addresses starting at the base address set into the interface board DIP switch. This switch sets the 6 most significant bits of the base address. The least 2 significant bits respond to address lines A0 and A1. Therefore to make the board respond to hex port addresses 10 through 13, the DIP switch would be set to 000100. For response at hex addresses 50 through 53 the DIP switch would be set to 010100, etc.

The addresses of the board will be referred to as A+0, A+1, A+2, and A+3 in this discussion where A is the offset determined by the 6 MSB's set into the DIP switch.

The arrangement of I/O lines in the 8255 is determined by a control word which is written once into address A+3. For the present board circuits, this control word must be 98 Hex (152 decimal). This control word sets up

address A+0 and the 4 MSB's of address A+2 as INPUT. It also sets up address A+1 and the 4 LSB's of address A+2 as OUTPUT. The meaning of the bits in these I/O words is given in Figure A-3.

In sending outputs to port addresses A+2 and A+1 which control the MUX address and state of the relays it should be remembered that all outputs are latching. The desired current state must be sent to ALL bit positions in the control word for EVERY output to the port. Potentiometers R9 and R10 allow full voltage from the rotator circuits to be set to 2.000 volts at the A/D converter input. This program assumes that 0 degrees elevation is represented by 0 volts and 90 degrees by 1.000 volts. Azimuth of -180 degrees is assumed to be 0 volts and +180 degrees to be 2.000 volts.

Appropriate modifications should be made to other rotator indicator circuits to obtain these voltages.

Software:

The BASIC source listing is given in Figure A-4. and the program variables are described in Figure A-5. The BASIC used is TDL 12K Super BASIC Version 3.0.

After initializing orbital constants, the program loads an assembly language routine at address 6E00H. This is the program which reads the MC14433 A/D converter on the interface board. It is programmed in assembly instead of BASIC solely for speed. Most BASICS would not be fast enough to read all BCD output digits on the same conversion cycle , leading to erratic results.

A source listing of the A/D program is given in Figure A-6. Note that an interface board base address of 20H is used and the A/D converter output is read on ports 20H and 22H. The program waits for an EOC bit to go true, then looks at the BCD position indicator bits and reads the BCD digit, storing it in the position register indicated. The nibble for overrange, MSD, and polarity is stored in D1 (at address 6E3F in this program) while the other 3 significant digit nibbles are stored at addresses D2, D3, and D4 from most to least significant. The BASIC program subsequently transfers these nibbles into the array variable DV for

conversion to a decimal voltage value. During this conversion the program checks D1 for overrange and sets the value output to 2.000 volts if overrange has occured.

The program then gives an opportunity to slew the antenna to any desired position for testing or whatever purpose is in mind. Next the hardware clock may be set by fast running if it is not already on time. The next option is for tracking the pass or else just printing the az-el pointing coordinates at intervals. The latter is useful for manual tracking before the rotator hardwre is built or connected. The next option is for pre-AOS and post-LOS tracking when the satellite is beyond the maximum angle of observation. It leaves the elevation at 0 (attenuation through the earth would be rather high) but tracks the proper azimuth for attempting over the horizon DX.

If the tracking mode has been selected and the satellite has not yet come over the local horizon, the program waits until it does. Then at intervals of real time set by variable IM the current azimuth and elevation are computed and the antenna is moved to those coordinates. Care has been taken to account for the possibilities of the track passing through the rotator stops at +/- 180 degrees. If the track reaches one of these limits, the program stops and slews the antenna so that the desired azimuth is reached from the complementary side of the stop and tracking continues from that point.

After each antenna update the current time, coordinates, range to the satellite and doppler shift are printed on the console. The doppler shift computation assumes the uplink and downlink frequencies of Oscar 7, mode B and is not accurate for ranges beyond the maximum observation angle.

Note that alignment errors in the antenna mounting can be easily corrected in the program (assuming that you can figure out the error). One such correction appears in the elevation routine of this program.

Two features of the BASIC used which may not correspond to other extended BASICS are the formatting strings for the PRINT USING statement and the proceedure for calling the assembly language routine. Users should check these areas especially when translating to other dialects of BASIC.

In order to apply this program to other satellites, the user must insert the appropriate values for P, H0, T9, and A0 in the initializing statements at the beginning. The station lattitude K0 and longitude K1 must be set to the proper values in any use of the program.

A typical console printout during tracking is shown in Figure A-7.

Conclusions:

The results of this effort have been generally rewarding. During a typical pass the system operates very well to alleviate attention to the antenna. The console display is a great aid in timing QSOS and looking for particular stations. In use with the NOAA satellites the system permits almost completely automatic picture acquisition.

Two negative features have turned up however, one of minor importance and the other a GREAT HAIRY MONSTER ! The simpler problem is that for some orientations of the antenna array the angular dynamics of the antenna-mast load interact with the control algorithm to produce a persistent oscillation of a few degrees beyond the dead band. This would be easily eliminated by a proportional control algorithm instead of "bang-bang" but the effect is so infrequent that it has been ignored.

The other situation is that awful spectre RFI! DX operation on OSCAR at low elevations is a weak signal afair and the last thing one wants is spurious signals. RFI generation by typical hobby microcomputers is intense within a hundred feet or so over the entire amateur spectrum. The author's system was no exception. A great deal of effort was put into shielding and "bead-choking" almost all lines exiting the computer case with only partial sucess. On the OSCAR-7 mode B downlink at 145.925 to 145.975 MHZ numerous birdies are still present and they are modulated by the various operations of the program, often obscuring desired stations. Additional shielding and RFI tracing is obviously needed.

This problem seems to be a serious inhibition to more widespread and sucessful application of microcomputers to amateur radio. It is hoped that this situation can be tackled by some of the numerous clever individuals in our hobby and some effective remedies found.

References:
1. B. Henson, WB0JHS, 73 Magazine, Feb. 1977, pg.72
2. A. Burke, W6UIX, 73 Magazine, Nov. 1977, pg. 58
3. T. Prewitt, W9IJ, 73 Magazine, Nov. 1977, pg. 64
4. D. Brown, W9GCI, 73 Magazine, July 1977, pg. 46
5. G.Bailey, WA3HLT, Ham Radio, Jan. 1975, pg.26
6. The interface board described is available in PC board or
   assembled and tested form from the author at: Dytron Inc.
   241 Crescent St. Waltham Ma. 02154.
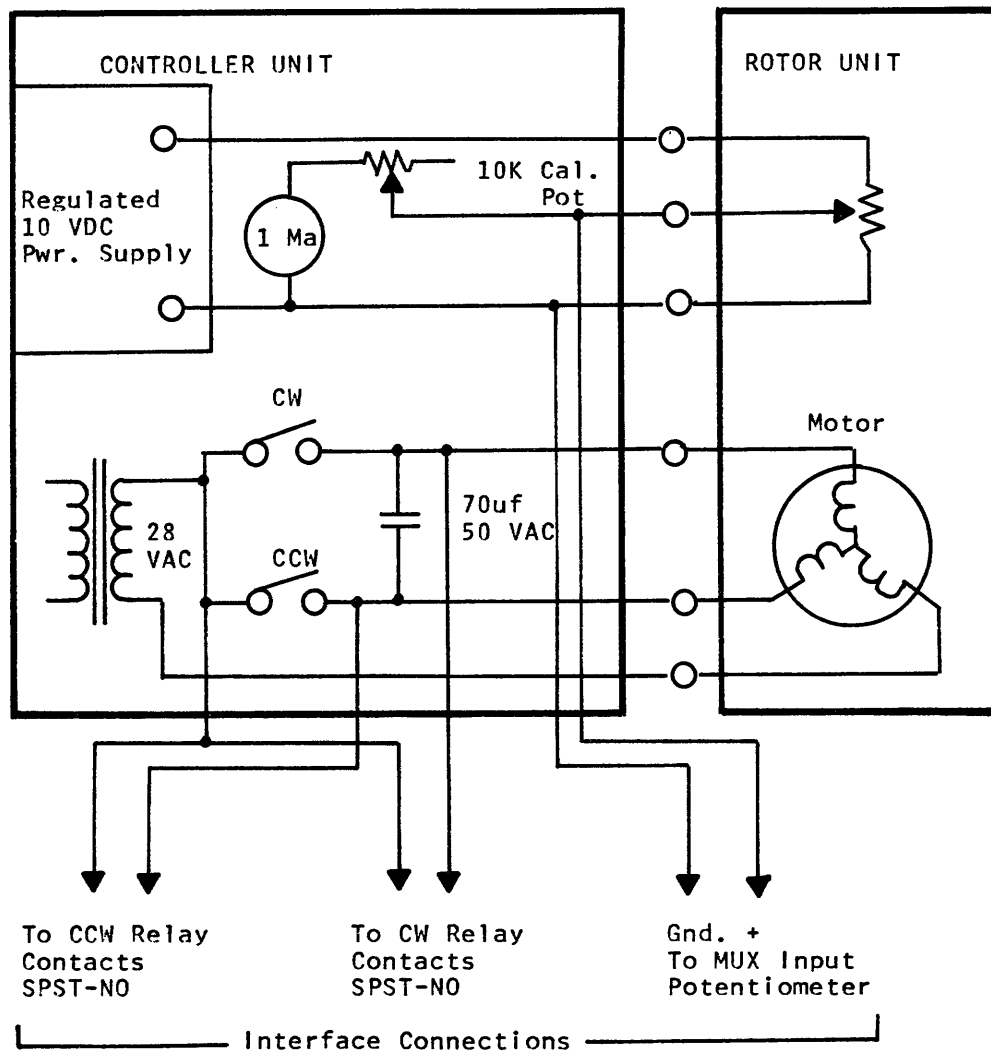
FIGURE A-1

Partial Rotator Schematic Showing Interface Wiring
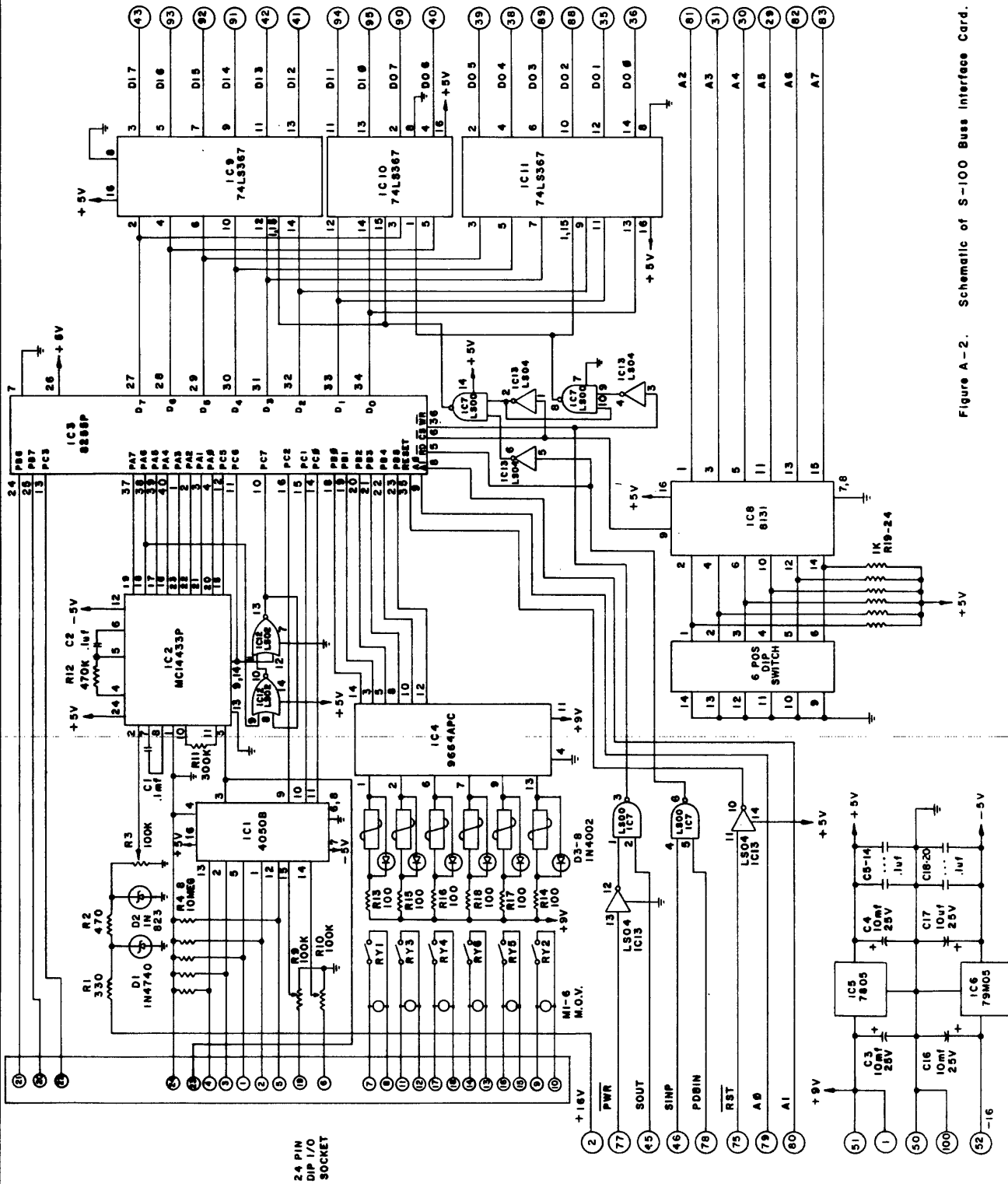
Figure A-2. Schematic of S-100 Buss Interface Card.

INPUT PORTS

ADDRESS   A+0   ADDRESS   A+2

7   0   7   0
┌─┬─┬─┬─┬─┬─┬─┬─┐   ┌─┬─┬─┬─┬─┬─┬─┬─┐
│ │ │ │ │ │ │ │ │   │ │▨│ │▨│ │▨│▨│▨│
└─┴─┴─┴─┴─┴─┴─┴─┘   └─┴─┴─┴─┴─┴─┴─┴─┘
DS1--DS4│Q3 -- Q0   ↑   ↑   ↑
MSD--LSD│   │   │   └─Direct Input(22)
       ↑   ↑   │   └─── Overrange
       │   └─BCD Data   └─────── EOC
       └─ Digit Strobes

OUTPUT PORTS

ADDRESS   A+1   ADDRESS   A+2

7   0   7   0
┌─┬─┬─┬─┬─┬─┬─┬─┐   ┌─┬─┬─┬─┬─┬─┬─┬─┐
│ │ │ │ │ │ │ │ │   │▨│▨│▨│▨│ │ │ │ │
└─┴─┴─┴─┴─┴─┴─┴─┘   └─┴─┴─┴─┴─┴─┴─┴─┘
21 │9 14   11   MSB-LSB
  20│ 16   17 7   ↑
   ↑   ↑   └─MUX Address
   │   └─Output Relays
   └──── Direct Output
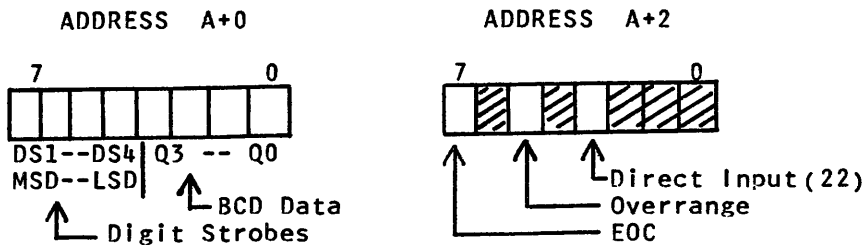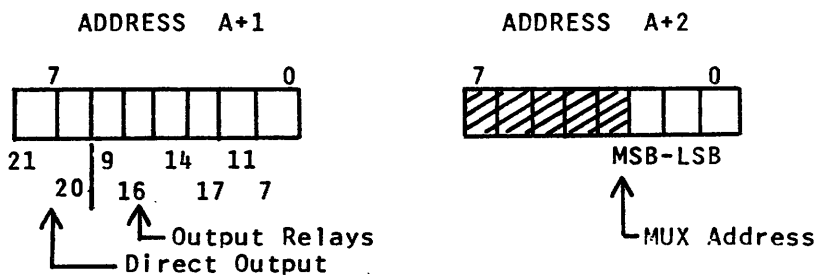
(Numbers are I/O socket pins)

FIGURE   A-3

Bit functions of I/O ports on interface card.

The following 7 pages are the BASIC listing
of the system operating program. They are
referred to in the text as Figure A-4.

```
10 REM BASIC ANTENNA TRACKING PROGRAM FOR POLAR ORBITING SATELLITES
20 REM
30 DIM DV(5), T(10)
40 REM
50 REM FORMATTING FOR PRINT USING STATEMENT
60 REM
70 !TIME=##:##:##    RANGE=#### MILES    DOPPLER=##.## KHZ
80 REM
90 REM STATION, ORBIT, AND MISCELLANEOUS CONSTANTS
100 REM
110 K0=42.5:K1=71.5:P2=6.283185:P9=P2/360.0:IL=60:IM=1:RE=3961
120 D1=&6E3F:PR=2:P=114.9448:H0=101.70:T9=28.73625:A0=910.0:RL=5000
130 REM
140 REM LOAD ASSEMBLY LANGUAGE A/D CONVERTER PROGRAM
150 REM
160 GOSUB 3470
170 REM
180 REM INITIALIZE 8255 PPI AND THE ROTATOR SWITCHES
190 REM
200 OUT 35,152
210 OUT 33,0
220 REM BEGIN PROGRAM ROUTINES
230 REM
240 INPUT "WANT TO SLEW ANT.";A$
250 IF A$="Y" THEN 3380
260 REM
270 REM CALCULATE MAXIMUM ANGLE OF OBSERVATION
280 X2=3957/(A0+3957)
290 GOSUB 2670   :REM CONVERT TO DEGREES
300 M0=X7
310 REM
320 INPUT"WANT TO SET CLOCK?";A$
330 IF A$="N" THEN 360
340 GOSUB 1320
350 REM
360 INPUT"PRINT RESULTS(P) OR TRACK(T)";A$
370 IF A$="P" THEN J1=1 ELSE J1=2
380 REM
390 REM
400 INPUT "EQX TIME";H,M,S
410 INPUT"EQX LONG.";L0
420 INPUT"WANT TO TRACK PRE-AOS/POST-LOS";A$
430 IF A$="Y" THEN J2=1 ELSE J2=2
440 REM
450 REM CONVERT EQX TIME TO OFFSET DECIMAL
460 REM
470 GOSUB 1730 : TX=TM
480 IF J1=1 THEN 530      :REM GO DIRECTLY TO COMPUTING IF NOT TRACKING
490 REM
500 REM WAIT FOR EQUATOR CROSSING
510 REM
520 GOSUB 1720 : IF TM<TX THEN 520
530 REM:
540 REM: MAIN PROGRAM LOOP
```

```
550 REM
560 PRINT
570 FOR I9=1 TO IL STEP IM
580 GOSUB 2490   :REM COMPUTE LAT. AND LONG. OF SUB-POINT
590 IF J1=1 THEN 640      :REM SKIP WAITING FOR HORIZON IF NOT TRACKING
600 REM
610 REM WAIT FOR LOCAL HORIZON CROSSING
620 REM
630 GOSUB 1720 : IF TM<(TX+I9/60-IM/120) THEN 630
640 GOSUB 3100 : IF D=0 THEN IF I9<IL THEN 1010 ELSE 1030 ELSE 680
650 REM
660 REM COMPUTE REQUIRED AZIMUTH AND ELEVATION FOR ANTENNA
670 REM
680 GOSUB 2830 : AZ=C
690 GOSUB 3310 : EL=E
700 REM
710 REM PRINT THE CURRENT AZ/EL AND POSITION THE ANTENNA
720 REM UNLESS WE ARE IN THE PRINT ONLY MODE (J1=1)
730 REM
740 ! ELEVATION = ##.#
750 ! AZIMUTH = ###.#
760 PRINT USING 750;AZ
770 REM
780 REM SKIP ANTENNA AZIMUTH IF NOT TRACKING
790 REM
800 IF J1=1 THEN 850
810 GOSUB 1780
820 REM
830 REM SKIP ANTENNA ELEVATION IF NOT TRACKING
840 REM
850 PRINT USING 740;EL
860 IF J1=1 THEN 910
870 GOSUB 2170
880 REM
890 REM COMPUTE RANGE AND DOPPLER
900 REM
910 GOSUB 2930
920 REM
930 REM IF NOT TRACKING, THEN ARTIFICALLY INCREMENT CLOCK
940 REM
950 IF J1=1 THEN TM=TX+I9/60
960 H=INT(TM-24):M=((TM-24)-H)*60:S=(M-INT(M))*60:IFH>=24THENH=H-24
970 REM
980 REM PRINT TIME, RANGE,AND DOPPLER AND GO AROUND LOOP AGAIN
990 REM
1000 PRINT USING 70;H,INT(M),INT(S),INT(R),FD
1010 PRINT : NEXT
1020 REM END OF THE PROGRAM, REQUEST NEXT ACTION.
1030 PRINT"LOS" : GOTO 360
1040 REM:
1050 REM ********** SUBROUTINES FOLLOW *************
1060 REM:
1070 REM: THIS SUBROUTINE CALLS AN ASSEMBLY LANGUAGE ROUTINE
1080 REM: TO READ THE A TO D CONVERTER. THE MUX ADDRESS AND
1090 REM: PPI MUST BE SET UP BEFORE ENTRY. THE VALUE RETURNS
```

```
1100 REM: IN D1,D2,D3,AND D4 AS BCD.
1110 REM
1120 CALL &6E00
1130 FOR I=0 TO 4 : DV(I)=PEEK(D1+I)AND15 : NEXT I
1140 IF (DV(0) AND 15)=(7 OR 3) THEN 1190
1150 DV(5)=- ((SGN(DV(0)AND8))-1)
1160 DV(4)=DV(5)+DV(1)/10+DV(2)/100+DV(3)/1000
1170 IF( DV(0) AND 4) THEN RETURN
1180 DV(4)=-DV(4) : RETURN
1190 IF( DV(0) AND 8) THEN DV(4)=0 : GOTO 1170
1200 DV(4)=2.0 : GOTO 1170
1210 RETURN
1220 REM:
1230 REM: SUBROUTINE TO FETCH THE CURRENT TIME.
1240 REM: TIME RETURNS IN VARIABLES H,M,S
1250 REM
1260 FOR I=0 TO 7 : LET T(I)=INP(168+I) : NEXT I
1270 H=10*T(7)+T(6)
1280 M=10*T(5)+T(1)
1290 S=10*T(2)+T(3)
1300 RETURN
1310 REM:
1320 REM: SUBROUTINE TO SET THE CLOCK
1330 REM
1340 PRINT"FLIP SS15 DOWN"
1350 PRINT"ENTER TIME + AT LEAST 2 MINUTES"
1360 PRINT"AS 4 DIGITS SEPARATED BY COMMAS."
1370 PRINT"FLIP SS15 UP AT EXACT TIME."
1380 OUT 169,0
1390 INPUT H9,H,M9,M
1400 OUT 169,4
1410 IF INP(175)=H9 THEN IF INP(174)=H THEN OUT 169,0:GOTO1430
1420 GOTO 1410
1430 OUT 169,2
1440 IF INP(173)=M9 THEN IF INP(169)=M THEN OUT 169,1:GOTO1460
1450 GOTO 1440
1460 WAIT 255,128
1470 OUT 169,0
1480 GOSUB 1230
1490 PRINT"TIME -- " H ":" M ":" S
1500 RETURN
1510 REM:
1520 REM: SUBROUTINE TO FETCH CURRENT AZIMUTH
1530 REM: VALUE RETURNS IN AT
1540 REM
1550 OUT 34,1 : WAIT 34,128 : GOSUB 1120
1560 IF DV(4)>=0 AND DV(4)<=1 THEN AT=180+DV(4)*180 : GOTO 1590
1570 IF DV(4)>1 AND DV(4)<=2 THEN AT=(DV(4)-1)*180 : GOTO 1590
1580 PRINT CHR$(7);CHR$(7);"AZIMUTH LIMIT ERROR": GOTO 1590
1590 RETURN
1600 REM:
1610 REM: SUBROUTINE TO FETCH CURRENT ELEVATION
1620 REM: VALUE RETURNS IN VARIABLE ET
1630 REM
1640 OUT 34,2 : WAIT 34,128 : GOSUB 1120
```

```
1650 IF DV(4)>=0 AND DV(4)<=2 THEN ET=DV(4)*90 : GOTO 1670
1660 PRINT CHR$(7);CHR$(7);"ELEVATION LIMIT ERROR": GOTO 1670
1670 RETURN
1680 REM:
1690 REM: SUBROUTINE TO FETCH TIME IN DECIMAL WITH 24HR OFFSET
1700 REM: TIME RETURNS IN TM IN HOURS + 24
1710 REM:
1720 GOSUB 1260
1730 TM=H+M/60+S/3600+24:RETURN
1740 REM:
1750 REM:
1760 REM: SUBROUTINE TO MOVE TO A REQUESTED VALUE OF AZIMUTH ,AZ
1770 REM:
1780 GOSUB 1550
1790 DF=AZ-AT
1800 IF ABS(DF)<PR THEN OUT 33,0 : RETURN
1810 IF ABS(DF)<3*PR THEN OUT 33,0
1820 IF DF<0 THEN 1980
1830 IF ABS(DF)<180 THEN 1920
1840 REM:
1850 REM: (AZ-AT)>0 AND ABS(AZ-AT)>180
1860 REM:
1870 OUT 33,1
1880 GOTO 1780
1890 REM:
1900 REM: (AZ-AT)>0 AND ABS(AZ-AT)<180
1910 REM:
1920 IF(AZ >180)AND(AT<=180) THEN J=-1 : GOTO 2100
1930 OUT 33,2
1940 GOTO 1780
1950 REM:
1960 REM: (AZ-AT)<0 AND ABS(AZ-AT)>180
1970 REM:
1980 IF ABS(DF)<180 THEN 2040
1990 OUT 33,2
2000 GOTO 1780
2010 REM:
2020 REM: (AZ-AT)<0 AND ABS(AZ-AT)<180
2030 REM:
2040 IF (AT>=180) AND (AZ<180) THEN J=1 : GOTO 2100
2050 OUT 33,1
2060 GOTO 1780
2070 REM:
2080 REM: SLEW 180 DEGREES
2090 REM:
2100 A1=AZ
2110 AZ=AT+J*180
2120 IF AZ<0 THEN AZ=AZ+360
2130 IF AZ>360 THEN AZ=AZ-360
2140 GOSUB 1760
2150 AZ=A1
2160 GOTO 1780
2170 REM:
2180 REM: SUBROUTINE TO MOVE TO A REQUESTED VALUE OF ELEVATION, EL
2190 REM:
```

```
2200 IF EL<0 OR EL>90 THEN EL=0
2210 REM: NEXT STATEMENT IS PURELY FUDGE,CORRECTS ALIGNMENT ERROR
2220 EL=EL+5
2230 REM:
2240 GOSUB 1640
2250 DF=EL-ET
2260 IF ABS(DF)<PR THEN OUT 33,0: RETURN
2270 IF DF<0 THEN 2300
2280 OUT 33,8
2290 GOTO 2240
2300 OUT 33,4
2310 GOTO 2240
2320 REM:
2330 REM: SUBROUTINE TO CALCULATE ARCSIN
2340 REM: ENTER ARGUMENT IN X1, VALUE RETURNS IN X8
2350 REM:
2360 IF X1>=1 THEN 2450
2370 IF X1<=-1 THEN 2460
2380 H9=90.0
2390 H1=-90.0
2400 X8=(H1+H9)/2
2410 IF ABS(SIN(X8*P9)-X1)<0.0001 THEN RETURN
2420 IF SIN(X8*P9)>X1 THEN 2440
2430 H1=X8 : GOTO 2400
2440 H9=X8 : GOTO 2400
2450 X8=90. : RETURN
2460 X8=-90. : RETURN
2470 REM:
2480 REM: SUBROUTINE TO CALCULATE LAT. AND LONG. OF SUB POINT
2490 REM:
2500 REM: REQUIRES TIME I9 AND EQX LONG L0
2510 REM: VALUES RETURN IN S0 AND S1 IN DEGREES
2520 REM:
2530 X1=.9790*SIN(5.4662E-2*I9)
2540 GOSUB 2360
2550 S0=X8
2560 X2=COS(5.4662E-2*I9)/COS(P9*S0)
2570 GOSUB 2670
2580 S1=X7
2590 S1=S1+I9/4+L0
2600 IF S1<360 THEN RETURN
2610 S1=S1-360 : RETURN
2620 REM:
2630 REM:
2640 REM:  SUBROUTINE TO CALCULATE ARCCOS
2650 REM: ENTER ARGUMENT IN X2, VALUE RETURNS IN X7
2660 REM:
2670 IF X2>=1.0 THEN 2760
2680 IF X2<=-1.0 THEN 2770
2690 H9=0.0
2700 H1=180.0
2710 X7=(H1+H9)/2
2720 IF ABS(COS(X7*P9)-X2)<0.0001 THEN RETURN
2730 IF COS(X7*P9)>X2 THEN 2750
2740 H1=X7 : GOTO 2710
```

```
2750 H9=X7 : GOTO 2710
2760 X7=0. : RETURN
2770 X7=180. : RETURN
2780 REM:
2790 REM: SUBROUTINE TO FIND AZIMUTH TO SUB POINT
2800 REM: REQUIRES PARAMETERS S0,P9,K0,D,L5
2810 REM: VALUE RETURNS IN C
2820 REM:
2830 X2=SIN(S0*P9)-SIN(K0*P9)*COS(D*P9)
2840 X2=X2/(COS(K0*P9)*SIN(D*P9))
2850 GOSUB 2670
2860 C=X7
2870 IF L5>=0 THEN RETURN
2880 C=360-X7 : RETURN
2890 REM:
2900 REM: SUBROUTINE TO CALCULATE RANGE AND DOPPLER
2910 REM: REQUIRES PARAMETERS: A0,D,P9,E,S0,RE,H0
2920 REM:
2930 R=((A0+3957)*COS(D*P9)-3957)/COS((90.-E)*P9)
2940 R=ABS(R)
2950 X2=(R~2+(RE+A0)~2-RE~2)/(2*R*(RE+A0))
2960 GOSUB 2670
2970 VP=4.43*SIN(X7*P9)
2980 VR=((VP*SIN(H0*P9))~2)*(VP*COS(H0*P9)-.285*COS(S0*P9))~2
2990 FD=3.108*SQR(VR)
3000 IF R>RL THEN FD=-FD
3010 RL=R
3020 RETURN
3030 REM:
3040 REM:
3050 REM: SUBROUTINE TO FIND G.C. DEGREES BETWEEN STATION AND
3060 REM: SATELLITE SUB POINT. REQUIRES PARAMETERS K0,P9,S0,S1,K1
3070 REM: VALUE RETURNS IN D
3080 REM:
3090 REM:
3100 IF S1<=180 THEN 3120
3110 S1=S1-360
3120 L5=K1-S1
3130 IF ABS(L5)<=180 THEN 3170
3140 IF(K1-S1)<0 THEN 3160
3150 L5=L5-360 : GOTO 3170
3160 L5=L5+360
3170 X2=SIN(K0*P9)*SIN(S0*P9)
3180 X2=X2+COS(K0*P9)*COS(S0*P9)*COS(L5*P9)
3190 GOSUB 2670
3200 D=X7
3210 IF J2=1 THEN 3240
3220 IF D<= M0 THEN RETURN
3230 D=0 : REM: OUT OF RANGE
3240 RETURN
3250 REM:
3260 REM
3270 REM: SUBROUTINE TO CALCULATE ELEVATION
3280 REM: REQUIRES PARAMETERS :A0,D,P9
3290 REM: VALUE RETURNS IN E
```

```
3300 REM:
3310 E=(A0+3957)*SIN(D*P9)
3320 E=E/((A0+3957)*COS(D*P9)-3957)
3330 E=ATN(E)
3340 E=90.-360*E/P2
3350 IF E>160 THEN E=E-180
3360 RETURN
3370 REM:
3380 REM:
3390 REM: UTILITY SLEW ROUTINE
3400 REM:
3410 INPUT "AZ=";AZ
3420 INPUT "EL=";EL
3430 GOSUB 1780
3440 GOSUB 2240
3450 GOTO 240
3460 REM:
3470 REM:
3480 REM: THIS SUBROUTINE LOADS THE A/D ROUTINE AT 6E00 H
3490 REM:
3500 FOR J=0 TO 62
3510 READ Z
3520 POKE ( &6E00+J ) ,Z : NEXT
3530 RETURN
3540 REM:
3550 DATA&DB,&22,&E6,&80,&CA,&00,&6E,&DB,&20,&E6,&80,&CA
3560 DATA&07,&6E,&DB,&20,&47,&DB,&20,&E6,&40,&CA,&11,&6E
3570 DATA&DB,&40,&4F,&DB,&40,&E6,&20,&CA,&1B,&6E,&DB,&20,&57
3580 DATA&DB,&20,&E6,&10,&CA,&25,&6E,&DB,&20,&32,&42,&6E
3590 DATA&7A,&32,&41,&6E,&79,&32,&40,&6E,&78,&32,&3F,&6E,&F9,&C9
```

```
AO:   Orbit altitude in miles.
AT:   Observed current azimuth in degrees.
AZ:   Requested azimuth in degrees.
C :   Azimuth of satellite sub-point in degrees.
D :   Angular separation of station and sub-point in degrees.
DF:   Current error from setpoint in azimuth or elevation.
DV:   Array containing D1-4 nibbles from A/D converter.
D1:   Most significant BCD nibble of A/D conversion.
D2:   Second most significant A/D nibble.
D3:   Third most significant A/D nibble.
D4:   Least significant A/D nibble.
E :   Elevation of satellite in degrees.
EL:   Requested elevation of antenna in degrees.
ET:   Observed current elevation of antenna in degrees.
FD:   Doppler shift in HZ.
H :   Time in hours.
HO:   Inclination of orbit in degrees.
H9:   Miscelaneous temporary variable.
IL:   Limit for variable I9.
IM:   Increment of I9 in minutes.
I9:   Main program loop index in minutes.
J1:   Print or track flag. 1=Print. 2=Track.
J2:   Pre-AOS/Post-LOS track flag. 1=Yes. 2=No.
KO:   N. lattitude of station in degrees.
K1:   W. longitude of station in degrees.
LO:   Equator crossing longitude in degrees.
L5:   Difference in longitude between station and satellite sub-point.
M :   Time in minutes.
MO:   Maximum observation angle at station in degrees.
M9:   Miscelaneous temporary variable.
PR:   Desired dead band around az/el set point.
PO:   Period of orbit in minutes.
P2:   2*PI.
P9:   Radians/degree
R :   Range to satellite in miles.
RE:   Radius of earth in miles.
RL:   Minimum range to satellite in miles.
S :   Time in seconds.
SO:   Lattitude of sub-point in degrees
S1:   Longitude of sub-point in degrees.
T :   Array variable containing BCD elements of time from
      hardware clock.
T9:   Degrees of westerly progression of orbit per orbit.
TM:   Current time in decimal hours+24.
TX:   Equator crossing time in decimal hours+24.
VP:   Miscellaneous temporary variable.
VR:   Miscellaneous temporary variable.
X1:   Argument to arcsin routine.
X2:   Argument to arccos routine.
X7:   Value returned by arccos routine.
X8:   Value returned by arcsin routine.
```

Figure A-5. Definition of Program Variables

```
                              .PABS
                    ;
                    ;
                    ;THIS ROUTINE READS MC14433P A/D CONVERTER AND STORES
                    ;BCD DIGITS FOR READING BY CALLING PROGRAM. PRIOR TO
                    ;USE 8255 PPI MUST BE SET TO CONTROL WORD 98H AND THE
                    ;MUX ADDRESS MUST BE OUTPUT TO SELECT DESIRED CHANNEL.
                    ;
                    ;
6E00                          .LOC 6E00H
                    ;
6E00    DB22        EOC:    IN      22H     ;CHECK THE EOC BIT
6E02    E680                ANI     80H     ;AND LOOP UNTIL TRUE
6E04    CA 6E00             JZ      EOC
6E07    DB20        DS1:    IN      20H     ;LOOK AT THE DIGIT STROBES
6E09    E680                ANI     80H     ;AND LOOP UNTIL D1 GOES TRUE
6E0B    CA 6E07             JZ      DS1
6E0E    DB20                IN      20H     ;THEN READ THE MSD
6E10    47                  MOV     B,A     ;AND STORE IN B REGISTER
6E11    DB20        DS2:    IN      20H     ;LOOK AT STROBES AGAIN
6E13    E640                ANI     40H     ;AND LOOP UNTIL D2 GOES TRUE
6E15    CA 6E11             JZ      DS2
6E18    DB20                IN      20H     ;THEN READ 2ND MSD
6E1A    4F                  MOV     C,A     ;AND STORE IN C REGISTER
6E1B    DB20        DS3:    IN      20H     ;LOOK AT DIGIT STROBES AGAIN
6E1D    E620                ANI     20H     ;AND LOOP UNTIL D3 GOES TRUE
6E1F    CA 6E1B             JZ      DS3
6E22    DB20                IN      20H     ;THEN READ 3RD MSD
6E24    57                  MOV     D,A     ;AND STORE IN D REGISTER
6E25    DB20        DS4:    IN      20H     ;LAST LOOK AT DIGIT STROBES
6E27    E610                ANI     10H     ;AND LOOP UNTIL D4 GOES TRUE
6E29    CA 6E25             JZ      DS4
6E2C    DB20                IN      20H     ;THEN READ LSD
6E2E    32 6E42             STA     D4      ;STORE LSD IN LOCATION D4
6E31    7A                  MOV     A,D     ;GET BACK 3RD MSD
6E32    32 6E41             STA     D3      ;AND STORE IN LOCATION D3
6E35    79                  MOV     A,C     ;GET BACK 2ND MSD
6E36    32 6E40             STA     D2      ;AND STORE IN LOCATION D2
6E39    78                  MOV     A,B     ;GET BACK MSD
6E3A    32 6E3F             STA     D1      ;AND STORE IN LOCATION D1
                    ;
6E3D    F9                  SPHL            ;RESTORE CALLING STACK
6E3E    C9                  RET             ;AND RETURN
                    ;
                    ;
6E3F                        D1=.    ;USE THESE LOCATIONS FOR DIGIT STORAGE
6E40                        D2=.+1
6E41                        D3=.+2
6E42                        D4=.+3
                    ;
                    ;
                            .END
```

Figure A-6.   Assembly listing for A/D converter "reader".

```
AZ= 154.6
EL= 21.2395
TIME= 0 : 7 : 59        RANGE= 1742 MI.      DOPPLER= 9.60851 KHZ.


AZ= 153.984
EL= 27.5999
TIME= 0 : 8 : 59        RANGE= 1541 MI.      DOPPLER= 8.74297 KHZ.


AZ= 152.798
EL= 35.2726
TIME= 0 : 9 : 59        RANGE= 1355 MI.      DOPPLER= 7.53359 KHZ.


AZ= 151.414
EL= 44.9193
TIME= 0 : 10 : 59       RANGE= 1185 MI.      DOPPLER= 5.85798 KHZ.


AZ= 146.843
EL= 56.4242
TIME= 0 : 12 : 0        RANGE= 1050 MI.      DOPPLER= 3.83427 KHZ.


AZ= 138.691
EL= 70.8016
TIME= 0 : 13 : 0        RANGE= 952 MI.       DOPPLER= 1.65726 KHZ.


AZ= 81.0242
EL= 82.5105
TIME= 0 : 14 : 0        RANGE= 916 MI.       DOPPLER= .433502 KHZ.


AZ= 0
EL= 73.4254
TIME= 0 : 15 : 0        RANGE= 941 MI.       DOPPLER=-1.27285 KHZ.


AZ= 353.232
EL= 59.0203
TIME= 0 : 15 : 59       RANGE= 1027 MI.      DOPPLER=-3.18307 KHZ.


AZ= 347.959
EL= 46.7227
TIME= 0 : 16 : 59       RANGE= 1160 MI.      DOPPLER=-5.09307 KHZ.


AZ= 346.113
EL= 36.8374
TIME= 0 : 17 : 59       RANGE= 1323 MI.      DOPPLER=-6.57922 KHZ.


AZ= 345.278
EL= 28.8728
TIME= 0 : 19 : 0        RANGE= 1506 MI.      DOPPLER=-7.60845 KHZ.


AZ= 344.707
EL= 22.3385
TIME= 0 : 20 : 0        RANGE= 1704 MI.      DOPPLER=-8.20 KHZ.
```

Figure   A-7

Typical Console Output

MICROPROCESSOR STANDARDS — THE SOFTWARE ISSUES

by
Tom Pittman
P.O.Box 23189
San Jose, Ca. 95153

Abstract

The IEEE Computer Society Microprocesor Standards Committee has been working on six specific goals, including bus standards and three software areas: Floating Point, Relocatable Code, and Assembly Language. We have learned not only about the areas under discussion, but also some of the political and social ramifications of standards. This paper reports on these as well as the progress in the software areas.

Introduction

The IEEE Computer Society Microprocessor Standards Committee first met in August 1977. I was invited to represent the hobbyist community, or more specifically the HomeBrew Computer Club, on the committee. At that first meeting the participants selected five areas that we thought deserved immediate attention and seemed achievable. Three of these had to do with bus standards, and are reported elsewhere. The other two were Relocatable Code and Assembly Language differences. Since that time we also took on the task of developing a Floating Point standard. I have been intimately involved in the work with these last three areas, and this paper deals partly with their progress.

In journalism considerable emphasis is placed on "the five W's" Who, What, When, Where, Why, and how. These (slightly re-ordered) will form the outline of this report. The what has already been mentioned: it is the three software areas which we have been considering.

Why

Why standardize in these three areas? The reasons are both unique to the areas considered, and common among them. The common reasons are to minimize the dislocation in moving from one microprocessor to another, or from one manufacturer's products for a single CPU to another's products for the same CPU. In each one of these areas there is no uniformity across a half dozen or more products. Programs developed on one system will not run unmodified on another.

Prior to the advent of the microprocessor, there was generally only one manufacturer for a given CPU. He made the decisions, and everyone went along. But with the micros, the chip manufacturer was often not the first out with a usable, standard-setting format. And there are dozens of systems builders using the same CPU, but designing their own software. Some of these are good, but expensive; others are in the public domain, but are badly flawed; still others are both economical and adequate, but come out so late that they cannot influence the whole market. Most of these are self-proclaimed "standards" of one kind or another. What we need are some standards with authority, and we bring the national authority of IEEE Computer Society to this work.

Relocatable Code. This particular area is of some personal concern to me, because I sell software to end-users. I find I am unable to sell a single product into every system of a single microprocessor, because every manufacturer puts the memory in a different place. I am obliged to support a variety of placements for the same program, and some potential customers simply cannot use it because it is uneconomical for me to support their perticular memory configurations. Result: higher costs for me, and higher prices to the customers (or it is simply unavailable). If I could sell it in relocatable form, one version would be sufficient.

So far I have had opportunity to examine a dozen or so relocatable code formats now on the market. I have yet to see one which will permit me to write the kinds of programs I sell. I have yet to see one that will work equally well for any of the several microprocessors I support with software. I hope I can influence the committee to adopt a standard that meets these requirements.

One of the major cost factors in the use of microprocessors is the software. If the adoption of a good relocatable code standard could reduce the costs of software houses, not only would their software become more widely available, but more companies would find the software business profitable, in spite of the increased competition and reduced prices.

Assembly Language. As microprocessors become more specialized, cheap, and numerous, more and more people will be obliged to write programs for several different chips. I do this, and one of the major sources of first pass assembly errors is the use of "BZ" in 6800 programs, "BEQ" in 8080 programs, "JZ" in 6502 programs, etc. It is precisely the same instruction in each processor I am dealing with, but each assembler spells it differently. Even worse, when I write an 8080 program for one assembler, it requires extensive editing before another will accept it. Companies whose product lines cross CPU lines, educators who must be up on all new processors, and consultants like myself want and need some uniformity.

Note that we are not trying to define a standard instruction set. Rather, we would like to say that if a microprocessor has an instruction that does thus, then this is the standard name for it. If it has this particular addressing mode, this is how to code instructions to use it.

Floating Point. Last time I looked, nearly every different BASIC on the market had a different way of doing arithmetic. Some of these had subtle flaws which could destroy computational precision. Many hobbyists would not notice the difference, but often these routines are used in scientific calculations outside the strictly hobbyist community. I think I would prefer not to drive over a bridge whose stress analysis was computed in Sphere BASIC [1]. With standards in this area, we can expect the same algorithm to produce the same answers in any processor, and moreover, we can expect the answers to be correct.

Who

Who is in favor of standards? Who opposes them? Who is doing the work? This is a touchy question.

Almost without exception, the opponents to standards are the designers, both individual and corporate. The individual designers are perhaps afraid that standards will inhibit their creativity. I suppose a poorly conceived standard might do that, and if the best designers refuse to participate in the standardization, they may well bring about the object of their fears. On the other hand, as Carl Helmers pointed out so clearly, standards are the basis from which you deviate. [2] Their main purpose is to reduce unnecessary variation, not to inhibit progress.

By "corporate designers" I mean those companies whose position of leadership in their field gives them the selection of market direction. IBM still commands the vast majority of large computer business, and IBM's participation in standards efforts is notoriously grudging. [3] The larger semiconductor houses are similarly staying away from the IEEE work in droves. I doubt that anyone would actually say this, but I think they firmly believe, "We don't abide by standards, we make them." In our industry this is a little more dangerous than it is for IBM: EBCDIC is carried on over ASCII by the sheer weight of IBM's market share, but none of the semiconductor houses have that big a market share. So they are gambling that our committee will not actually produce any widespread standards. I think they are wrong, and they may find themselves where UNIVAC was with round-hole tab cards.

Who supports the work? The most visible seems to be OEMs who use microprocessors from various manufacturers, or who build compatible peripherals. These companies have suffered the most from lack of standards. In the software areas, however, it is hard to identify these — perhaps there are FEW.

I suspect the greatest number of supporters are the end users, who are only now discovering that the lack of standards (both hardware and software) is costing them time and money. Unfortunately the users have had no clout in these matters. More about this later.

We are also getting a great deal of support from the smaller semiconductor houses. Perhaps they see the handwriting on the wall: "Participate or be left out."

When.

When will we begin to see some results? Will not any meaningful standards take so long to formulate that they become irrelevant? This is a good question. Standards committees often take years in their deliberations. The microprocessor fields are moving so fast that what is agreeable today may be nonsense in two years.

We recognized very early the problem of obsolescence. Instead of quarterly meetings, as I understand is usual for standards committees, we meet monthly. Even so, for those of us who are accustomed to turning a product around in a few weeks or months, the progress has been excruciatingly slow. But it is progress.

The other defense against obsolescence is careful planing with the future in mind. None of us are omniscient, and it is certainly more difficult when the movers and shakers of the industry decline to participate, but I think we can succeed here.

How.

How can you, the user, help make our standards work succeed? Talk it up. Make sure the manufacturers and retailers where you spend your money know that you will not buy nonconforming products. We need grass-roots suport. We need sources of material that may affect our deliberations; perhaps you can be part of a supply-line.

And we need no competition. I realize that sounds awful. Five standards are no standards. Encourage those who want to call what they are doing "standard" to work with us, not against us. When we started there was no national body effectively working with microprocessor standards. As far as I know we are still alone in this, though there have been other attempts. If we can refrain from diluting the national interest in standards, we have a better chance of making what the IEEE is doing stick.

Conclusion.

We will have standards. If we work together, they will be good ones, and they will come sooner. If we drag our collective heels, they may not be so good, and they will certainly take longer coming.

References.

[1]   Rich Didday "A Tale of Four BASICs" KILOBAUD Jan 78 p54.
[2]   Carl T Helmers, Jr, in an unpublished remark at the S-100 Symposium at Diablo Valley College Nov 20, 1976.
[3]   An example of this is quoted in DATAMATION Dec 77, p224.

PROPOSED IEEE STANDARD
FOR THE
S-100 BUS

Preliminary Specification

by

George Morrow
Thinker Toys
1201 - 10th Street
Berkeley, CA 94707
(415) 527-7548

and

Howard Fullmer
Parasitic Engineering Inc.
P. O. Box 6314
Albany, CA 94706
(415) 547-6612

The follwoing is a preliminary specification for the computer bus commonly known as the S-100 Bus. This bus was first introduced by MITS Inc. with their Altair kit. It has since spread throughout the electronics industry and beyond. Today over a hundred manufacturers make products which claim to be compatible with the S-100 bus even though, until now, no complete specification has been available.

This document is a specification for both timing and signal discipline. Signal discipline is described using the Bus Master/Bus Slave language long associated with Digital Equipment's PDP 11. This point of view facilitated the development of a simple and highly reliable DMA protocol which is perhaps the most important aspect of the standard.

Acknowledgement

S-100 BUS SIGNAL DEFINITIONS:

Pin No.	Signal Name & Type	Polarity	
1	+8 volts (B)[1]		Instantaneous minimum greater than 7 volts, instantaneous maximum less than 35 volts, average maximum less than 11 volts.
2	+16 volts (B)		Instantaneous minimum greater than 14 volts, instantaneous maximum less than 35 volts, average maximum less than 20 volts.
3	XRDY (S)[1,10]	positive	One of two ready inputs to the current Bus Master. The bus is ready when both these ready inputs are true.
4	VI_0 (S)[10]	"	Vectored interrupt line Ø.
5	VI_1 (S)[10]	"	" 1.
6	VI_2 (S)[10]	"	" 2.
7	VI_3 (S)[10]	"	" 3.
8	VI_4 (S)[10]	"	" 4.
9	VI_5 (S)[10]	"	" 5.
10	VI_6 (S)[10]	"	" 6.
11	VI_7 (S)[10]	"	" 7.
12	-	-	Not specified.
13	-	-	"
14	-	-	"
15	-	-	"
16	-	-	"
17	-	-	"

PRELIMINARY SUBJECT TO REVISION

S-100 Bus Signal Definitions

18	$\overline{\text{STAT DSB}}$ (M)[1,10]	negative	The control signal to disable the 8 status signals[2].
19	$\overline{\text{C/C DSB}}$ (M)[10]	"	The control signal to disable the 6 command/control signals[3].
20	UNPROT	-	Not specified.
21	SS	-	Not specified.
22	$\overline{\text{ADD DSB}}$ (M)[10]	negative	The control signal to disable the 16 address signals[4].
23	$\overline{\text{DO DSB}}$ (M)[10]	"	The control signal to disable the 8 data output[5] signals.
24	ϕ_2 (B)	positive	The master timing signal for the bus.
25	ϕ_1	-	Not specified.
26	PHLDA (M)	positive	A command/control signal used in conjunction with PHOLD to coordinate Bus Master transfer operations.
27	PWAIT (M)	"	The acknowledge signal to either of the bus ready signals XRDY, PRDY or to a HLT instruction.
28	PINTE	"	Not specified.
29	A5 (M)	"	Address bit 5.
30	A4 ( )	"	Address bit 4.
31	A3 (M)	"	Address bit 3.
32	A15 (M)	"	Address bit 15 (most significant).
33	A12 (M)	"	Address bit 12.
34	A9 (M)	"	Address bit 9.
35	DO1 (M)	"	Data out bit 1.
36	DOØ (M)	"	Data out bit Ø (least significant).
37	A10 (M)	"	Address bit 10.

S-100 Bus Signal Definitions

38	DO4 (M)	positive	Data out bit 4.
39	DO5 (M)	"	" 5.
40	DO6 (M)	"	" 6.
41	D12 (M)	"	Data in[6] bit 2.
42	D13 (M)	"	Data in bit 3.
43	D17 (M)	"	Data in bit 7 (most significant).
44	SM1 (M)	"	The status signal which indicates that the current bus cycle[7] is an op-code fetch.
45	SOUT (M)	"	The status signal identifying the data transfer bus cycle of an OUT instruction.
46	SINP (M)	"	The status signal identifying the data transfer bus cycle of an IN instruction.
47	SMEMR (M)	"	The status signal identifying bus cycles which transfer data from memory to a Bus Master which are not interrupt acknowledge instruction fetch cycle(s).
48	SHLTA (M)	"	The status signals which acknowledges that a HLT instruction has been executed.
49	$\overline{\text{CLOCK}}$	-	Not specified.
50	GND		Signal and pwer ground.
51	+8 volts (B)		See comments above for pin #1.
52	-16 volts (B)		Instantaneous maximum less than -14 volts, instantaneous maximum greater than -35 volts, average minimum greater than -20 volts.
53	$\overline{\text{SSWI}}$	-	Not specified.
54	$\overline{\text{EXT CLR}}$	negative	A reset signal to reset Bus Slaves. When this signal goes low, it must stay low for at least 3 bus states.
55	-	-	Not specified.
56	-	-	"
57	-	-	"

S-100 Bus Signal Definitions

58	-	-	Not specified.
59	-	-	Not specified.
60	-	-	Not specified.
61	-	-	Not specified.
62	-	-	Not specified.
63	-	-	Not specified.
64	-	-	Not specified.
65	-	-	Not specified.
66	-	-	Not specified.
67	$\overline{\text{PHANTOM}}$	-	Not specified.
68	MWRITE (B)	positive	The logical negation of $\overline{\text{PWR}}$ and $\overline{\text{SOUT}}$; this signal must follow $\overline{\text{PWR}}$ by not more than 30 ns.
69	$\overline{\text{PS}}$	-	Not specified.
70	PROT	-	Not specified.
71	RUN	-	Not specified.
72	PRDY (S) [10]	positive	See comments above for pin #3.
73	$\overline{\text{PINT}}$ (S) [10]	negative	The primary interrupt request bus signal.
74	$\overline{\text{PHOLD}}$ (M) [10]	''	The command/control signal used in conjunction with PHLDA to coordinate Bus Master transfer operations.
75	$\overline{\text{PRESET}}$ (B) [10]	''	The reset signal to reset bus master devices. When this signal goes low, it must stay low for at least 3 bus states.
76	PSYNC (M)	positive	The command/control signal identifying BS_1. (See bus states comments.)
77	$\overline{\text{PWR}}$ (M)	negative	The command/control signal signifying the presence of valid data on the DO bus[8].
78	PDBIN (M)	positive	The command/control signal that requests data on the DI bus[9] from the currently addressed slave.

PRELIMINARY SUBJECT TO REVISION

S-100 Bus Signal Definitions

79	A0 (M)	positive	Address bit 0 (least significant).
80	A1 (M)	"	Address bit 1.
81	A2 (M)	"	" 2.
82	A6 (M)	"	" 6.
83	A7 (M)	"	" 7.
84	A8 (M)	"	" 8.
85	A13 (M)	"	" 13.
86	A14 (M)	"	" 14.
87	A11 (M)	"	" 11.
88	DO2 (M)	"	Data out bit 2.
89	DO3 (M)	"	" 3.
90	DO7 (M)	"	Data out bit 7 (most significant).
91	DI4 (S)	"	Data in bit 4.
92	DI5 (S)	"	" 5.
93	DI6 (S)	"	" 6.
94	DI1 (S)	"	" 1.
95	DI0 (S)	"	Data in bit 0 (least significant).
96	SINTA (M)	"	The status signal identifying the instruction fetch cycle(s) that immediately follow an accepted interrupt request presented on $\overline{\text{PINT}}$.
97	$\overline{\text{SWO}}$ (M)	negative	The status signal identifying a bus cycle which transfers data from a Bus Master to a slave.
98	SSTACK	–	Not specified.
99	$\overline{\text{POC}}$ (B)	negative	The power-on clear signal for all bus devices; when this signal goes low, it must stay low for at least 3 bus states.
100	GND		Signal and power ground.

PRELIMINARY SUBJECT TO REVISION

S-100 Bus Signal Definitions

Bus Signal Notes

[1]There are three types of signals on the S-100 bus. M stands for Bus Master. Signals designated by (M) are those which a Bus Master must generate. The Bus Master currently controlling the bus has the responsibility for faithfully generating all signals of type M during the duration of its control of the bus.

S stands for Bus Slave. A Bus Slave need generate only that subset of of type S signals which are necessary to communicate with Bus Masters which have the ability to address the slave.

B stands for Bus. Any bus signal which is not of type M or S is by default type B. This is not to say that some Bus Master is not in fact generating one or more type B signals. Rather a type B signal is one that (a) not all Bus Masters are required to generate, and (b) not any Bus Slave is required to generate.

A Bus Master is, by definition, a bus device which generates at least all of the type M signals. A Bus Slave is a bus device which generates some subset of type S signals. A Bus Master may also be a bus slave and vice-versa. Memory devices are almost always Bus Slaves while DMA devices are usually both a Bus Master (data transfers) and a Bus Slave (accepting commands). Central Processing Units are usually Bus Masters.

[2]The 8 status signals are: SMEMR, SINP, SM1, SOUT, SHLTA, SSTACK (not specified), $\overline{SWO}$, and SINTA.

[3]The 6 command/control signals are: PHLDA, PSYNC, PDBIN, PINTE (not specified), $\overline{PWR}$, and PWAIT.

[4]The 16 address signals are: A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1 and A0.

PRELIMINARY SUBJECT TO REVISION

S-100 Bus Signal Definitions

[5]Data output is specified relative to a Bus Master. By definition, data which is transmitted by a Bus Master is always data output and occurs on the DO bus.

[6]Data input is specified relative to a Bus Master. By definition, data which is received by a Bus Master is always data input and occurs on the DI bus.

[7]A bus cycle is a collection of bus states (BS_α). A bus cycle always starts with a BS_1 state which is followed by a BS_2 state. After BS_2 comes an indeterminate number of BS_w states. A bus cycle may have zero BS_w states or it may have an arbitrarily large number of BS_w states. BS_3 is the bus state which follows BS_w (or BS_2 if there are no BS_w states present). BS_3 is followed by zero to three BS_i states. A BS_3 or BS_i state terminates a bus cycle.

[8]The DO bus is the following set of signals: DO7, DO6, DO5, Do4, DO3, DO2, DO1, DO0.

[9]The DI bus is the following set of signals: DI7, DI6, DI5, DI4, DI3, DI2, DI1, DI0.

[10]These signals should be generated by an open collector bus driver capable of sinking at least 20 ma. at no more than .5 volts.

Signal Characteristics

Bus drivers must sink at least 24 ma. at no more than .5 volts. Except for open collector drivers, they must source at least 2 ma. at no less than 2.4 volts.

Bus receivers must have diode clamps to prevent excessive negative excursions. Any bus signal less than .8 volts must be recognized as a logic zero and any signal more than 2 volts must be interpreted as a logic one.

Receivers are to source no more than .8 ma. at .5 volts and are to sink no more than 80 µa. at 2.4 volts. The capacitive load of an input from the bus must not exceed 25 pf.

S-100 Bus Signal Definitions

Bus State Comments

BS_1 is the initial bus state of a bus cycle. The address lines are in a state of flux during through most of BS_1 and PSYNC is active starting with the second half of BS_1.

BS_2 is the second bus state when address data, status, and ready signals become stable.

BS_w states occur as needed to synchronize a Bus Master with a Bus Slave which has brought one of the ready lines false.

BS_3 is the data transfer state when a Bus Master transfers data to a slave or vice-versa.

BS_i is a state during which the bus is idle.

Timing Notes

All timing references in the timing diagrams are specified at the midpoint of the rising or falling edge of the signal. Rise and fall times are not to exceed 50 ns.

All signals referred to in the timing diagrams are S-100 Bus signals with the exceptions in note 6.

[1]The falling edge of $\overline{PWR}$ must occur within the area shown. The rising edge must occur within a similar area of the next Bus State.

[2]BS is either BS_2 or BS_w.

[3]Addresses, Data, and Status signals must remain stable during BS_w.

[4]The interrupt lines must be stable for the period shown in the Bus State preceding BS_1 of an op-code fetch. The proposal is that when an interrupt line is true, it remains true until the CPU responds. Normally, this response would be an I/O instruction that addresses the interrupting device.

S-100 Bus Signal Definitions

[5]The rising edge of PWAIT must occur within the area shown. The falling edge must occur within a similar area following the rising edge of the logical AND of PRDY and XRDY.

[6]Signals prefixed by "DMA" refer to internal logic of the new Bus Master. These signals control the buffers of this Bus Master which drive the Command and Control, Status, Address, and Data output Bus lines. The timing diagram depicts logic levels which are high when these buffers are disabled.

[7]BS_β is either BS_3 or BS_i

[8]BS_γ is either BS_i or BS_1

PRELIMINARY SUBJECT TO REVISION

S-100 Bus Timing

All times in nanoseconds
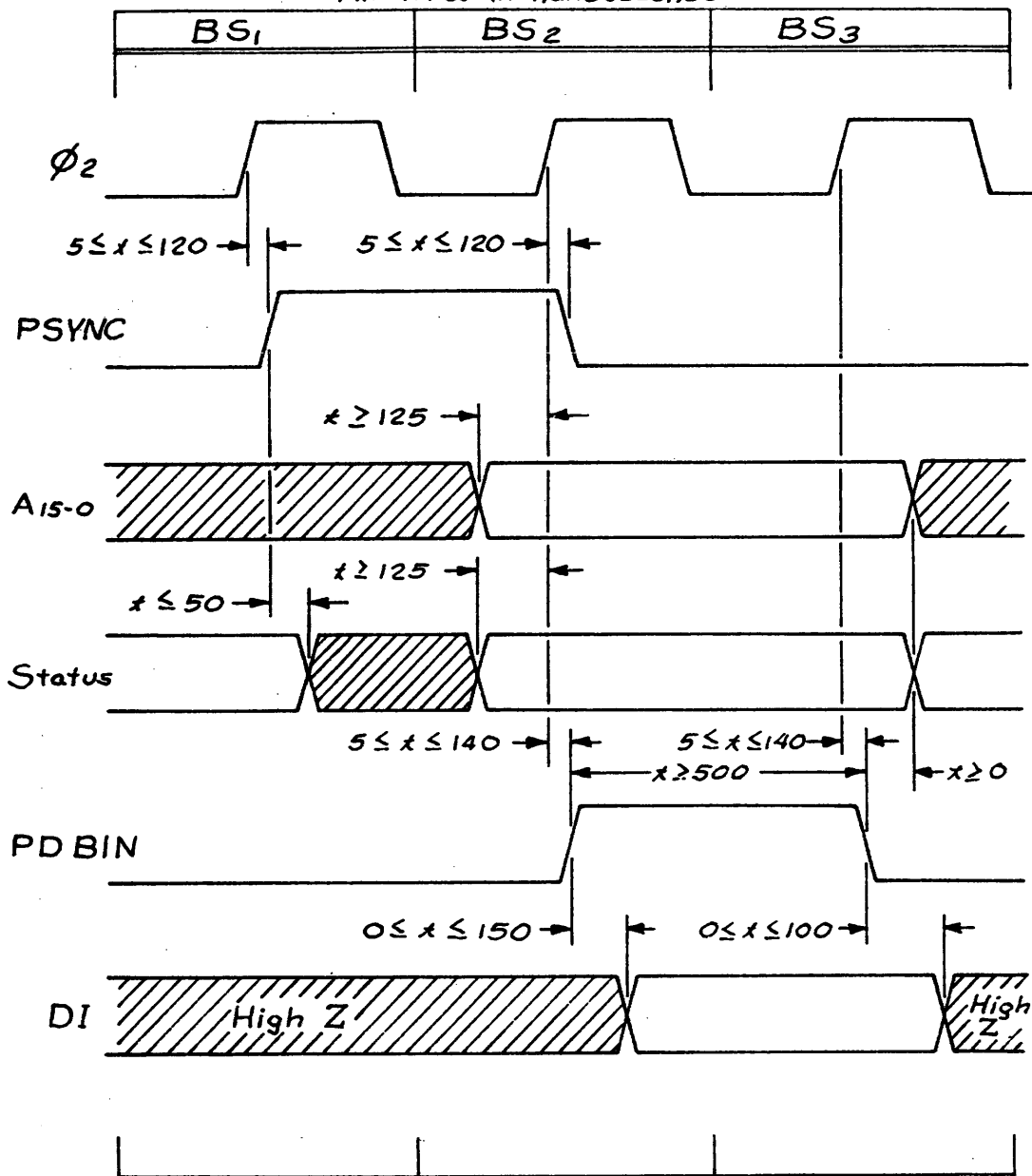
Memory or I/O Read

PRELIMINARY SUBJECT TO REVISION

Memory or I/O Write
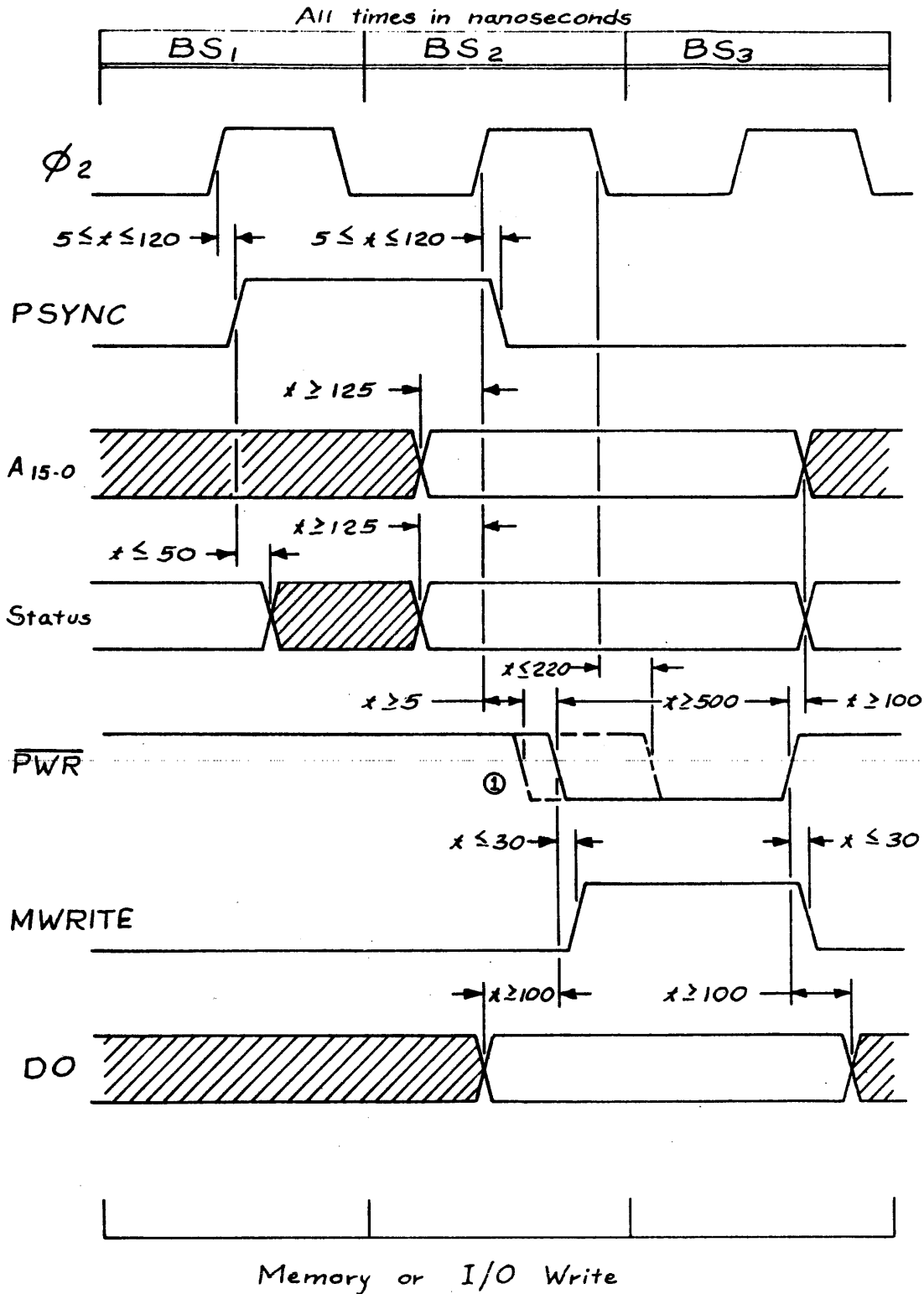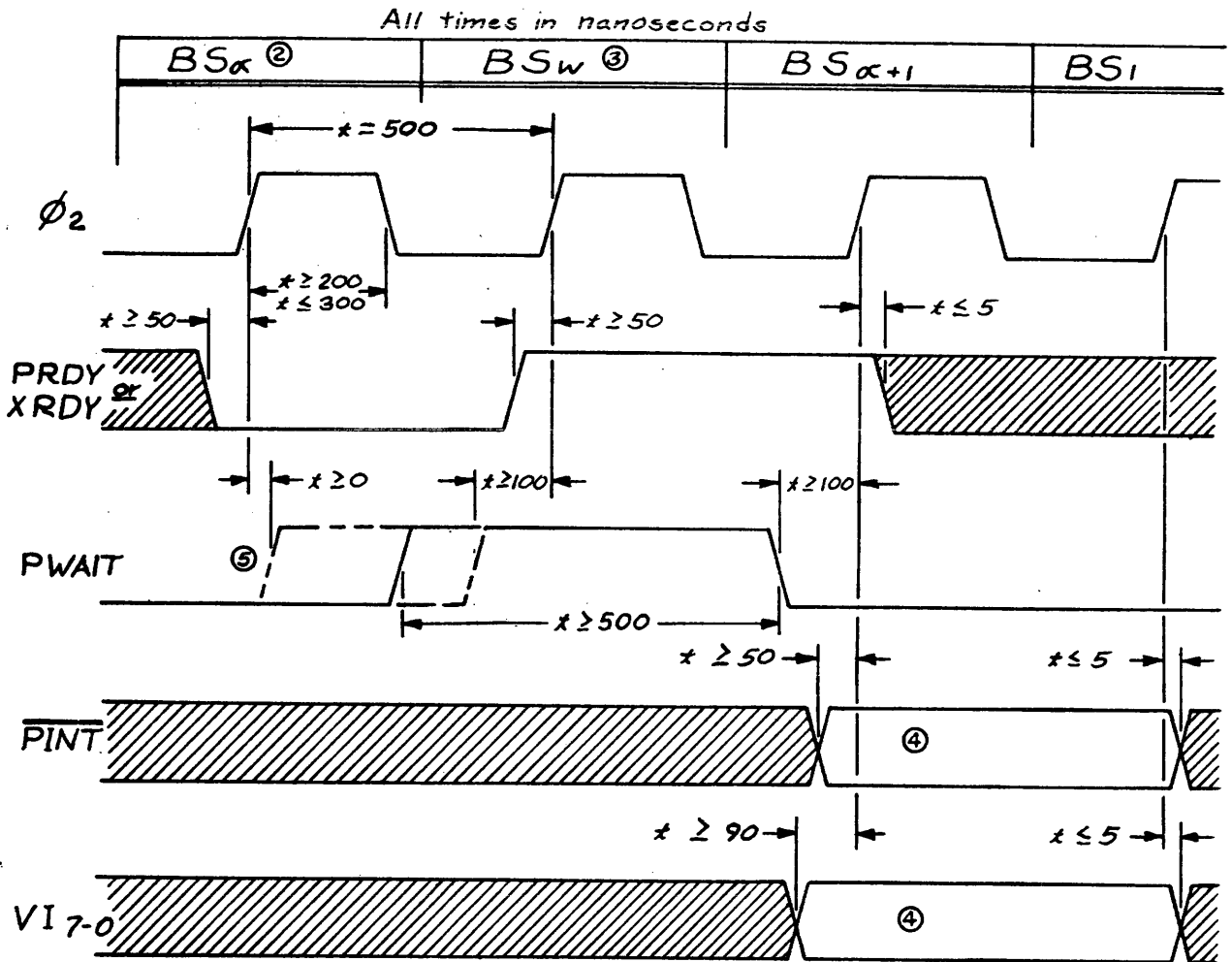
Interrupt and Wait Timing

All times in nanoseconds

BSα ②	BSβ ⑦	BS⅄ ⑧	BS⅄ ⑨

$t = 500$

ϕ_2

$t \geq 200$

$\overline{PHOLD}$

$t \geq 150$

PHLDA

$t \geq 5$ $t \geq 0$

$\overline{ADD\ DSB}$,
$\overline{DO\ DSB}$,
DMA C/C DSB
⑥

$t \geq 200$ $t \geq 200$

$\overline{C/C\ DSB}$, $\overline{STAT\ DSB}$,
DMA ADD DSB, DMA STAT DSB,
DMA DO DSB
⑥

$t \geq 200$

Bus Exchange Timing

PRELIMINARY SUBJECT TO REVISION

Direct Memory Access (DMA) Requirements

Introduction

A DMA cycle is a special case of a Bus Master taking over the Bus to execute a memory read or write cycle. A DMA device is required to generate all type M (Bus Master) signals on the Bus.

The Bus Exchange

PHOLD is the signal used by one Bus Master to request that another Bus Master give up control of the Bus. PHOLD must not be asserted true unless PHLDA is false and PRIORITY (if implemented) is true.

One Bus Master (CPU) relinquishes control of the Bus to another (DMA) as shown in the Bus Exchange Timing diagram. The DMA device must control the CPU's bus drivers through the use of ADD DSB, DO DSB, STAT DSB and C/C DSB. It must also control its own bus drivers through the use of signals similar to those shown in note 6.

The CPU (current master) and the DMA device (new master) must both drive the Command and Control signals for at least 200 ns at two different periods as shown in the Bus Exchange timing diagram. During these two times, the Command and Control signals are required to have the following levels:

       1. PSYNC = 0

       2. PWAIT = 0

       3. PHLDA = 1

       4. PDBIN = 0

       5. PWR = 1

The DMA cycle timing sequence which follows is a suggested implementation that meets all the requirements of the generalized Bus Exchange timing. The sequence is controlled by the edges of ϕ_2. At some previous time, $\overline{PHOLD}$ was asserted according to the limitations described in the first paragraph of this section. PHLDA is asserted true by the CPU during BS_3 of the last CPU bus cycle. The bus exchange begins on the falling edge of ϕ_2 while PHLDA is true (labeled 1 on the timing diagram). The DMA bus cycle then proceeds as described in the following section. At edge 8 of ϕ_2, $\overline{PHOLD}$ is driven false by the DMA device and henceforth the CPU is again in control of the bus.

Proposed DMA Cycle Sequence

ϕ_2 edge:

1. CPU Address and Data bus drivers turned off. DMA Command and Control drivers turned on. The CPU and DMA Command and Control signals must match values described in the previous section.

2. CPU Status and Command and Control drivers turned off. DMA Address, Data output and Status drivers turned on. PSYNC = 1.

3. No change.

4. PSYNC = $\emptyset$. PDBIN = 1 if memory read or $\overline{PWR}$ = $\emptyset$ if memory write.

5. No change.

6. PDBIN = $\emptyset$ and $\overline{PWR}$ = 1.

7. CPU Command and control drivers turned on. DMA Address and Data output drivers turned off.

8. CPU Address, Data output, and Status drivers turned on. DMA Status and Command and Control drivers turned off. $\overline{PHOLD}$ = 1.
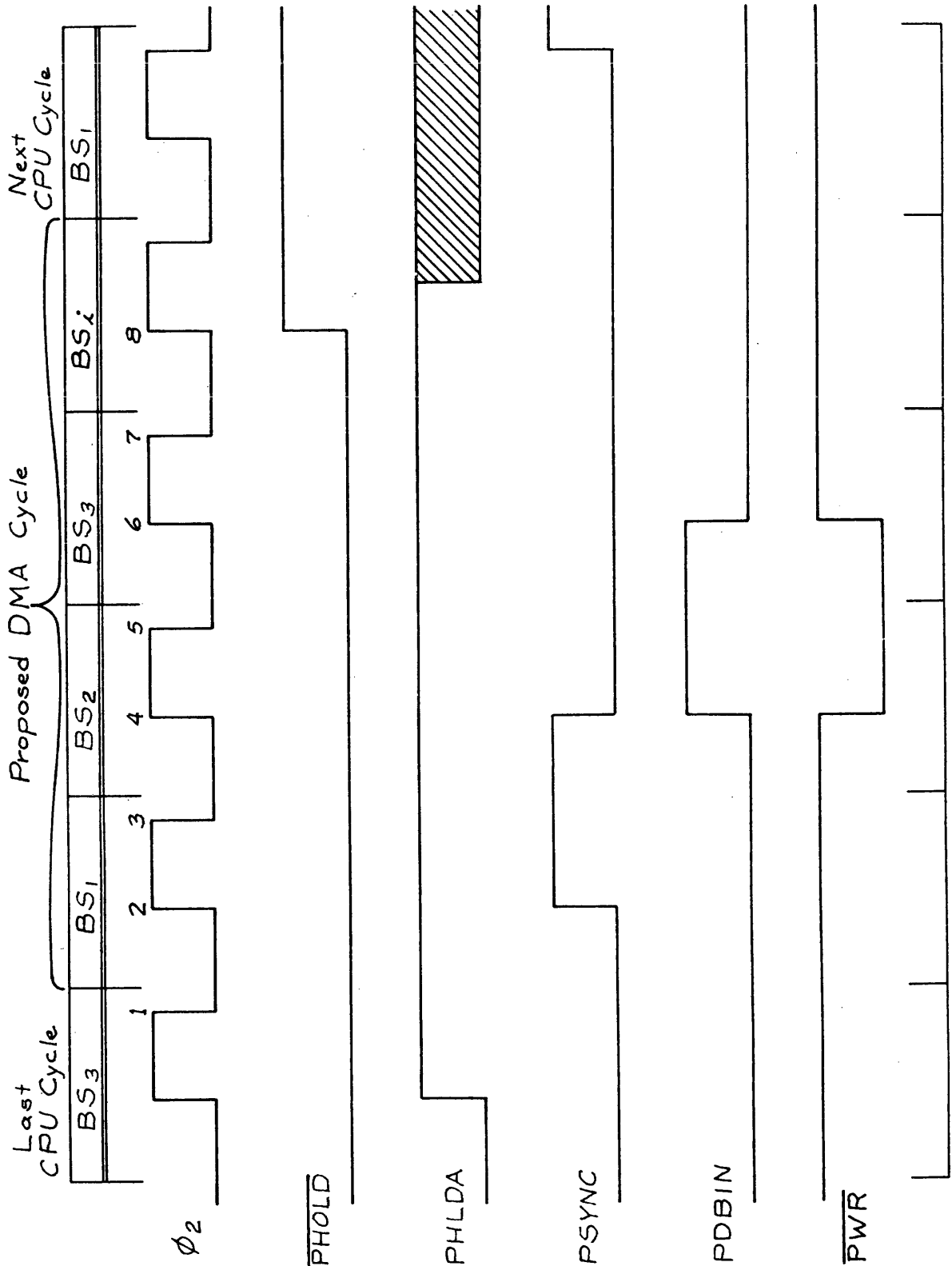
PRELIMINARY SUBJECT TO REVISION

TWO CHEAP VIDEO SECRETS

Don Lancaster
Synergetics

CHEAP VIDEO

Cheap Video is a brand new collection of hardware and software ideas that dramatically slash the cost and complexity of both alphanumeric and graphics microprocessor based video displays.

A typical cheap video system (A-1) lets you do things like a 12 X 80 scrolling display using only seven ordinary IC's with a total circuit cost as low as $20, and transparently running on a microcomputer system that still has as much as 2/3 of its throughput remaining for other programs.

Cheap video displays run on an ordinary TV set with unmodified video bandwidth, even when doing 64 or 80 character lines. Changing a single IC switches you between upper or combined case alphanumerics or high resolution or color graphics modes.

The basic idea behind cheap video is to totally eliminate any TVT system timing and let the microprocessor do all the work. As (A-2) through (A-4) show us, the object is to use both the existing microcomputer and tv set with a minimum of modifications, putting as little dedicated hardware as possible between the two.

There are two key secrets involved in cheap video. One is a software secret called a Scan Microinstruction. The other is its hardware companion called an Upstream Tap.

Together, the scan microinstruction and the upstream tap cause the microcomputer to output characters at a rate fast enough for direct video use.

THE SCAN MICROINSTRUCTION

A Scan Microinstruction is a subroutine combination of ordinary instructions running at ordinary speed that tricks the computer into putting its program counter on the address bus and sequentially advancing the addresses fed to all memory in the computer at a one word per microsecond rate.

For the 6502, a suitable scan microinstruction looks like this:

```
Enter Via
Subroutine

    ┗━ 6000  LDY  AO  AO
       6002  LDY  AO  AO
       6004  LDY  AO  AO
                .
                .
       601C  LDY  AO  AO
    ┏━ 601E  RTS  60
    ┇
Exit to main
Scan Program
```

As (A-5) shows us, the address lines on a 6502 are normally a mix of "go fetch" values and program counter values. When the program counter is on the address bus, the bus will typically advance at a one or two microsecond rate.

When we do a scan microinstruction, the program counter appears continuously on the address lines (A-6) and advances the address bus binary counter style, once per microsecond.

The scan microinstruction is usually stored in a small 32 x 8 PROM. The length of the sequence decides the total number of characters or graphics chunks output per line.

As the address bus advances during the scan microinstruction, each and every memory in the entire computer is sequentially addressed. By using a redundant calling of the scan microinstruction, the subroutine becomes portable and can be moved around as needed to pick up various lines stored in display memory, or can call the various dot combinations needed for a particular part of an alphanumeric character.

Usually, a scan microinstruction will last 32,40,64, or 80 microseconds, the normal length of a selected character or graphics line. The scan microinstruction is called over and over again as part of a larger main scan program. It is this larger scan program that causes TVT refresh, while the scan microinstruction causes individual characters to be output at a proper rate.

With some add-ons, the TVT refresh process can be made totally transparent, letting you run other computer programs at the same time you provide a continuous display.

THE UPSTREAM TAP

Normally, while the scan microinstruction is controlling the computer, nothing else is allowed data bus access. This means we are addressing everything else in the computer, but preventing everything else from doing anything useful at the same time.

Somehow, we have to get characters out of the display memory when the memory does NOT have data bus access. This is done with the upstream tap of (A-7).

An upstream tap is nothing but eight pieces of wire at the output of the display memory but before the output bus drivers. Extra enable logic activates the otherwise normal display memory RAM during a scan microinstruction but does so only as far as the upstream tap.

The upstream tap in turn is connected to the interface hardware for conversion to serial video.

FOR MORE READING

Complete details on cheap video techniques appear in the Sams Cheap Video Cookbook.
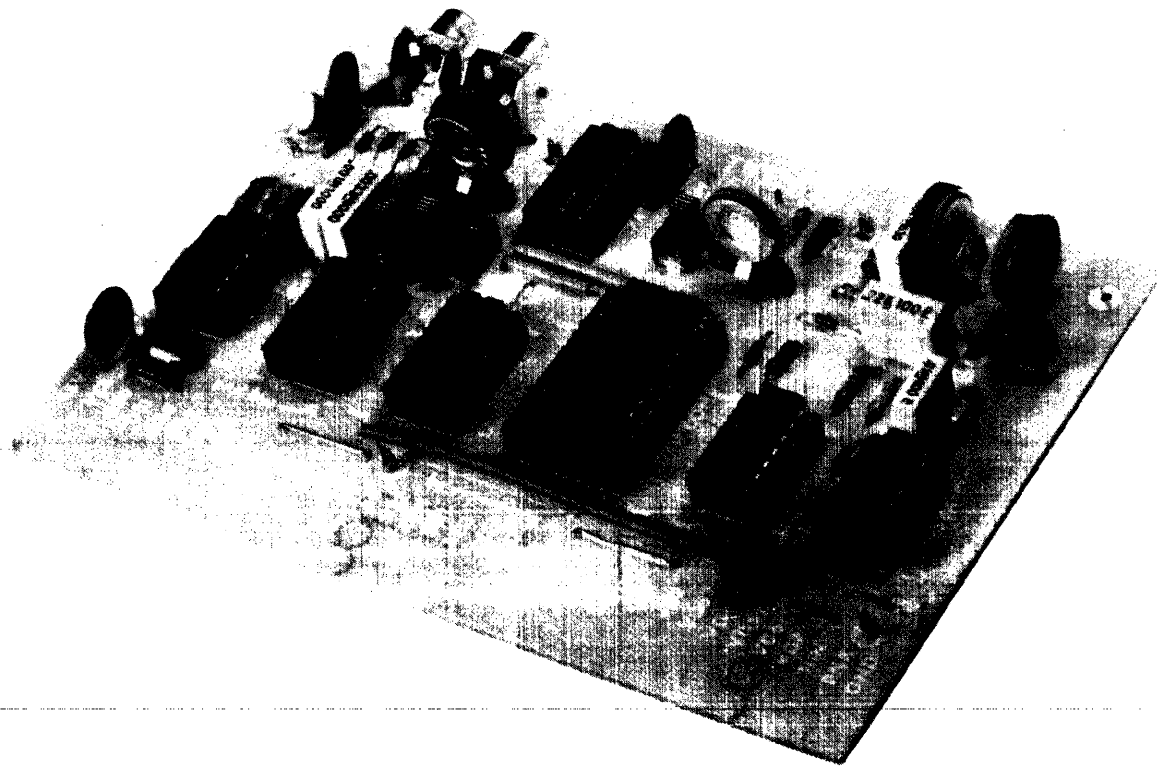
A-1   This PAIA TVT 6-5/8 is a typical cheap video system.   Only
      seven low cost integrated circuits are needed for a high
      performance alphanumeric or graphics video display.

A MICROCOMPUTER HAS.....

* A [CLOCK] THAT CAN BE USED FOR ALL SYSTEM TIMING.

* A [MEMORY] THAT CAN BE USED TO STORE CHARACTERS, OP-CODE, OR GRAPHIC SYMBOLS.

* SOME [ADDRESS] LINES THAT CAN SEQUENTIALLY ACCESS CHARACTERS.

* SOME [DATA] LINES THAT CAN CONTROL HOW WE ACCESS MEMORY.

* AND [CONTROL] LINES THAT LET US SWITCH BETWEEN VIDEO AND NORMAL OPERATION.
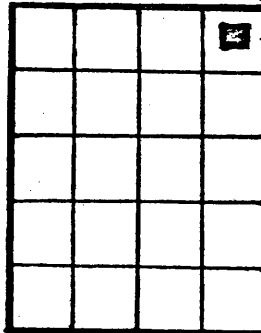
A-2   Cheap Video gets us from here . . . . . .

A TV SET NEEDS......

* VIDEO DOTS TO FORM CHARACTERS OR GRAPHIC SYMBOLS, ARRIVING AT THE RIGHT PLACE AT THE RIGHT TIME.

* HORIZONTAL AND VERTICAL SYNC PULSES TO LOCK THE TV TO THE MICROCOMPUTER.

* COMPENSATION OR SOME OTHER WAY TO MAKE UP FOR A LIMITED BANDWIDTH

* SOME INPUT ACCESS SUCH AS A CLIP-ON RF MODULATOR OR A HEADPHONE JACK USED AS A DIRECT VIDEO CHANGEOVER SWITCH.
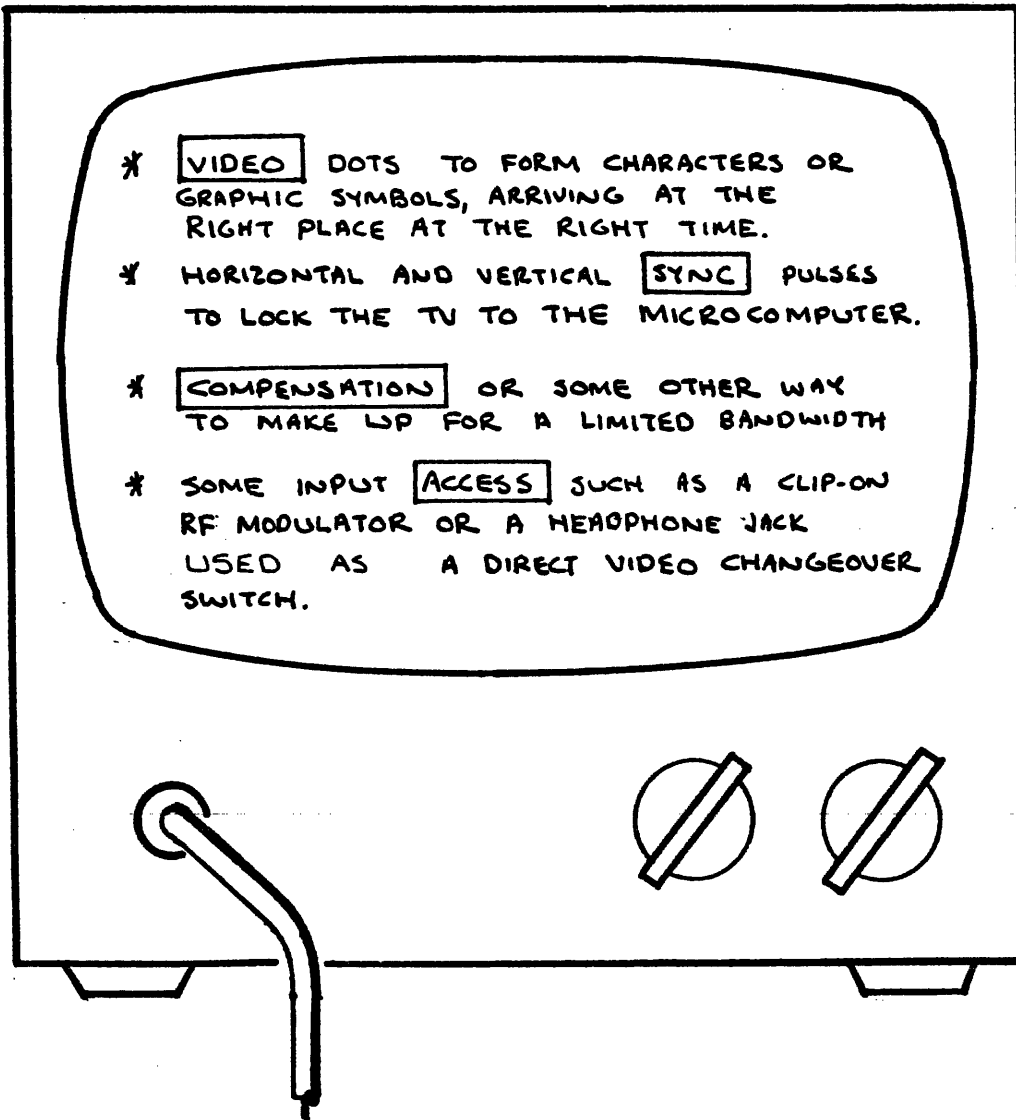
A-3 . . . . To here with . . . .

COMBINE THEM WITH INTERFACE HARDWARE TO BUILD A MICROPROCESSOR BASED CHEAP VIDEO DISPLAY.
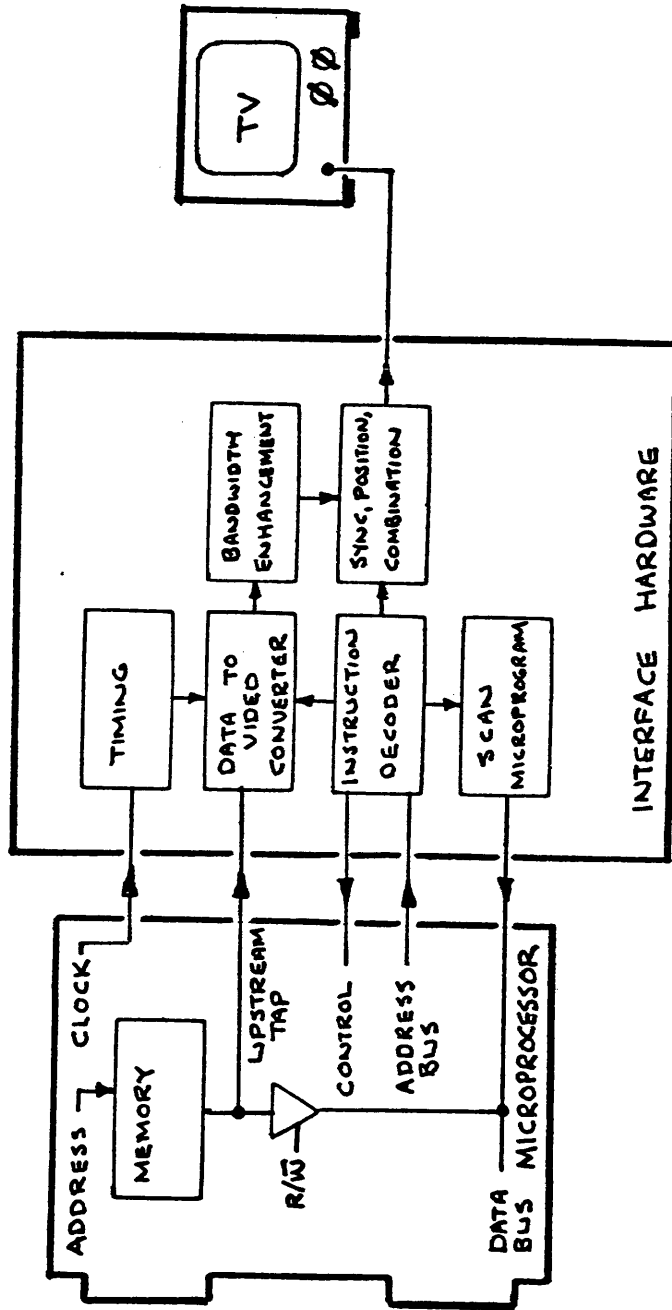
.... As little of this as is possible.

A-4
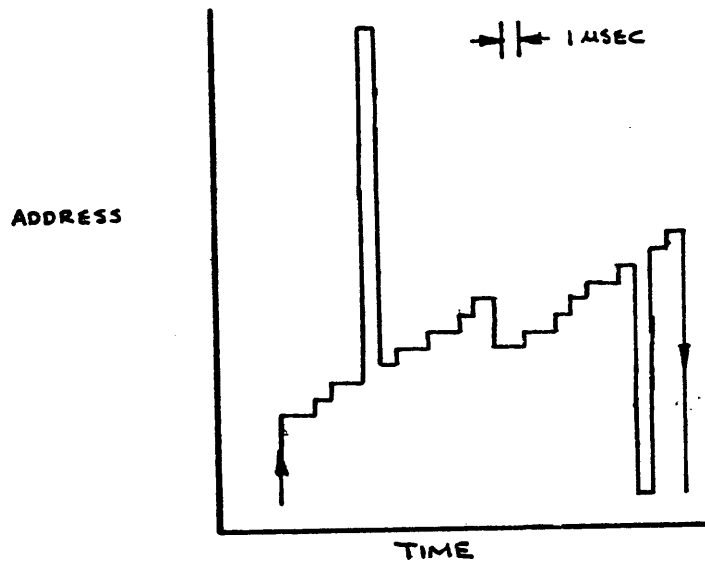
A-5 Typical behavior of a 6502 address bus during
     a normal program. Bus may advance at a one or two
     microsecond rate, can loop, or can fetch higher
     or lower memory values.
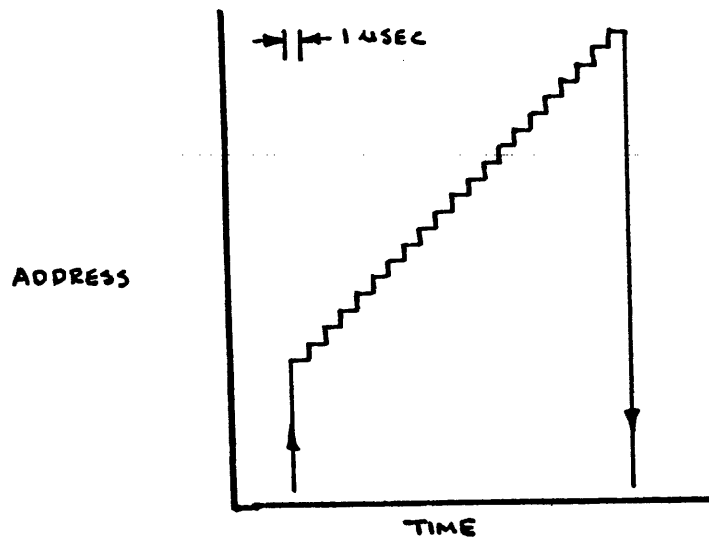
A-6 A secret cheap video scan microinstruction forces the
     6502 to uniformly advance addresses once each microsecond
     from a starting address. This sequentially accesses
     a block of display memory corresponding to a horizontal
     character line or its graphics equivalent.

EXTERNAL
TRI-STATE
BUS DRIVERS

—— = EXISTING
—— = NEW

CHIP SELECT LINE

NEW $\overline{CS}$ LOGIC PROVIDES NEGATIVE LOGIC OR OF SCAN & USUAL $\overline{CS}$

DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0

EXISTING DATA BUS

CUT
ADDRESS
K∅
R/W̄

7
6
5
4
3
2
1
0

NEW UPSTREAM TAP

NEW SCAN ENABLE
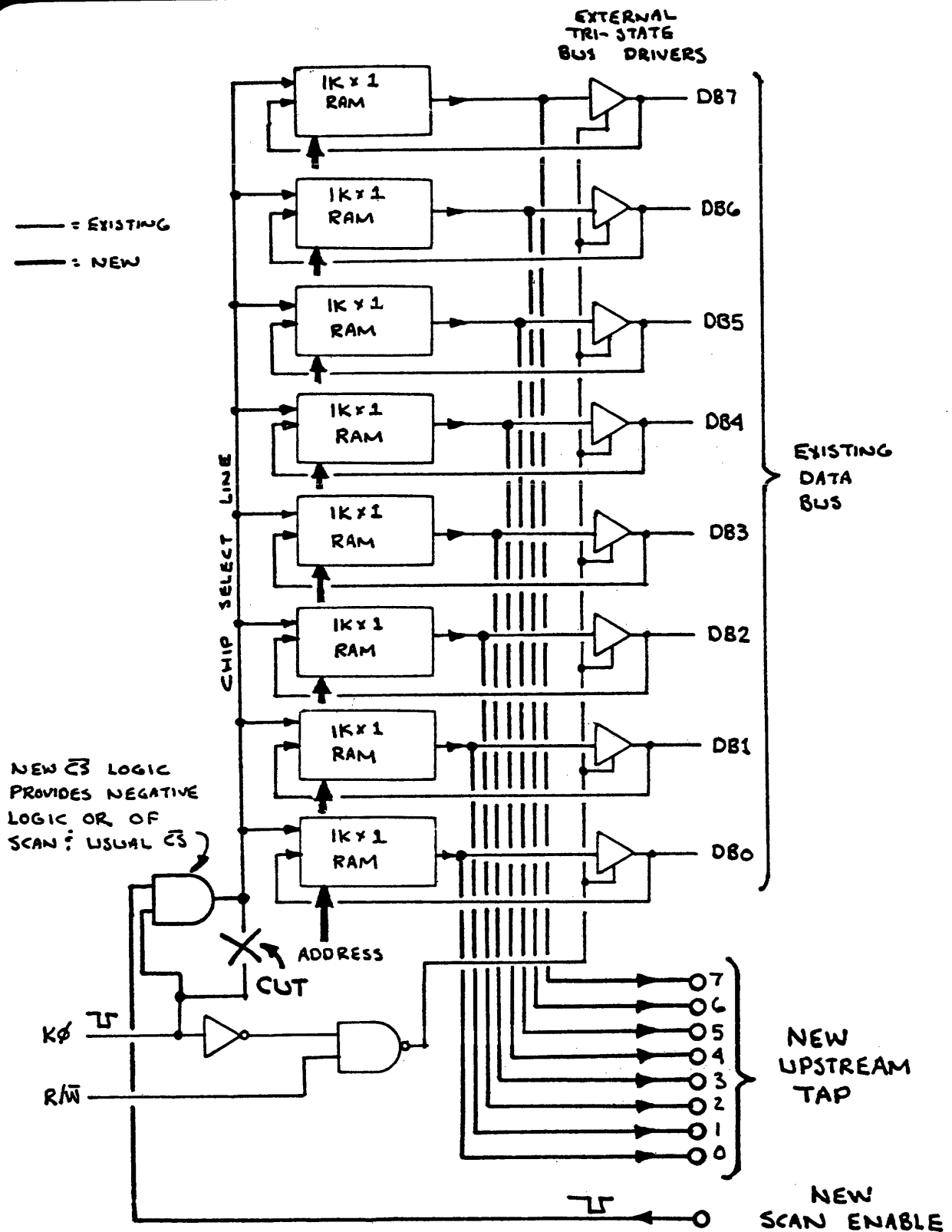
A-7   A secret _upstream_ _tap_ works with the scan microinstruction
to output characters. Total DMA hardware consists of eight
pieces of wire.

A RECIPE FOR HOMEBREW ECL

Chuck Hastings, 4890 Hamilton Avenue, San Jose, CA 95130

Abstract

Emitter-coupled logic (ECL) is understood by most computer designers to be the fastest stuff available, which it is — and as too diffi-cult for anyone but the largest companies to design with, which it isn't! If an appropriate recipe is followed, ECL systems can be developed with very limited resources with as good, or better, chances of technical success as with equivalent TTL systems. Thus, homebrew ECL is a serious possibility for applications which need the speed. Such applications may occur in some technical approaches to music synthesis, speech analysis, and personal scientific computing involving matrices or partial differential equations.

Such a recipe isn't written down anywhere — existing ECL tutorials make ECL design sound formidable. However, a careful amateur can achieve a reliable 100 MHz small system today if he knows what to do. This paper will present a practical recipe, used once successfully, for designing, building, and troubleshooting a small ECL system with the level of resources available in a well-equipped homebrew lab.

This recipe was developed during the course of one task in a project at Racal-Milgo, a medium-sized Florida company with no previous ECL systems experience. The circumstances were in many ways quite similar to those of a home-brew project. The outcome of the task was a 24-bit general-purpose stored-microprogram computer, capable of 6,000,000 three-address fixed-point add/subtract/Boolean instructions or 900,000 fixed-point multiply instructions per second, which was completed and has since been operated 10 hours a day for several months in a signal-processing system.

Introduction

What I hope to do in this presentation is to get you thinking about emitter-coupled logic (ECL) as a viable alternative for homebrew projects requiring very high processing speed, perhaps in music synthesis (see Reference 1), speech processing, or simply fireside number-crunching.

Why ECL? For openers, the industry-standard 10,000-series ECL (hereafter referred to as "10K") offers at least twice the net speed of Schottky TTL when actually designed into typical systems. 10K provides a more natural and less brute-force approach to high-speed signal transmission than Schottky, and is in a number of respects actually easier to use.

ECL has probably not been considered for many applications where it would have been appropriate, both in industry and more recently in hobby work, because people tend to be scared to death of it. Frankly, ECL has an image problem! (See Figure 1 below.) Like many image problems, this one has some basis in truth; but there has been a considerable overlay of exaggeration, distortion, and mythology, which I will do my best to dispel based on the results obtained in one medium-sized computer hardware development project.

Much of what I have to say concerns a subject euphemistically called "interconnection practice," which means all the things you have to do to keep your logic from being thoroughly confused by its own noise after you turn it on. Except as occasionally noted, all of my remarks concern 10K in a wirewrap environment. Later on, I'll have a little to say about other ECL families such as MECL III, PECL III, and Fairchild 100K.
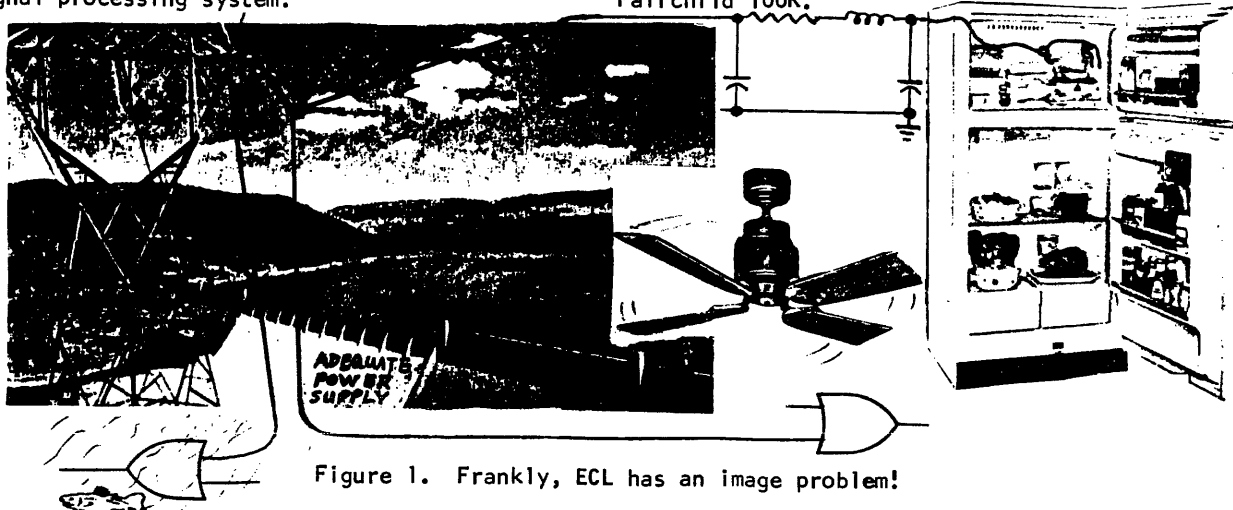
Figure 1. Frankly, ECL has an image problem!

A good wirewrap board, believe it or not, is an excellent signal environment for high-speed logic. I have met people who solemnly claimed that one can't wirewrap ECL, but it just ain't so. Communications Satellite Corporation, for instance, has done it for years. I have also met people who claimed that wirewrap fabrication was something which one does only for prototypes, and that it is too expensive to be a manufacturing technique; but Modular Computer Systems, in Florida near where I used to live, has been cranking out wirewrapped minicomputers since about the beginning of this decade. Much of the wirewrap equipment used in industry is made by Gardner-Denver, but there is also a company called OK Machine and Tool Corporation which makes a line of wirewrap equipment specifically designed for use by hobbyists.

Since most hobbyists probably prefer to have their systems work without a major initial checkout hassle, my interconnection-practice recipe probably errs on the side of overkill. If for some underground entrepreneurial reason you are intensely concerned with the cost of replicating a homebrew ECL system once it is working, you can do a cost-reduction job by deleting some of the practices I am advocating one by one until the system goes bananas. But don't start out doing an el cheapo job — if the system doesn't work at all, you may not have the equipment, resources, and patience to find out why. Big companies do have the luxury of trading off more product development engineering hours against lower manufacturing costs, but you probably don't. The first time you do it, do it right.

An Astounding Claim

The primordial fear of ECL in the industry is so great that it requires some chutzpah on my part simply to state, straight out, that yes, you too can successfully build, debug, and operate ECL logic systems in your spare bedroom, garage, or rumpus room - just like TTL and MOS. You don't have to have the vast resources of a huge company like Control Data, Univac, IBM, or Burroughs behind you to succeed - or even those of a rather unusual small company such as Cray Research or Biomation, to name two with some obvious ECL expertise.

I make this statement not on the basis of having an ECL computer running in my spare bedroom — since I have five children, I don't even _have_ a spare bedroom — but on the basis of successfully developing a medium-sized, high-performance ECL midicomputer under what might be called primitive industrial conditions, at a company (Racal-Milgo, in Miami) having no prior experience building either ECL systems or computers. The company management did not particularly even understand digital computers,
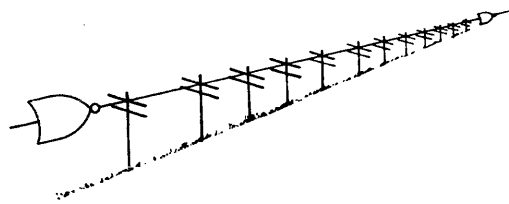
although they did have some expertise in analog computers. The backup resources which one expects to find in place in even a small computer mainframe house simply weren't there.

To top it all off, I myself am a computer systems type — ones and zeroes, architecture, logic design, machine-level software, micro-programming — with very little expertise in, say, linear circuit design or electromagnetic field theory. All the same, with one sharp technician working with me full-time plus part-time help from a few other people, I was able to get a high-performance digital system of about 900 ECL 10K DIPs developed and operating in about 15 months. Thereafter, for several months, it was operated many hours a day five or six days a week, as part of a larger signal-processing system, with very few maintenance problems. If I can do something like that, probably you can too.

The Miami Number-Cruncher

The architecture of this midicomputer is not the main point of my presentation, so I'll say just enough about it to put it in perspective. It is three-address, with a 48-bit instruction word and a 12-bit data word. Instructions and data come from separate memories with separate addressing spaces ("Harvard architecture"). Arithmetic is generally 24-bit twos-complement, with some 12-bit operations also available. The minor cycle ("clock" interval) is about 10.17 nanoseconds, which is the reciprocal of the 98.304 MHz basic frequency. One microprogram step requires a major cycle consisting of 5 to 12 minor cycles according to a 3-bit microprogrammed field.

Normal execution time for a 24-bit add or subtract instruction is 163 nanoseconds, and a Boolean instruction requires one minor cycle less; instructions of both of these types require two major cycles. The time of 163 nanoseconds is for a _memory-to-memory_ operation, not merely register-to-register, since the main data memory (4K 12-bit words) is comprised of 20-nanosecond-access 1Kx1 ECL memory DIPs (type 10415A/10146). Two copies of all main memory words are implemented, in order to avoid the penalty of an extra major cycle on each execution of one of these instructions.

The approximate times for some other 24-bit three-address operations are: 1.1 microseconds for multiplication, 3.5 microseconds for division, and 13 microseconds for the square root of a sum. There are both single-word and block-

oriented input and output instructions, and an external command instruction, with a fully asynchronous handshake control philosophy. All instruction sequences are controlled entirely by stored-microprogram techniques.

The computer itself, including both data and instruction memories, occupies three large (418 DIP locations) wirewrap boards mounted in aluminum frames, and draws a little more than 300 watts. It is part of a larger experimental signal-processing system for a proprietary real-time application, and was never intended to be a product in its own right.
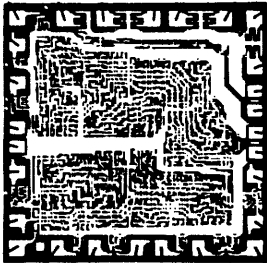
Motorola
type 10181
ALU

Test Equipment

Probably the scale of this machine is larger than should be attempted under home lab conditions! Nevertheless, the only important resources I had which would be difficult to match in a well-equipped homebrew lab were a much larger test equipment budget and other people to do some of the work.

By far the two most important pieces of test equipment were a Tektronix type 485 portable 350-MHz oscilloscope and a Data I/O model VI PROM programmer. The 485 is a marvelous scope, but is much higher in performance than needed for routine measurements, even in ECL work, and is priced out of the reach of most hobbyists. I had previously used a Tektronix 150-MHz type 454 scope for TTL work, and this model should be quite adequate for ECL. The 454 is now several years old, and I am told that the going price for a rebuilt one is about $1800. Since that is probably still too much, unless two or three hobbyists share one, I will state a belief that a 50-MHz or 60-MHz scope such as a Tektronix type 547 or type 453 could be used effectively as long as its limitations were understood and conservative design practices were followed. More on this later.

As for the PROM programmer, this was needed because Miami is, for digital systems work, an isolated area far from the bright lights of technology. Here in Silicon Valley an enterprising hobbyist should be able to buy preprogrammed ECL PROMs from a distributor or even a manufacturer, although it may still be a while before they are sold over the counter in every shopping center.

ECL Transmission Lines — Image and Reality

Perhaps the single statement which scared me most, as I embarked on the development of Racal-Milgo's ECL number-cruncher, was

> "In high speed systems, the inductance, capacitance, and signal delay along interconnections cannot be ignored. The only practical way of dealing with these factors is to treat interconnections as transmission lines." (Reference 2, page VI. This is a good book even if it scares you a little.)

This statement is of course literally true in a technical sense, and yet is enormously misleading. It raises vivid mental images of huge steel towers marching across the wasteland, with long wires dangling from brown insulators in catenary curves, as in the portion of Figure 1 where I pasted up a picture of Grand Coulee Dam. Thus, it tends to scare hell out of people who are used to treating logic signals simply as wires from one point to another, as in garden-variety or low-power Schottky TTL.

However, the real truth is that any type of logic operating at relatively high speeds has to be treated with extreme care - not just ECL. I have found that there is essentially no difference between the care which must be taken to design a good Schottky TTL system, with respect to interconnection practice, and that needed to design a good ECL system — but the ECL system will run somewhat more than twice as fast, is actually easier to debug and get running, springs fewer nasty surprises on you during the checkout process, and tends on the whole to come closer to treating you right if you have treated it right.

ALL THAT THE TRANSMISSION-LINE PROPERTY ACTUALLY MEANS IN PRACTICE IS THAT THE LAST THING ATTACHED TO EACH AND EVERY SIGNAL WIRE IN AN ECL COMPUTER IS A RESISTOR. If your system is entirely wirewrapped, as mine was, on a board with good voltage planes (more on that later), the characteristic impedance of each wire is that of a "wire over ground" and is somewhere between 100 and 120 ohms. And, if you wade through all the formidable equations in Z's and i's in Motorola's various handbooks, one of the things you discover is that nothing really bad happens — just a few per cent reflection — if there is a fair amount of mismatch between the line and the terminating resistor. Because the wire over ground on a wirewrap board full of other wires is at a varying height anyway, and the characteristic impedance depends on that height, the characteristic impedance of that wire is bound to be "smeared out" and not very precise anyway.

I used two types of resistors: thick-film, which come in 16-pin DIPs costing $1.25 to $2.50
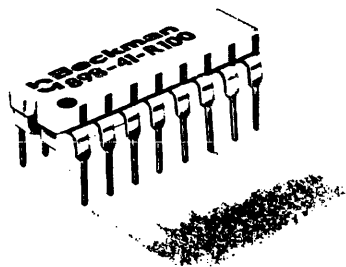
each depending on quantity, from Beckman, Bourns, and other vendors, with 11 individual resistors per DIP; and 1/8-watt carbon resistors, which are so tiny that the leads can be wirewrapped around backplane pins. Most of the resistors overall were of the thick-film DIP variety, and the ratio of ECL IC DIPs to resistor DIPs was roughly 3:1.

There are also single-inline (SIP) resistor packages, and "active terminators" (Fairchild type 10014) with a nonlinear current vs. voltage characteristic. In any case, the other end of the resistor is terminated to a supply voltage (V_{TT}) intermediate between the two usual supply voltages (V_{CC} and V_{EE}).

The "Thevenin equivalent" scheme is a second way of terminating a signal line in its characteristic impedance. This approach avoids having a V_{TT} plane at all, and presumably also inflicts less noise from the logic on the main power supplies in some cases, but dissipates about 11 times as much extra power per line termination as does the previous method. In the Thevenin equivalent scheme, the termination point for each signal line is connected to both V_{CC} and V_{EE} by resistors, whose values are chosen to form a voltage divider (Thevenin network) such that the voltage drop produces V_{TT} at the termination point. Beckman also makes Thevenin network thick-film termination resistor packs, with 4 such networks per DIP.

To be sure, there are other ways of approaching signal interconnection besides my recipe of terminating each and every signal line in its characteristic impedance, if you are (a) skilled in linear circuit design, and/or (b) a masochist. You can simply not terminate the line, and compute out the maximum number of inches or tenths of an inch allowable for line length under each given set of conditions for each signal line. Or you can use "series termination," in which there is a resistor in between the output stage of your gate or whatever and the input (just one per line, with many lines fanning out from one origin) which is being driven. Possibly, in a big-company environment where one is using 17-layer etched circuit boards like those used in the Texas Instruments Advanced Scientific Computer, there are real advantages to these schemes. In the wirewrap world there aren't any, and it is better to terminate each and every signal line in a resistor and then relax, since you have thereby at one stroke slain most of the big,

scary goblins of high-speed logic systems — crosstalk, ringing and reflections, limits on line length, and so forth.

Some Good News

And now for some pleasant surprises.

First, 10K outputs are "open-emitter," and may be tied together in almost the same way as TTL open-collector outputs, but with wire-ORing of outputs viewed as assertive-high and wire-ANDing of outputs viewed as assertive-low. (I'll delve into the mystique surrounding logic polarities before I get done, I promise.) However, since the termination resistor has a value determined by the characteristic impedance of the signal line, one no longer has to recompute this value every time the number of driving outputs or the number of driven inputs changes, as one is supposed to when stringing together open-collector TTL. ECL logic isn't slowed down much by stringing together open-emitter outputs; Motorola estimates 50 picoseconds per additional output. When five or more outputs are strung together, one may start to see minor glitches in the waveform, and so I never tried that. Stringing together open-emitter outputs turns out to be a valuable technique in ECL, for two reasons: It does an extra level of logic with essentially no extra logic delay and no additional gates, which together with the usual two-rail outputs makes ECL SSI much more powerful per gate package than TTL SSI. Also, it allows in-circuit stimulation of ECL devices while your system is running, or trying unsuccessfully to run, at full speed; any logic point can be tied to a logic "1" source with impunity in order to change what is happening so that you can study it, which is a very powerful troubleshooting technique. It is normally a no-no in TTL troubleshooting because of an unfortunate tendency to melt IC output transistors in totem-pole devices.

Second, since virtually any ECL IC output stage will drive a 50-ohm line, it will also drive two properly terminated 100-ohm lines going to different places, which is very useful for instance when driving a lot of memory address lines. By way of comparison, there are only a few TTL devices — the 74S140 dual NAND buffer and the 74128 quad NOR buffer, for instance — which will drive such low-impedance es.

Third, once you have bitten the bullet and ninated a signal line in its characteristic impedance, you can stop worrying about how long that line is, at least as long as it doesn't go off the board away from the ground plane. The boards I used were roughly a foot wide and almost two feet long, and some signal lines were longer than two feet, which would be rather unacceptable using TTL gates since the usually

quoted line-length limit is 10 inches. (For TTL 3-state buffers it is much longer.) ECL signals which go off the board should be differential, but even that turns out to be less frightening than it sounds, as will be discussed later on.

Fourth, in ECL the only limitation on fanout that matters is that each additional input connected to a line adds a few picofarads of capacitance, just as additional TTL or MOS inputs do in other systems, and as the number of inputs increases the rise and fall times lengthen a bit. But, instead of a fanout of 10 as for garden-variety or Schottky TTL, or 21 as for low-power Schottky TTL, the fanout limit imposed by driving capability is something like 92, which is as good as infinity for most purposes. I never really had to test this proposition out; it usually was not necessary to go beyond driving 10 to 12 loads with one output, except in special situations like driving memory IC address inputs with buffer gates, and even there I stayed conservative.

Voltage Planes and "Positive Earth"

ECL logic, even easier-to-use 10K, should still be built on a good board for best results. "Good" here means that the voltage plane or planes occupies at least 50% of the available area of the board as it is viewed from above, say by Superman with X-ray vision if the board is multilayer with internal voltage planes. I knew where to get really deluxe boards, from a successor company (Kleffman Electronics, Minnetonka, Minnesota) to one I once worked for, with four complete voltage planes, but up until now these boards have not been offered for public sale. However, a number of circuit-board companies do now offer wirewrap bread-boards which look satisfactory for ECL, and in some cases state such a design objective. (See list in Appendix.) Augat pioneered in this area, with a 3-layer board, and boards with a similar design philosophy are now also available from Excel, Garry, Mupac, and SAE. Interdyne has a rather different type of board which also looks plausible. These various boards do of course cost more than vector board — probably $200-$300 for one to accomodate 150 or so DIPs. In most cases the DIPs plug directly into the holes in the round pins on the board, and no additional IC sockets are needed.

Figures 2 and 3 show an Augat board in top view and local cross-section, and Figure 4 shows Mupac's "Sponge" (TM) board. Reference 3 is a useful technical note available from Augat on wirewrapping ECL logic using their boards.

One of the disconcerting facts about ECL which seems to baffle each person newly introduced to the stuff is that V_{CC} — yes, I did say V_{CC} — is normally specified as +0.0 volts, or "positive earth" as British car aficionados say.
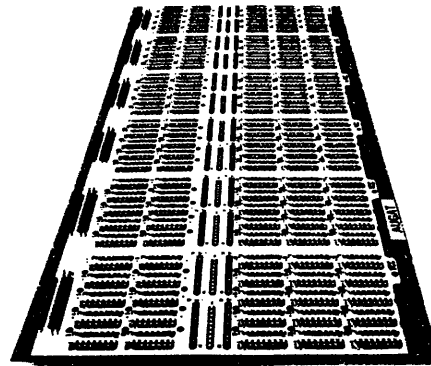
Figure 2. Top view of Augat board.

USE GARDNER DENVER HIT
NO 505279 AND SLEEVE NO
507100 FOR WRAPPING AXIAL
RESISTOR LEADS (NO.26 WIRE)
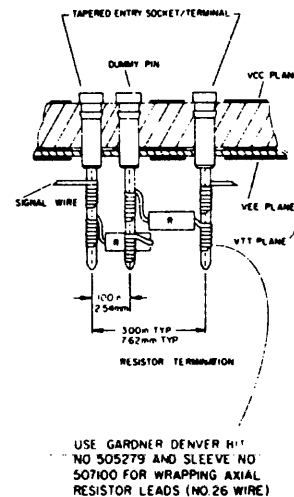
Figure 3. Cross-section of Augat board.
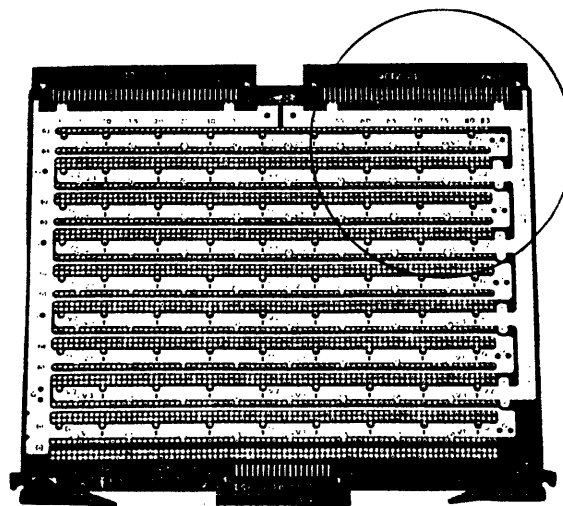
Figure 4. Top view of Mupac board.

After all, everyone who has designed TTL or MOS systems knows that V_{CC} has to be +5.0 volts and that it is the other voltage supply which is at +0.0 volts — why, it is even called ground. So then, what is this V_{EE} which is specified as -5.2 volts? Why isn't V_{CC} specified as +5.2 volts and V_{EE} as ground, the way any normal person would? Certainly the logic doesn't care what the dc potential of various circuit points is relative to Mother Earth, does it? For that matter, why can't ECL run on a 5.0-volt spread between the two main supply voltages like TTL does?

Well, it turns out that when Motorola originally instituted this now-universal +0.0/-5.2 specification, the goal which they were in a subtle way trying to achieve was to get their customers to use the best plane on the board for V_{CC} rather than for V_{EE} in case there was any difference in the extent of the planes. The circuit properties of ECL are such that system performance is affected much more by inadequacy of the V_{CC} plane than by, say, a V_{EE} plane which only covers part of the board and shares the same surface with the V_{TT} plane. To keep the internal workings of ECL ICs from being confused by electrical transients due to their own output stages, most of them (except the ones with particularly serendipitous internal layout) have 2 or even 3 separate V_{CC} pins. Do not, however, draw the conclusion that you must actually connect these different pins to different V_{CC} planes — they don't want you to do that, but rather to connect them separately to the same V_{CC} plane. It makes sense if you think about it.

The Kleffman boards I used had two complete ground planes and two other voltage planes, having been designed to accommodate a mixture of Schottky MSI devices with linears which often required a -5.0-volt supply in addition to the normal TTL supply voltages. I adapted these boards for ECL by using the ground planes for V_{CC} (after all, it is at ground), the TTL V_{CC} plane for V_{EE}, and the -5.0 plane for V_{TT}.

By the way, now that the upside-down supply voltage polarity issue has been disposed of, you will know what I mean when I state that V_{TT} (or the equivalent individual termination points if the Thevenin-equivalent voltage divider scheme is used) is normally specified as -2.0 volts.
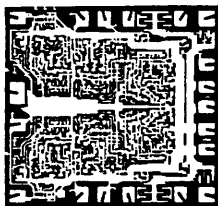
Motorola
type 10186
hex D-flipflop

Decoupling Capacitors

In any high-speed logic system, not just ECL, there should be an easy path for high-frequency noise to get between the two main supply-voltage planes without passing through the logic and confusing the hell out of it on the way. Lower-frequency noise usually is dealt with by connecting a fairly large tantalum electrolytic capacitor, say 22 microfarads or larger, between the two main supply voltages (for ECL, V_{CC} and V_{EE}) at the point where they are brought onto the board, and perhaps at other points on the board also. Higher-frequency noise is similarly shorted out using little ceramic disk capacitors scattered all over the board, mingled with the semiconductors.

Although I have seen printed recommendations as mild as using an 0.01-microfarad disk capacitor for every few ICs, again — as in the case of signal-line termination — my recipe calls for doing it right everywhere to start with and finishing off the goblins for good. Here, doing it right means using one 0.1-microfarad disk capacitor (ten times as large) for each and every DIP on the board, which thoroughly slays many noise problems otherwise likely to be encountered in either ECL or Schottky systems. AVX (nee Aerovox) is the brand I have used, and there are some other vendors also whom I haven't personally calibrated. These capacitors are physically quite small, and cost in the range of 20¢ each in modest quantities.

The reason that I insist (and the people who write ECL applications notes for semiconductor manufacturers also insist) on using disk capacitors for this application is that they provide the best practical way to get just a capacitor, without at the same time getting an inductor and a resistor willy-nilly into the bargain. The last thing you need is to have all your little decoupling capacitors turn into little tank circuits scattered all over your c board.

The Kleffman boards I used, and some of the commercially-available boards such as Augat's (see again Figure 3) and Interdyne's, provide yet one more weapon in the battle against supply-voltage noise. A pair of supply-voltage planes are physically separated by only a very small thickness — an 0.004" mylar layer in the case of the Kleffman boards — so that there is in effect a distributed capacitor, sufficiently large to severely restrict the magnitude of the very-highest-frequency noise (say 150 MHz and up), between all points on these planes.

Motorola
type 10116
triple dif-
ferential
line receiver

Keeping Supply Voltages Smooth

The ever-present possibility of ac noise on the power supply voltages is probably the real reason for the general industry concern with power-supply-voltage margins in digital logic. It isn't hard today to build a fairly economical power supply with very tight regulation — 0.1% to 0.2% according to what one seasoned power-supply designer once told me. The non-trivial part is getting that precise voltage conveyed to each and every DIP. ECL, by the way, is normally specified as having a ±10% supply-voltage tolerance, and one major vendor (Fairchild) offers 10K logic with internal voltage compensation. In contrast, normal commercial-grade TTL is specified to tolerate just ±5% supply-voltage misbehavior.

Although I did not find it necessary to do this in my system, and I doubt that you will either, it is worth noting that an ECL system can straightforwardly be designed to present an invariant load to the power supply — in contrast to a TTL system, since some TTL gate packages may draw as much as 7 times as much supply current with all outputs low as with all outputs high, and thus full-word-complementing operations may result in high-frequency supply-voltage hiccups.

The world's fastest computer, the CRAY-1, capable of 138,000,000 floating-point arithmetic operations per second on a sustained basis, is designed according to this invariant-power-supply-load philosophy. (See reference 4, page 72, and also reference 5.) A number of rather extreme measures have been taken in the CRAY-1 to control various types of noise, for obvious reasons. The non-memory portions are designed largely with simple gates (Fairchild type 11C01) having "two-rail" outputs (the output and its complement, on separate leads), with both outputs terminated even if both are not used. In this configuration, each 11C01 presents an invariant load to the power supply. Single-rail output devices such as memory ICs use Thevenin termination. The 64-bit, 1,048,576-word CRAY-1 main memory is comprised entirely of 1Kx1 ECL memory ICs essentially similar (and actually bought to a longer access-time specification) to the ones I used. Of course, I only needed about 200 of them rather than 66,000 or so.

Off-Board Interconnection

Probably one could, at least in some cases, get away with running a properly-terminated signal line right off one board onto another if all of the precautions already discussed were taken. I never tried it. In the first place, one can't assume that the voltage-plane potentials on one board exactly match those on some other board the way they're supposed to, even if one has used 0.1-microfarad capacitors like popcorn as I have recommended. In the second place, there has to be some way to keep the signal lines at the same characteristic impedance, without discontinuities, as they leap through space between boards — and, worse yet, to keep them shielded from various forms of electromagnetic interference (such as each other) now that they are no longer safely close to a ground or other voltage plane.

Again, there is a simple, seemingly drastic, very effective way of solving the problem which pretty well decimates the goblins. As I just stated in the previous section, many ECL gates have two-rail outputs. (This may be an unfamiliar idea to TTL chauvinists; except for flipflops, one mux configuration, and a rather new and little-known two-rail buffer called the 74265, TTL devices don't usually offer this feature.) ECL gate circuit parameters are such that any two-rail gate can be used to drive a differential line, with all of the implied advantages of common-mode-noise rejection and insensitivity to temperature and dc-voltage discrepancies between different boards. Since such a line may in principle be as long as a few hundred feet before purely circuit-design-parameter alligators start snapping at one, there is no abrupt length limit of concern to a hobbyist. Of course, remember that a nanosecond is approximately a "light foot," and that electrons in a wire only travel about 2/3 as fast as light travels. Thus you may observe, at least if you can borrow a 485 scope for a while, that each 6" to 8" of signal line requires another nanosecond for the signal to traverse it, for differential as well as for single-ended signals. I was at one point rather startled to realize that some 15" signal lines were actually a bigger delay factor in one data path than a whole row of 2-nanosecond buffers with short signal lines coming and going.

At the other end of the differential line, on the other board, one uses a differential receiver element with a resistor between the two differential line ends. These elements come in four flavors: three types of triple elements with two-rail outputs (type 10116 for plain vanilla, 10114 for hysteresis, and 10216 for extra blazing speed), and one (type 10115) with quadruple elements with single-rail outputs.

For more details see reference 6, which also describes (on page 9) how to turn one of the triple two-rail devices into a Schmitt trigger circuit. Two cautions: First, any unused elements in a differential receiver DIP should be "strapped" to force their outputs into one logic state or the other, as otherwise they will hover right at the logic threshold point and the on-chip bias networks will get screwed up and confuse the elements which are being used. Also, we found that the resistor values suggested for use with the receiver elements by reference 6 were not the right ones for our interconnection system, and wound up using resistors

with values close to 300 ohms for all three
resistors shown in Figure 3 of that reference.

Cabling

There are probably other acceptable physi-
cal means for getting these differential signals
from one board to another, but the one I recom-
mend is flat ribbon cable, available in various
forms from 3M, Augat, Elco, Spectra-Strip, and
probably other companies. Specifically, what I
have used is 40-wire 3M cable, which is physi-
cally surprisingly small. Many such cables .
stack neatly in a small thickness, and fairly
abrupt turns and at least some limited hinge
action are possible. 3M supplies little press-
on connectors, and a tool to crunch them into
place, one at each end of the cable.

The electrical shielding properties of this
cable are <u>excellent</u> if one does not get greedy
about how many logic signals pass through a
single cable. There should be one or more
ground wires at both edges of the cable, and
alternating signal and ground wires within the
cable. This means that one can only transmit 9
differential logic signals in one 40-wire cable,
since doing just that according to the recipe
demands a minimum of 37 wires. The cable format
is, of course,

$$G \ G \ S_1 \ G \ \overline{S}_1 \ G \ S_2 \ G \ \overline{S}_2 \ G \ \ldots \ G \ S_9 \ G \ \overline{S}_9 \ G \ G \ G$$

When one stacks several such cables, the
signal-ground-signalbar-ground philosophy should
also prevail along the "z-axis," that is, in the
direction perpendicular to the plane of each
cable as one goes through successive cables.
Thus, in the cable immediately above the one
whose format has just been typed out, there
would be three edge G's on the left and two on
the right, and likewise in the one just below,
and so forth, so that the signal wires are
staggered.

Keeping it all Cool

If one firmly grasps a 1Kx1 ECL memory IC
(type 10415A/10146) after the computer has been
running for a few minutes, one can literally get
second-degree burns. This IC type dissipates
as much as 3/4 watt, and we measured ceramic
DIP case temperatures as high as 60° C. Up-down
counters (type 10136) and hex D-flipflops (type
10186) also run pretty hot, although not quite
that hot. 256x4 PROMs (type 10149), oddly
enough, run much cooler - about 45° C. The
<u>average</u> power dissipation for all ICs in the
entire midicomputer, including SSI and MSI types
as well as LSI types such as the 10149 and
10415A, is about 1/3 watt. Of course, probably
about 1/10 of that is dissipated as heat not
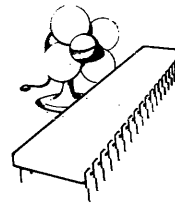
within the ICs themselves but within the
termination resistor DIPs.

Despite all that, we encountered few if any
problems attributable to heat. The ICs simply
sat out in the open, on large boards which were
mounted vertically like pages of a book on a
central vertical post, free to flop back and
forth through a small arc since they were
interconnected by ribbon cable as just described.
Although we had a forced-air-cooling scheme
figured out in case we needed it, we never had
to use it, and relied purely on convection and
radiant cooling. I also found that ECL ICs,
once installed and running properly, very rarely
died of natural causes, at least as compared
with TTL ICs in similar applications. Probably
the very high percentage (about 78) of voltage
plane on our boards helped a lot to conduct the
heat efficiently away from the ICs.

An ECL system installed within a closed
metal cabinet, particularly if the boards are
mounted horizontally, should doubtless be cooled
by some more active technique, such as forced
air. As for what the big-machine people do,
Control Data's big computers are Freon-cooled,
with lots of little pipes running along chassis
structural members. The CRAY-1 uses not only
Freon cooling but heavy-gage heat-conductive
copper sheets. (And, even though the CRAY-1
mainframe itself actually is physically small
enough to fit into your spare bedroom, the
auxiliary cooling apparatus might drive you out
of the rest of the house. Oh, well, you didn't
need quite that much speed anyway.) Some large
IBM computers use chilled-water cooling. You
may now note a delicate allusion to each of
these cooling techniques in Figure 1. Someday,
an ingenious hobbyist trying to cool a really
massive homebrew ECL system may wind up using
the refrigeration unit from a used Sears
Coldspot, but open-rack convection cooling or
forced air should do the trick for most systems.

There were a few Saturdays when we worked
on the Racal-Milgo system, and Plant Engineering
forgot to turn on the airconditioning until the
middle of the morning, and in Miami during the
summer an unventilated room is bad news for
people as well as for computers. The system
didn't run too well on those days until the
airconditioning had been turned on long enough
to pull the temperature in the lab down below,
say, 90° F. However, we really had no reason
to believe that it was the ECL which was giving

problems. We were using a semi-homebrew IM6100-based, PDP-8-compatible microcomputer to control the larger ECL machine, which is also a plausible technique for a hobbyist who already has one or two other microcomputers. Most of the system reliability problems which we were actually able to pin down turned out to be trouble with the 2102-type MOS memory ICs used with the IM6100, and the specific symptom of trouble we had on those hot Saturday mornings was usually inability to load the ECL midicomputer instruction memory from the IM6100. Nevertheless, if you do choose to rely on open-rack convection cooling, your ECL homebrew number-cruncher may run a bit better if you keep your spare bedroom at a temperature which you also find agreeable.

If Your Scope Isn't Fast Enough

ECL 10K output stages have a nominal logic swing from about -0.900 volts (considered to be a "logic 1" or "high") down to about -1.750 volts ("logic 0" or "low"), with the logic threshold being around -1.290 volts. (Aha! Now you know what these negative numbers really mean!) If you have followed the interconnection recipe of the preceding paragraphs faithfully, you should see picture-book square waveforms everywhere, although they do look just a bit cleaner at the end of a signal line close to the termination resistor than at points along the way. Even with a 485 scope, which shows every little wiggle, the ECL waveforms I observed looked very clean compared to the grassy ones sometimes seen in high-speed TTL systems.

If you must use a slower scope for money-type reasons, you will of course observe even cleaner waveforms (which aren't exactly real!) with slightly rounded corners, which may not deceive you very much about anything essential as long as you remember why it is that they look so clean. The principal danger is that, now and then, there will be a glitch on some signal line which is insufficiently wide to show up on the scope, or at least to look to you as if it is of sufficient magnitude to reach the logic threshold — when, all the while, here is an up-down counter (type 10136) or hex D-flip-flop (type 10186) or other edge-sensitive device whose clock input is connected to that signal line, which sure is acting as if it is getting an edge at just about that time. Probably it is, and you just can't see it because your scope has smoothed it out for you. Realizing that such must be the case, you have your choice of (a) getting hold of a more expensive scope, or (b) "reading between the lines" of what your humbler scope is telling you, terminating that clock line better, and seeing if the problem doesn't then go away. Or perhaps the clock-line glitch is really due to a switching hazard in the logic you designed to control that clock line.
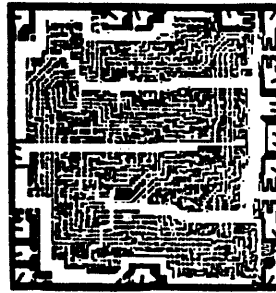
Motorola
type 10136
up-down counter

Product Families

To oversimplify things a bit in a manner meaningful to a ones-and-zeroes type like myself, ECL is one type of "current-mode logic." The state of an ECL gate is determined by which of the two output legs the main current is being steered through, and the resulting voltages at the output points are interesting side effects but are not the basic switching phenomenon.

In TTL and MOS, on the other hand, the output voltage states are where the action is, and the currents tag along after the voltages as interesting (and often inconvenient) side effects. In Schottky TTL logic, for instance, the voltage states you actually see on a scope are about +0.2 volts for a "logic 0" and +4.1 volts for a "logic 1." Incidentally, since Schottky TTL rise times are probably a bit faster than the 3.5 to 4-nanosecond times characteristic of ECL 10K, you may notice that the "voltage slew rate" or whatever that parameter should be called is many times greater for Schottky TTL, and is in fact roughly equal to the corresponding rate for the very fastest ECL families.

There are lots of custom families made by IC manufacturers for direct sale to large computer companies which fall into the general category of "current-mode logic." However, most these are not available for sale to the general public. There are six product families, more specifically considered to be ECL, which are sold to all comers:

o ECL I, contemporary with DTL and now obsolete and not used in new designs.

o ECL II, contemporary with and somewhat philosophically akin to H-series TTL.

o ECL III, extremely fast, but with only a modest selection of SSI and MSI types.

o ECL 10K, the only one which I am claiming is well-adapted to homebrew usage.

o ECL 95K, philosophically much like 10K, but a marketplace also-ran.

o ECL 100K, the fastest one of all.

The first four of the above families were introduced by Motorola, and the last two by Fairchild. Fairchild, Motorola, and Plessey also have various other ECL products which are not really organized into families — for instance, Fairchild's 11C01 OR/NOR gate (used in the CRAY-1), which is an 100K-technology device offered in a non-100K package. The last four families are, or can be made to be, electrically compatible so that with some care devices may be mixed in a system.

Except for ECL 10K, the second-sourcing picture isn't too bright. ECL III is available from Motorola (which of course calls it MECL III) and also from Plessey (which calls it PECL III). Signetics, or its French affiliate which is also owned by Philips of the Netherlands, is moving towards becoming a second source for ECL 100K.

ECL 10K, however, has a large number of viable second sources. Besides Motorola, Fairchild, Plessey, and Signetics make virtually the complete line, Nippon Electric makes some of it, and Fujitsu, Monolithic Memories, and Texas Instruments make 10K-compatible bipolar memories. This list covers only those firms which offer their wares in the United States today through distributors. There are other firms which sell only in Japan or in Europe, where ECL 10K probably has a larger share of the total digital logic market than it does here. It was a British computer manufacturer, International Computers Ltd., which originally sponsored Motorola's development of ECL 10K.

Mundane Details

As far as one can tell by scanning current Fairchild and Motorola OEM price lists, ECL 10K ICs in plastic packages tend to cost about 1½ times as much as the closest equivalent part in low-power Schottky TTL. There are certain items which don't follow that rule and cost relatively more, such as the type 10136 up-down counter. However, most ECL 10K gate parts in plastic today cost well under a dollar even in unit quantities.

Some of the ECL 10K LSI parts are still up in the $10 to $30 range, such as the Motorola 10800 which is a rough equivalent to an AMD 2901 at least when equipped with external RAM. The cure for that situation is of course second-sourcing, which is happening but takes time. Sooner or later Fairchild will probably be making the 10800, and then the price will drop.

Almost all ECL 10K parts are now available in full military-temperature-range versions, in case you are taking your homebrew system with you to Montana this winter or to Phoenix next summer.

The Even Faster Stuff

A hobbyist willing to hand-solder, rather than wirewrap, all interconnections to that part of his system might with due care succeed in making some limited use of ECL III and/or ECL 100K. (A few turns of wrapped wire, it turns out, functions all too well as an inductor when hit with the 900 or 700-nanosecond edges respectively characteristic of those families, and the resulting impedance discontinuities make reflections.) The technology required to build a system of any size using one of these families, however, remains pretty difficult at the present time.

There are, however, two ECL III devices of some interest to a hobbyist: type 1648, which is called a "voltage-controlled oscillator," and type 1658, which is called a "voltage-controlled multivibrator." There is no announced ECL 10K product matching either of these descriptions, and they are useful. I used a 1648 once in an otherwise all-TTL system to provide a hand-adjustable system clock source for testing clock margin - that is, how fast the system could be made to go before it failed. (In this case, I didn't personally do the nitty-gritty.) It worked fine, after being well shielded.

Those ECL 10K parts having type numbers of form 102XX (106XX in military temperature) form a subfamily, with appreciably different properties. Hand-soldered connections are also advisable when using any of these. They are quite a lot faster than their normal ECL 10K equivalents, and consume essentially no more power, so they sound like a super-good deal. Alas, this greater speed has been obtained by cutting very short the leisurely rise and fall times deliberately designed into normal ECL 10K. To oversimplify a bit, normal ECL 10K rise and fall times are perhaps 3.5 to 4 nanoseconds — much longer than the nominal logic delay of 2 nanoseconds or so; whereas 102XX rise and fall times are probably not much different from the logic delay which I have observed to be about 1.25 nanoseconds. Reluctantly, I concluded that system noise problems would be minimized by restricting the use of 102XX parts to those situations where that last ounce of speed is really required, for instance in the clock generation circuits. The only sure-enough example of proven crosstalk trouble in the Racal-Milgo midicomputer was due to an unnecessarily long wire being driven by a type 10212 buffer, and was eliminated by relocating a few ICs so that that buffer was closer to its load.

Motorola
type 10211
NOR buffer

One other simple and plausible usage of ECL III or ECL 100K is to build high-speed, fixed-frequency oscillators. Two type 1688 OR/NOR gates on the same IC in series make a dandy oscillator (see Figure 5), which we observed to generate various frequencies from 110 MHz to 170 MHz depending on which particular MECL or PECL sample we were using. Probably an 11C01 would produce a somewhat higher frequency. Don't try to build this type of oscillator with a single gate — it won't even oscillate, but will just hang in there with its output at about the threshold voltage. If swapping ICs around doesn't get you the frequency you are shooting for, try hanging <u>very</u> small capacitors on gate outputs to slow them down. Caution: with ECL III at least, the speed of oscillation may depend on the ambient temperature.

Logic Drawing Conventions

If one wishes to think "assertive-high" — that is, the more positive of the two output voltage states represents a "1" and the more negative state represents a "0" — then it follows that in TTL logic the simplest and most natural gate structure is the NAND gate, whereas in ECL logic it turns out to be an OR/NOR gate. It is fairly clearcut in both cases what is really the simplest circuit for the silicon people to build.

Since the ORing together of minterms seems to be more natural to human psychology than the ANDing together of maxterms, part of learning to use ECL consists of learning to use "mixed-logic" conventions in some form. These conventions allow you to think of — and draw — a given physical gate circuit as performing either an OR function or an AND function, and then to consider as a separate issue the assertiveness of that gate's input and output signals.

Soaking up the mixed-logic viewpoint <u>should</u> also be part of learning to design with TTL, but unfortunately it isn't always. In fact, many erudite polemics have been written defending older and less general viewpoints, which I feel are now best understood as evolutionary way stations on the way to the full-blown mixed-logic viewpoint. One of these older viewpoints is the one which results in the dozens of busy little black and white triangular flag symbols on the inputs and outputs of ECL 10K parts as drawn on Motorola's data sheets.

I can only say that, once I forced myself to give up the particular Bronze-Age viewpoint I formerly had and learned to use mixed-logic conventions, within a week I was really wondering why I had ever used anything else. If one uses an additional logic symbol to denote "psychological inversion," implying that although

the electrical polarity of some signal (say, for example, BANANAS) hasn't changed one's perception of its meaning or "psychological polarity" has changed (in the example, to YES WE HAVE NO BANANAS), then logic drawings can be made almost as semantically precise and self-checking as logic equations. This is not an academic exercise — it helps you spot real mistakes before you wire it up wrong and waste a lot of time trying to figure out why it isn't working.

Reference 7 is an eminently sane paper on this whole dogma-ridden subject; I have relied on it for several years for guidance as to logic drawing conventions. Recently there have been more papers in the same vein. I have just one minor quibble with reference 7; the symbol suggested there for psychological inversion, a small line drawn across the signal line at right angles, tends not to show up too well on blue-line copies. A little solid triangle or arrow-head drawn next to the signal line shows up much better, and there is a hole of the right shape on most templates.

Figure 6 shows the same physical gate element, one of the two-input elements from a type 10105 triple OR/NOR gate part, drawn first with assertive-high inputs as an OR/NOR gate and second with assertive-low inputs as an AND/NAND gate. You get the idea. The pins are all still in the same relative positions. If one uses the standard "inversion bubbles" correctly, one doesn't really need the additional symbols such as triangular flags - they in fact introduce one too many degrees of freedom and just confuse the issue. I suggest the proper use of mixed-logic conventions as a sort of "software adjunct" to the rest of my homebrew ECL recipe.
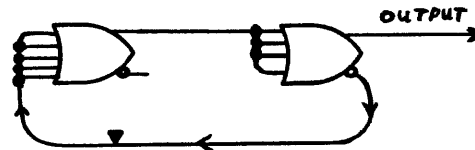
Figure 5. Fixed-frequency oscillator.
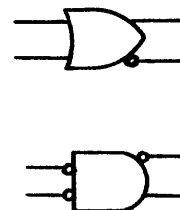
Figure 6. Two representations of the same gate.

Finale

I have used references sparingly, because many of them tend to give ECL system design an air of awesome complexity and desperate peril, and this is exactly what I am trying to tell you ain't necessarily so. However, if you get seriously into ECL you probably will want to get hold of Motorola's whole set of application notes pertaining to ECL 10K, and also some worth-while material published by Fairchild and Signetics. Good luck, and may the Force be with you. #

Motorola
type 10105
triple OR/NOR gate

Humble Thanks

There are several people who were at Racal-Milgo when I was, without whose efforts the number-cruncher would never have happened and I wouldn't have been able to tell you about it. Two particularly deserve mention: Dick Joerger is the "sharp technician" referred to in the text of this paper, and built most of the machine with his own two hands. Rick Johnston did the preliminary design of the control section and improved it greatly from what I had originally planned; he made it a pipelined, overlapped, stored-microprogram machine with a self-restarting main timing circuit.

There are also three electronics industry veterans at other companies with whom I have had numerous conversations, usually by WATS line, and without whose wise opinions I would not have had the temerity to try to build the machine out of ECL. They are: Stan Bruederle of Signetics, Rob Walker of Intel (at that time still at Fairchild), and Norm Winningstad of Floating Point Systems. Everything each of them told me about ECL turned out to be absolutely correct, and much of it appears in the text of this paper. If the paper is wildly off-base on any topic, it's because I didn't get their complete message.

Ad Hominem

Chuck Hastings has spent two decades in the computer mainframe business, mostly as an architect/designer, and currently works for Itek Applied Technology in Sunnyvale, CA. He has previously worked full-time at Racal-Milgo, two subsidiaries of United Telecom, Control Data, Honeywell, and TRW, and has been a consultant to various big and little companies

in four states. Besides the computer described in this presentation, he earlier managed the technical end of a hardware project in which the Control Data 3300 virtual-memory computer architecture was successfully emulated using TTL MSI. He has a BA in physics and math from Grinnell College, an MA in math from UCLA, and many additional EE and computer courses at the University of Minnesota, and has embarked on the MBA program at the University of Santa Clara. He has two patents in electro-optic mass memories, belongs to ACM and IEEE, and has five children and three station wagons.

References

1. "Notes on Microcomputer Music," Marc LeBrun, pages 128-130, The First West Coast Computer Faire Conference Proceedings, Box 1579, Palo Alto, CA 94302; 4/1977 — & John Reykjalin, private communication; 1/1978.

2. The ECL Handbook, Fairchild Semiconductor, 464 Ellis Street, Mountain View, CA 94042; 7/1974.

3. "Packaging High Speed ECL Integrated Circuits," Leonard A. Doucet, Augat Inc., 33 Perry Avenue, Attleboro, MA 02703; about 1974.

4. "The CRAY-1 Computer System," Richard M. Russell, pages 63-72, Communications of the ACM; 1/1978.

5. "CRAY-1; The Smaller Supercomputer," New Products department on page 53, Computer (IEEE Computer Society magazine); 3/1976.

6. Interfacing with MECL 10,000 Integrated Circuits, Application Note AN-720, Motorola Semiconductor Products, Inc., Box 20912, Phoenix, AZ 85036; 1974. Author of this note is Bill Blood.

7. "Mixed Logic; A Tool for Design Simplification," Paul M. Kintner, pages 55-60, Computer Design; 8/1971.

+ + + + + + + + + + + + +

APPENDIX

About this List

I promised you a list of vendors of various items which you will need. Here they are, with names and addresses, in alphabetical order by topic. I don't, of course, imply any warranty

that you will find them utterly perfect about everything, simply by listing them here, but I have had positive dealings with most of them.

ECL Logic and Memories

Fairchild Semiconductor
464 Ellis Street
Mountain View, CA 94042

Motorola Semiconductor Products, Inc.
Box 20912, Phoenix, AZ 85036

Plessey Semiconductors
1674G McGaw Avenue
Santa Ana, CA 92715
(Head office: Swindon, U.K.)

Signetics Corporation
811 East Arques Avenue
Sunnyvale, CA 94086

(No direct information)
Nippon Electric Company, Japan

ECL Memories Only

Fujitsu America Inc.
2945 Oakmead Village Court
Santa Clara, CA 95051

Monolithic Memories Inc.
1165 East Arques Avenue
Sunnyvale, CA 94086

Texas Instruments Inc.
P. O. Box 5012, MS 308
Dallas, Texas 75222

ECL-Grade Logic Breadboards

Augat Inc.
33 Perry Avenue
Attleboro, MA 02703

Excel Products Company, Inc.
401 Joyce Kilmer Avenue
New Brunswick, NJ 08903

Garry Manufacturing Company
1010 Jersey Avenue
New Brunswick, NJ 08902

Interdyne, Inc.
14761 Califa Street
Van Nuys, CA 91411

Mupac Corporation
646 Summer Street
Brockton, MA 02402

Stanford Applied Engineering Inc. (SAE)
340 Martin Avenue
Santa Clara, CA 95050

Flat Ribbon Cable

Augat (see above)

3M Company, Industrial Electrical Products
3M Center, St. Paul, MN 55101

Elco Corporation
2250 Park Place
El Segundo, CA 90245

Eltra Spectra-Strip
7100 Lampson Avenue
Garden Grove, CA 92642

Ceramic Disk Capacitors

AVX Ceramics
P. O. Box 867
Myrtle Beach, SC 29577

Centre Engineering
2820 East College Avenue
State College, PA 16801

Resistor Packages

Beckman Instruments, Inc.
2500 Harbor Boulevard
Fullerton, CA 92634

Bourns, Inc.
1200 Columbia Avenue
Riverside, CA 92507

ILC Data Device Corporation
Airport International Plaza
Bohemia, Long Island, NY 11716

Wirewrap Equipment

Gardner Denver Company
1333 Fulton Street
Grand Haven, MI 49417

OK Machine and Tool Corporation
3455 Conner Street
Bronx, NY 10475
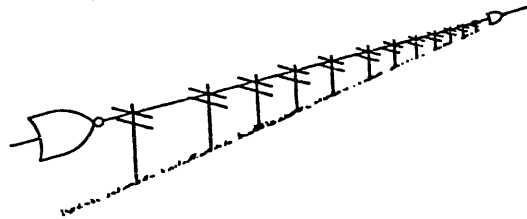
Ed Schoell, B Tech (Electronics), VK3BDS

1. ABSTRACT

This paper describes a microcomputer system based on the National Semiconductor Pace (INS8900) 16-bit microprocessor. This system has been developed by the author in Australia to fill the gap between evaluation kits offered by microprocessor chip manufacturers and the full sized microcomputers available from USA sources.

The use of a 16-bit processor has a number of advantages for hobby applications, especially where any maths work is contemplated, and at $19.95 for the CPU chip the provision of a powerful instruction set and minicomputer architecture make it very attractive.

The system described consists of a family of boards 6.25 by 8.5 inches with a single sided 85-way edge connector. These boards plug into a mother board in a case and power supply combination box to form a microcomputer. However the CPU card can be run on its own by connecting it to a power supply and TTY or terminal thus providing a low cost introduction to the system without limiting later expansion as is usually the case when a person tries to expland an 'evaluation kit'.

Ed Schoell, Box 30., Boronia 3155., Victoria, Australia.

The CPU card, a multi-purpose interface card and an 80x24/28 video card are completed and a 16Kx16 memory card and a floppy disc interface cards are planned in the near future.

2. INTRODUCTION:

By way of background, the hobby microcomputer scene in Australia has been dominated by the two hobby magazines, Electronics Australia and Electronics Today International. Both these magazines have published projects based on evaluation kits from the microprocessor chip manufacturers. Both have published simple VDU TV-typewriter systems, (of which quite a few hundred have been constructed) and both have reviewed offerings from importers of USA originating systems from Altair etc.

Computer stores have opened in the major cities, and a number of active computer clubs have started with membership in the 100 to 200.

One popular project has been a SC/MP based system call "MINISCAMP" using a switch and LED simple control panel to enter data into RAM and execute a program. Some 800 odd systems have been built of these and it has proved an attractive learning tool priced under $100.

Fig. 1. PACE (INS8900) INTERNAL ARCHITECTURE

Data and addresses are connected to the outside world by the 16-bit data bus at the top right of the diagram. Stobes NADS,IDS,ODS control bus activity.

Because of the cost and difficulty of importing USA equipment into Australia and the lack of locally designed equivalents, about two years ago design was started on a system using the SC/MP microprocessor as the controller for an intelligent VDU. The original intention was just to build up a system for personal use, but discussion with a lot of interested other people convinced one of the need to design reproductable printed circuit boards and to offer these as kits for others to build as well.

The video system was prototyped first as a wire-wrapped conglomeration across the bench, hooked up to a SC/MP low cost development system from National and this worked very well.

About this time National announced a price reduction on the PACE cpu chip down to $25 for a 3 microsecond version of the P-channel part, and it was too great a temptation to resist putting a much more powerful micro chip into the system and releasing it as a microcomputer with a video front panel (although still usable as a terminal with other systems).

Having made this decision the long hard task of detailed interface design, CPU design and PCB layout began and occupied some six months of spare time activity. The first results have been gratifying as we now have a unique combination of a 16-bit CPU with a powerful debug program and easy expandability, a multichannel interface card able to drive a combination of parallel and serial I/O devices and a video card with 4K of character storage in a variety of formats, plus a flexible graphics system giving 160 by 140 points able to be mixed with upper or lower case text. Let us now look at the design in more detail, starting with the CPU card.

As mentioned before the change to Pace was made when National released the $25 3 microsecond chip. Since then a $19.95 two microsecond N-channel chip the INS8900 has been released and this was used for the released printed circuit boards. It offers a number of advantages, in that the clock system is now a single CMOS line and bus interfacing can now be done at low-power schottky levels via the 8-bit wide INS8208(DP8304) transceivers.

3. CPU DESCRIPTION

On the CPU board a crystal oscillator at 11 MHZ provides timing for the CPU after division and this clock is also buffered off the card for use by the video system for timing the display and by the interface card for the programmable baud rate generators. The Pace internal architecture is very minicomputer like (actually rather similar to the Data-General NOVA). It has four 16-bit accumulators, all of which can be used in memory reference instructions, although accumulator zero is used as the prime data accumulator, because of the availability of branch on condition instructions which can test data or individual bits of this accumulator as well, a group of logical memory reference instructions use this accumulator (in these respects the pace instruction set is a considerable enhancement of the NOVA's).

Accumulator 2 and 3, as well as being able to be used as destinations and sources for 16-bit arithmetic operations, can also be used as 16-bit index registers. All memory reference instructions (with the exception of the branch on condition which is PC-relative) can use either of these for indexing, alternatively, memory reference instructions can go PC relative or into the first 256 words as a base page. Indirect instructions are also provided.

Figure 1 gives the internal structure of the CPU.

The 16-bit I/O bus is provided to couple the PACE to the outside world, and data in and out as well as addresses are placed on the bus. Control strobe lines (NADS, IDS, ODS) indicate what is happening on the bus, and in the case of this CPU card, are used to latch address data into a pair of DP8212 8-bit latches giving a latch address bus pined out from pins 20 to 35 on the card edge. The buffered data bus is available on the card edge pins 40 through 55 and is turned around by the IDS (input data strobe) signal. An additional pair of buffers on the card buffers the on card memory onto the system bus when memory in the range is decoded.

A variety of memory is provided on this card. Firstly, 6 PROM sockets for 2708 (or 2716) PROMS give 3K (or 6K) by 16 of firmware storage. One pair is used for the DEBUG program described later. The PROMS are normally located in high memory, but two pairs can be moved to location zero (the initialise address). As well, optional automatic power up vectoring to the monitor is switch selectable.

Because the DEBUG is out of the way of low memory RAM can be located there, with no restrictions at all. In fact address space for the bottom 52K (x16) is unassigned, and slots above this are provided for VIDEO RAM (4K), Disc Operating System, standard peripherals and the ROM mentioned above.

A block of 256 words (x16) of RAM is located at the very top of memory. Some of the lower half of this is used by the monitor, but most is available for users. It resides at the split base page if this is enabled.

A second block of 4K RAM can be provided by plugging in 16, 4Kx1 static RAMs. This can be allocated anywhere in memory by DIP switch, and provides enough capacity on the single card to run the editor, and assembler, as well as a TINY BASIC compiler planned for release later this year.

This card can be used stand-alone, and TTY/20mA/ RS232C interface allows for bit-serial communications with ASCII and BAUDUT devices at 50, 110, 300 and 1200 BAUD. The flags (F13,F14/and JC15 are used for this simple I/O.

A 16-pin DIP connector is also provided on this card to convert to a range of low-cost microprocessor peripherals developed in Australia on a ribbon cable interface. These include PROM programmers for the 2708Q and 5204Q, LED displays, and calculator-style control panels.

4. Multi-Interface Card:

To expand the system over the "flag-waving" serial I/O on the CPU card, a second card adds multiple serial and parallel interfaces. One (or optionally 2) USARTS are provided, giving synchronous or asynchronous serial I/O at programmable, split or externally controlled baud rates. A three-channel 16-bit interval timer chip is used for baud rate generation from the crystal clock, so precise clocks can be generated for all standard rates as well as the odd BAUDOT ones used by radio hams etc. Channels not used for baud rate clocks can be used for real time clock or external counter applications in binary or decimal.

Serial I/O is at TTL, 20 mA or RS232C with modem control lines provided.

To provide low cost backup storage, it is hard to beat an audio cassette recorder, so an interface in BYTE/KANSAS CITY format is provided, along with remote control for two tape recorders, and a higher speed option as well. A high speed synchronous port is also provided to convert to a low cost Australian developed 3M cartridge drive

system. Various parallel interfaces are provided to allow input from a high speed paper tape reader and an output for a Centronics or similar style printer. Strobe and control polarities are link programmed to allow for the usual odd collection of peripherals hobbyists and schools accumulate!

Rather than provide seperate cards for such a range of I/O devices, it was easier and much cheaper to combine them all onto one card. In this way, address decoding, bus buffering and I/O control flags are shared rather than being duplicated as most minicomputers do it, and each interface costs only very few dollars.

5. VIDEO INTERFACE

A third plug-in card adds full intelligent VDU facilities to the system. On this card, 4096 bytes of RAM are scanned by the displays logic to produce an 80 character by 28 line video output suitable for feeding to a monitor or modified TV set. The CPU has direct access to the screen memory, so data can be output to the screen very rapidly without waiting for a serial transfer, and can, of course, go anywhere it likes, reading or writing.

The RAM is isolated from the CPU via multiplexers, although the RAM appears in the CPU memory space as a 4K block. Several video cards can be supported in a multi user environment, as memory management enable/disable logic is provided.

Care has been taken to blank the video during CPU access, thus reducing interference and "flashing" on the screen. Scrolling is handled at high speed (in seperate blocks of 16 and 12 lines) by the CPU simply writing a screen offset constant into a top or bottom screen latch.

Each character position on the screen can be addressed in a graphics mode as five horizontal lines the width of the character. A plug-in option expands the video word to 12-bits (easy to handle with a 16-bit micro) spliting each bar vertically to give an overall graphics resolution of 160 horizontal by 240 vertical square dots able to be mixed with upper or lower case alphanumerics.

Optional data formats are provided, allowing for 64 characters (by 64, scrolled) or for 40 double-width characters (with lower video bandwidth).

(A link provides for 625 x 50 or 525 x 60 video standards, but on 525, only 25 lines can be displayed, rather than 28.)

6. SOFTWARE

Quite a lot of development time have gone into the software provided in the two monitor PROMS - and quite a few years experience in writing and using debug programs has provided the background. The usual commands to type and alter memory, registers and stack data are provided with enhancements to allow reference to memory addresses not only asa hex number, but as an arithmetic expression (e.g. an offset can be added to allow for relocated code) or as an indexed expression via either of the index registers.

Also a leading sign can imply the length of the typing range, without needing a full definition (e.g. TM -3(2)/+6 types 6 memory location onwards from 3 locations below the address pointed to by index register 2).

Automatic calculation of displacement for PC relative instruction is provided when using the Alter command (e.g. AM 4003, C1(3F92) alter memory location 4003 to C18F the PC relative negative displacement of 8E being generated automatically). One bonus with the 16-bit wide instructions is that

PC relative (i.e. 2-byte) instructions can reach 2 or 3 times further than the 8-bit micro's. Also instructions are constant lengths, a bonus for hand-assembly.

The monitor/debug has dump and load facilities in hex and binary onto paper tape, audio cassette, and "down-line" to and from other computers when the system uses a PACE cross-assembler, or acts as a cross-assembler host machine to SC/MP, PACE or 8080 external systems.

One area most monitor/debugs fall down on is facilities provided for program checkout. Most usually only provide a break-point (or two) if you are lucky. At National (in Australia) we developed the single-step debug now used in the PACE Low Cost Development System from NSC. This has been expanded in this system to give options of program stepping, printing (or displaying) each step (with or without registers), or with registers which altered with execution highlighted by a reverse video field. Optionally, the video display can either show the last step or scroll up at each step or each branch of program. The single step facility makes use of the level zero non-maskable interrupt provided in the CPU.

All input-output for system software (text editor, reloading loader macro-assemblers, BASIC, etc) goes via the monitor. I/O is defined at startup and, can be changed dynamically between drivers provided in PROM or user-supplied I/O routines called via link addresses examined as I/O routines are called. This allows a user to use video as the system console, but cause the hard copy of an assembler output to go to a Baudot or Selectric printer (Baudot code driver provided, Selectric user loaded). Alternatively, a Baudot machine can be used as system console (unique!), but allow programs to write ASCII hex or binary data to cassette.

7. CONCLUSION

This has been a brief introduction to the system. It has caused a fair amount of interest in Australia, and as the 16-bit field has been neglected on a world-wide basis, it is hoped that there will be interest in it from further afield.

8. REFERENCE

INS8900 single chip 16-bit n-channel microcomputer data sheet.

Oct '77, National Semiconductor Corp.

9. AUTHOR BACKGROUND.

The author has been working in digital design for 12 years, initially with Hawker Siddeley and since 1972 as an applications engineer with National Semiconductor. This system is a private development, and kits and cards are available from JED MICROPROCESSORS PO Box 30 ,BORONIA 3155 ,VICTORIA,AUSTRALIA.

A1. PACE (INS8900) CPU CARD.

This photo shows the CPU chip, the address decoding
and the level zero interrupt logic (for single stepping)
with the 4K-RAM, 3K-ROM X16 memory. TTY and RS232
interfaces allow this card to be used stand-alone.
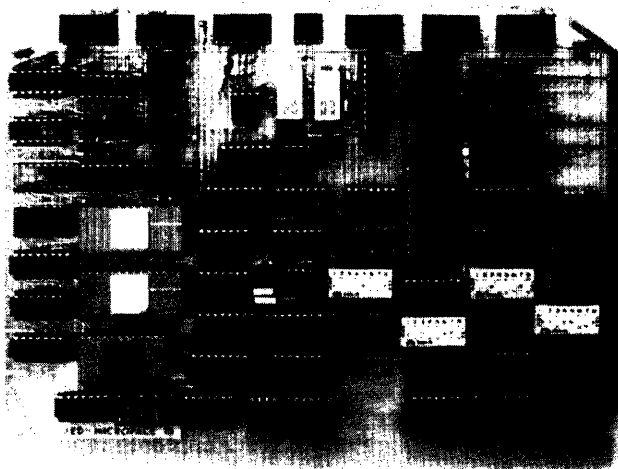
A2. MULTI-INTERFACE CARD FOR PACE CPU.

Connectors across the top of this card connect to
RS232 20 milliamps, 3M cartridge, Kansas City
audio cassette, paper tape reader, paper tape
punch and parallel printer systems. Two
USARTS on this card provide serial interfaces and
DP8212 and other logic chips handle parallel I/O.

A3. VIDEO INTERFACE FOR PACE CPU.

This card, with its four K by twelve bit RAM
provides video interface to a TV set. Formats
provided are 28 lines of 40 or 80 characters
and an alternative of 28 lines by 64 characters
scrollable through 64 lines. 525 and 625
standards are link selectable. Graphics on a
160 by 140 matrix are also provided.

MICROPROCESSOR INTERFACING TECHNIQUES

Rodnay Zaks, President and Austin Lesea, Senior Engineer, Lecturer
Sybex, Inc., 2161 Shattuck Avenue, Berkeley, California 94704, 415/848-8233

No longer is interfacing an art which requires years of experience and hundreds of integrated circuits. With the introduction of the microprocessor and other LSI chips, interfacing most peripherals becomes a technique of applying the proper LSI chips.

Any interfacing requirement involving less than 20,000 bytes per second transfer rate can be solved by one chip. Many other tasks can be solved by the new controller chips, or a three or four chip microcomputer.

Presented here are: the basic input-output interface chips for simple serial and parallel interfacing, the advanced protocal controllers, two device A/D, D/A modules for real world interface problems.

Included here are:

-Basic serial and parallel input-output IC's.

-Specialized device controllers.

-One chip interfaces.

-New D/A, A/D products to simplify "real world" interfacing.

Basic IO

Typically, latches are used to store output values, and buffers are used to input values. Such specialized SSI circuitry has been improved greatly by providing program-mability.

A programmable paralleled interface pro-vides not only the input drivers and output latches, but also the interrupt circuitry, direction selection, and handshaking signals usually required. An ideal parallel interface chip is shown in Fig. A-1. The companies coming closest are Zilog, Motorola, and Intel with the Z-80 PIO, 6820 PIA, and 8255 PIA, and 8255 PPI, respectively. Now the basic microprocessor board is universal, as the I/O can be allocated by software depending on the application. Hardware is the same in almost all cases. Only the software changes.

Serial interfacing has long been integrated into the classic UART or Universal Asynchronous Receiver-Transmitter. Now we have the appear-ance of even more useful serial interface chips. The new UART provides asynchronous or synchronous communications capability as well as modem control features. Again, interrupt circuitry, and a variety of program-mable character lengths and rates are desir-able. The authors' ideal USART appears in Fig.A- 2. Again everyone has such an element in one form or another. Examples are the TI 9901 and 9903, Intel 8251, Motorola 6850, and Zilog 2-SIO.

Special Device Controllers

The UART was first put in LSI form because it was a basic building block required by most systems. The same is now true of the circuitry required for printers, CRT's, floppy disks, cassettes, and other common peripherals.

For your favorite printer, there is an interface circuit that will connect it to your favorite microprocessor. All the hardware has been done for you except for possible motor drive circuits or relay drive circuits.

The most difficult and disagreeable inter-face task of all, that of floppy disk interfac-ing, has now been solved on one chip! The IBM dual density 3740 compatible format is built into the Western Digital 1781 interface chip. The only external circuitry required the data separator and line drivers to the floppy. Such parameters as stepping time, head settling time, block size, etc. are programmable. One chip then will work with most of the available floppies and mini floppies.from all the manu-facturers. The 1781 is illustrated in Fig. A-3.

One Chip Interfaces

With the introduction of single chip micro-computers such as the Intel 8040 family and Texas Instrument 9940 family, and the Mostek one chip F-8, we can now program our interface problems.

We use the one chip processor as a dedicat-ed interface element in a larger system. This is shown in Fig. A-4. As long as the processor has enough time to handle the peripheral data rate requirements, we have the perfect general purpose interface.

Intel has announced the 8741 or UPI. This is a programmable interface element which is a special microcode version of the 8048 family. It is meant to be used as the peripheral inter-face element in the system. The 8741 is the EPROM version which can be user programmed and erased.

In the future, no special purpose inter-
faces will be designed, they will all be mask
programmed parts as ROM's are used today for
software packages. We will have BASIC ROM's,
FORTRAN ROM's, and Shugart UPI's, Pertec UPI's,
OKIDATA PRC's, etc. Not only will we have
"plastic software", but "plastic interfaces"
as well.

A/D and D/A

So far we are unable to measure voltages
and currents from physical sensors. This is
the "real world" of industrial applications.
Designing A/D (analog to digital) and D/A
(digital to analog) converters has long been
a tedious process. Once done with the design,
they were not easily interfaced to our standard
microprocessors.

Now we have the ADC 0816 from National.
A CMOS part costing $20.00 in quantities, it
provides 16 multiplexed analog inputs with an
8 bit digital three-state output. It replaces
previous modules costing upwards of $200.00.
In the same way that the ADC 0816 gets the
information in, other monolithic products are
available to do the D/A conversion.

In addition to A/D's, there are other parts,
such as; the Signetics NE5018. It is a complete
8 bit D/A on a chip. It has a built-in
reference and input latch so that no external
components are required for interfacing, and
costs $8.00 in large quantities.

Conclusion

The new LSI components have liberated
the designer to spend more energy on improving
utility and performance. No longer are inter-
faces "dumb", but they actually reduce the
amount of processing required by the use of
distributed intelligence. We now have the
"plastic software" interface, where other than
buffering or level shifting, all interfacing
is done in software. This is the ideal situa-
tion as hardware now becomes standardized, and
the only real cost becomes programming.

This material is a short summary of the
concepts presented in "Microprocessor Inter-
facing Techniques", by Austin Lesea and Rodnay
Zaks available through Sybex, Inc., Berkeley,
California 94704, telephone: 415/848-8233.
Also available in most computer stores.

Typical PIO

A - 1

UART Block Diagram
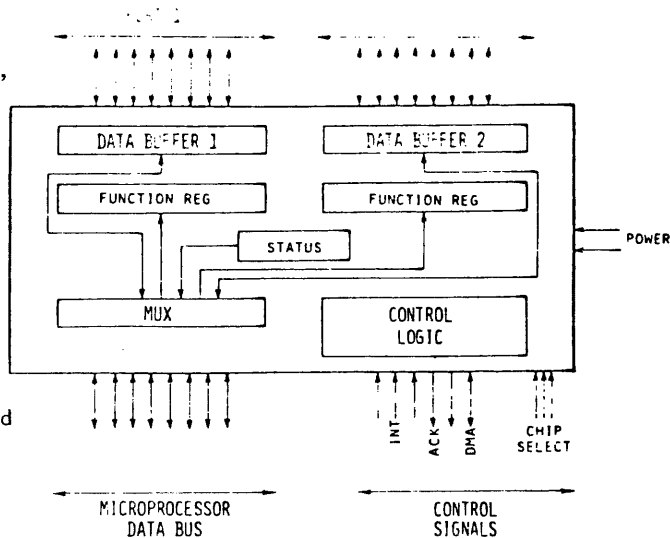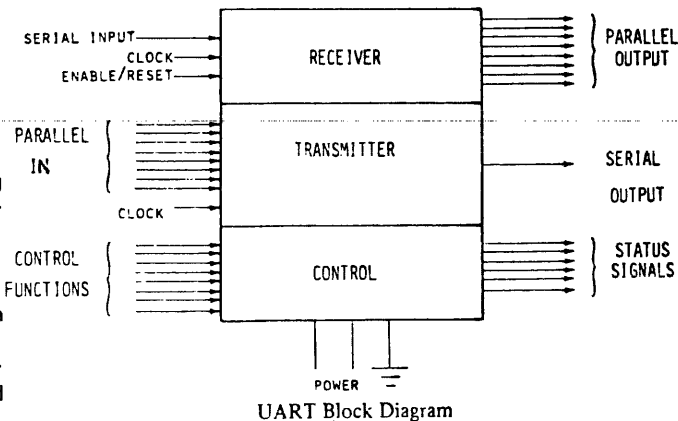
A - 2
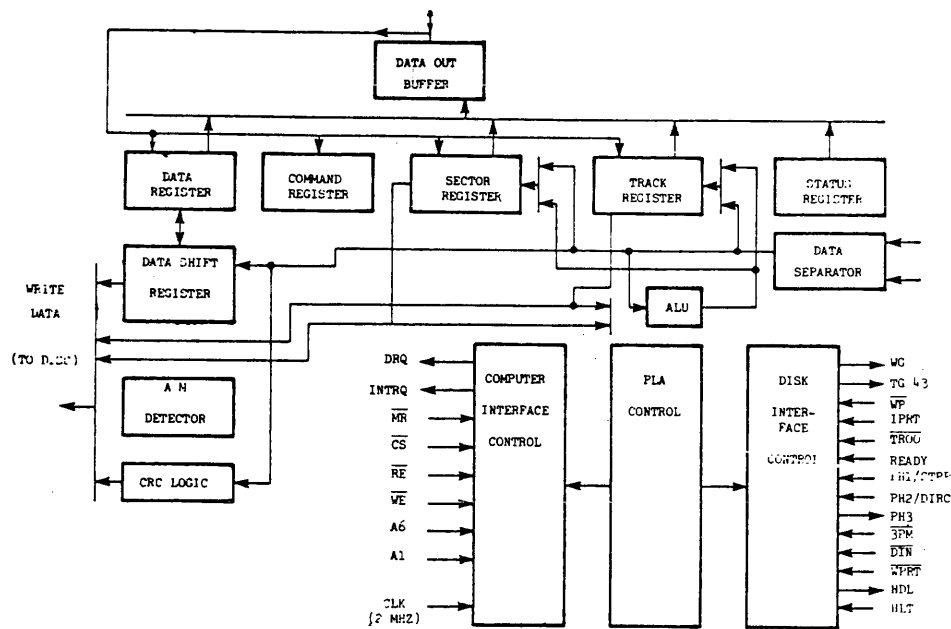
A - 3
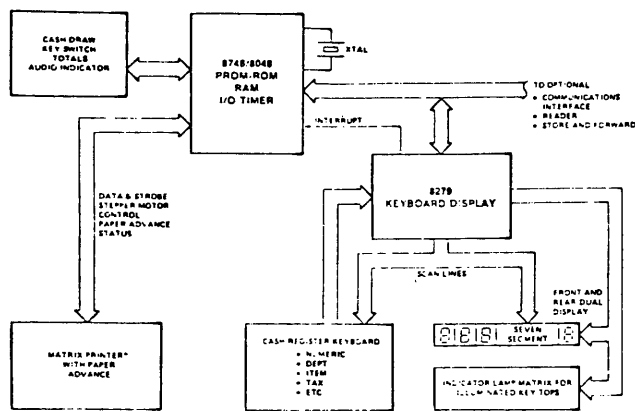
8048 Point of Sale Terminal with 8279

A - 4

TESTING FOR OVERHEATING IN PERSONAL COMPUTERS

Peter S. Merrill, Consulting Engineer
1460 Diamond Street
San Diego, California 92109

Abstract

Overheating in personal computers can be determined inexpensively with temperature indicating lacquers. Permissible chip junction temperatures and corresponding lacquer temperatures are discussed. Results of tests on the Commodore PET 2001 Series 8K computer are presented.

Introduction

Excessive temperatures can cause errors, intermittent operation, and can shorten the operating life of semiconductor components in personal computers. The existence of overheating can be estimated easily with inexpensive temperature indicating lacquer which has an opaque dull color when a drop is placed on the surface of a component. The lacquer changes color irreversibly or turns clear when the indicating temperature is reached or exceeded. A variety of lacquers are available (see Table A-3) with indicating temperatures throughout the range applicable to personal computers. The lacquer can be removed with 99% isopropyl alcohol. Stick-on labels are available also, but these are generally more expensive to use than lacquers.

Estimating Semiconductor Junction Temperatures from Case Hot Spot Temperatures

The important semiconductor component temperature is that experienced by the hottest junction on the chip. This junction temperature is related to the temperature indicated by the lacquer on the hot spot of the component case by the relationship:

(1) $\quad Tj = Tc + {}_j\delta_c * W$

where:

Tj = Chip junction temperature, °C.

Tc = Temperature at hottest spot on component case, °C.

${}_j\delta_c$ = Thermal resistance between junction and hottest spot on case, °C per watt.

The parameter ${}_j\delta_c$ is not necessarily identical to the overall thermal resistence of a component, $({}_j\theta_c)$.

Estimating Junction Temperatures of Power Semiconductors

For power semiconductors with heat sink flanges, ${}_j\delta_c$ and ${}_j\theta_c$ values are approximately equal and manufacturers' ${}_j\theta_c$ values can be used (with caution) for junction temperature estimates using equation (1). For example, consider a voltage regulator with a ${}_j\theta_c$ of 4°C per watt which dissipates 10 watts in a personal computer power supply. If a lacquer with a 66°C indicating temperature on the mounting tab remained opaque under operating conditions with a 25°C ambient temperature at sea level, then the following junction temperature estimate can be made:

$${}_j\delta_c = {}_j\theta_c + 4°C/W$$

$$W = 10 \text{ watts}$$

$$Tc < 66°C$$

and

(2) $\quad Tj < Tc + {}_j\delta_c * W$

$$Tj < 66 + 4 * 10$$

$$Tj < 106°C$$

The regulator junction temperature is less than 106°C with a 25°C ambient temperature.

Estimating Junction Temperatures of Integrated Circuits in DIPs

Figure A-1A(1) is a plot of junction-to-case temperature differences vs. position along the top of a Fairchild 723 voltage regulator in a 14 pin ceramic DIP. This plot was obtained from tests of the regulator using the forward voltage characteristics of Zener diode on the semiconductor chip to determine junction temperature, and liquid crystals to determine case temperature under operating conditions. ${}_j\delta_c$ is approximately 22°C per watt (${}_j\delta_c$ is approximately 80% of the measured ${}_j\theta_c$ value of 27°C per watt).

Figure A-1B is a plot similar to Figure A-1A for the Fairchild 723 voltage regulator in a 14 pin plastic DIP. ${}_j\delta_c$ is approximately 38°C per watt (${}_j\delta_c$ is approximately 54% of the measured ${}_j\theta_c$ value of 70°C per watt).

For personal computer testing, the measured ${}_j\delta_c$ values for the 723 14-pin DIPs can be used for estimating junction temperatures of devices in DIPs different than the 723 package in lieu of better data. For natural convection cooling and/or for 16 pin and larger DIPs, junction temperatures calculated with the 723 ${}_j\delta_c$ values may be different (but perhaps higher) than actual values, but errors of less than 10°C can be expected. Junction temperatures can be estimated from DIP case temperatures in the same way as for the power regulator example.

Maximum Junction Temperatures

I.C. semiconductor manufacturers usually specify maximum junction operating temperatures of 125°C or higher. Temperature related reliability can be expected to improve approximately by a factor of 2 with each 10°C decrease in junction temperature. For personal computer semiconductors, maximum junction temperatures of less than the 100°C range may be appropriate. Where high reliability is desired, maximum junction temperatures of less than the 85°C range may be selected.

Screening Out Cool Components

As a first step in evaluating semiconductor junction operating temperatures, all cool semiconductors can be screened out. Selection of the appropriate lacquer indicating temperature for ceramic and plastic DIPs is outlined in Table A-1 for a maximum DIP junction temperature in the 85°C temperature range. The power dissipation values of 0.9 and 0.5 watts for ceramic and plastic DIPs represent maximum values that would be expected for the hottest DIP types used in personal computers.

Although plastic DIPs have higher thermal resistances than ceramic DIPs, the maximum expected power level of plastic DIPs is lower. Thus, case temperatures for screening plastic and ceramic DIPs are nearly the same. This is shown in Table A-1. For screening out DIPs with junction temperatures in the 85°C range or less, a lacquer with indicating temperature of 66°C is recommended.

Testing Hotter DIPs Which Do Not Pass the Screen

. Power levels of those components which do not pass the screen (the lacquer melts) can be determined from manufacturers' data (with caution) and higher temperature lacquers can be applied to determine maximum hot spot temperatures. Then, junction temperatures can be estimated using equation (1).

Where it is desired to decrease component temperatures, changes in the cooling system may be employed, such as adding air vent holes, repositioning components or P.C. boards to cooler locations or to augment convection, shielding or cooling adjacent hot components (such as power resistors), installing heat sinks, or adding a fan. If a hot component is a power semiconductor, circuit changes to reduce component power dissipation might be considered.

Example: Testing Component Temperatures in the Commodore PET 2001 Series Personal Computer with 8K Memory

Screening all low temperature components: 66°C indicating temperature lacquer (Tempil®) was applied to all components including power semiconductors. After operation for 3 hours in a 23°C ambient temperature at sea level while running the program "Wrap Trap," no clear lacquer was observed on any component except for all of the MOS Technology 6550 RAMs. From the values of Table A-1, it was concluded that all DIP junctions, except for all of the 6550 RAMs, were cooler than approximately 85°C.

The 6550 RAMs, packaged in 24 pin plastic DIPs, have typical power dissipations of 0.45 watts listed by the manufacturer. With a thermal resistance of 38°C per watt, a modification of equation (1) gives a value of Tj - Tc of 17°C. Thus, it is estimated that the junction temperatures of the 6550 RAMs exceeded 83°C.

No further testing of the power components was undertaken.

Testing the MOS Technology RAMs: Additional temperature tests with 83°C and 76°C indicating temperature lacquers were conducted on the 6550 RAMs. No 6550 RAM case temperature exceeded 83°C. DIPs I2, I4, and J3 had case temperatures exceeding 76°C. (Those DIPs that are hottest may vary according to the particular program that is running.) As noted, junction temperatures are expected to be 17°C hotter than case hot spot temperatures.

Results: The temperature test results of the PET computer are summarized in Table A-2. It was concluded that the PET computer cooling was adequate for the operating conditions tested.

Concluding Remarks and Recommendations

The technique here outlined is convenient and inexpensive for temperature testing of personal computers. The accuracy of the technique is dependent on values for $_j\delta_c$ which are available for only 14 pin ceramic and plastic DIPs. However, extrapolation of the 14 pin results to other DIPs is expected to give errors within acceptable limits.

It is recommended that the accuracy of each particular lacquer used be verified on dummy components in an oven using an accurate thermometer. With some lacquers, detection of melting after the component cools may be difficult. Also, lacquer indicating temperatures may change after long term exposure in personal computer operating environments.

The author solicits comments on the temperature testing technique, additional $_j\delta_c$ data, and information on additional suppliers of temperature indicating lacquers and/or labels.

Reference (1)

Test and Analysis for Development and Evaluation of Computer Air Cooling Systems. P. Merrill, Paper presented to Thermal session, 1978 NEPCON West, February 28 - March 2, 1978.

Table A-1. Determining Lacquer Indicating Temperatures for Screening Out DIPs with Junction Temperatures Less Than Approximately 85°C

COMPONENT	① Estimated θ_{jc} °C/W	② Maximum Power W	③ Maximum Tj-Tc ① * ② °C	④ Maximum Junction Temp. Tj °C	⑥ Maximum Case Temp. ④ - ③ °C	Selected Lacquer Indi- cating Temp. °C
Ceramic DIP	22	0.9	20	85	65	66
Plastic DIP	38	0.5	19	85	66	66

Table A-2. Summary of Temperature Test Results on Commodore PET Series 2001 with 8K Memory, Serial No. 0010159. Ambient Temperature 23°C. Sea Level Altitude. Running the Program "Wrap Trap."

COMPONENT	ESTIMATED CHIP JUNCTION TEMPERATURE °C
(1) MOS Technology 6550 RAMs:	
(1a) I1, I4, J3	93 to 100
(1b) I1 through 8 and J1 through 8 except for (1a) above.	83 to 93
(2) All DIPs except for (1)	< 85 (approximately)

Note: All power component and discrete semiconductor case temperatures were less than 66°C.

Table A-3. List of Suppliers of Temperature Indicating Lacquers and/or Labels

Note: Some of the temperature sensitive materials may be available from local welding equipment suppliers

Markal Paint Company
270 North Washington Avenue
Chicago, Illinois 60612
Telephone: 312/826-1700

Teletemp Corporation
P.O. Box 5160
Fullerton, California 92635
Telephone: 714/879-2901

Omega Engineering Inc.
31 Knapp Street
Stamford, Connecticut 06907
Telephone: 203/322-1666

Tempil Division, Big Three Industries, Inc.
2901 Hamilton Avenue
South Plainfield, New Jersey 07080
Telephone: 201/757-8300

CERAMIC DIP

$_j\delta_c = 22\,°C/W$

POSITION FROM LEADING EDGE ALONG
LENGTH OF PACKAGE, INCHES.

Figure A-1A

PLASTIC DIP

$_j\delta_c = 38\,°C/W$

POSITION FROM LEADING EDGE ALONG
LENGTH OF PACKAGE, INCHES.
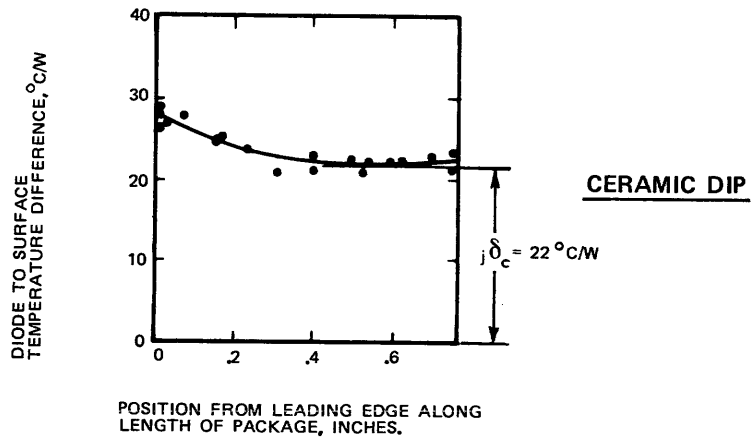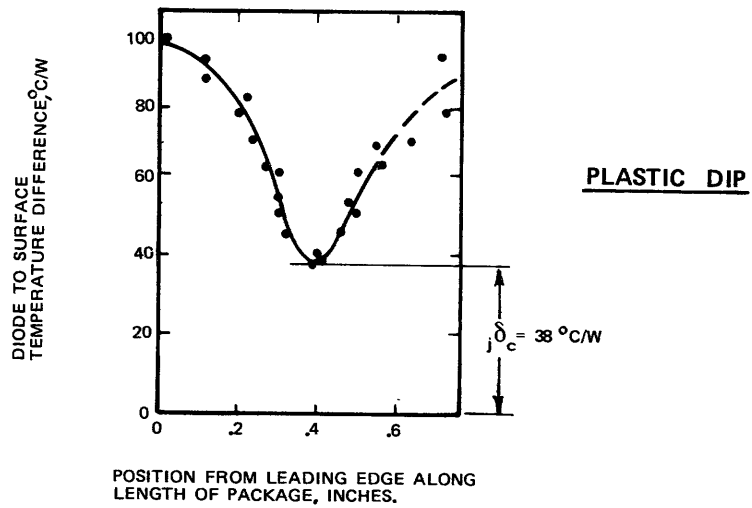
Figure A-1B

TEMPERATURE DIFFERENCES BETWEEN JUNCTIONS AND
TOP SURFACES OF 723 VOLTAGE REGULATOR CHIPS IN
14 PIN DIPS WITH AIR FLOWING AT 250 FEET AND 500
FEET PER MINUTE ALONG LENGTH OF PACKAGE.

John Walker, Partner, Marinchip Systems
16 St. Jude Road, Mill Valley, CA 94941
(415) 383-1545

Introduction: The M9900 CPU

Marinchip Systems has developed a
CPU board which allows the Texas
Instruments TMS9900 processor to be
used on the enormously popular S-100
bus. This board, the M9900 CPU,
replaces the 8080 or Z-80 CPU board
in an S-100 mainframe, and uses the
existing memory and peripherals, or
almost any S-100 compatible board.
The TMS9900 is an extremely powerful
processor, which features hardware
multiply and divide instructions,
multiple sets of 16 general purpose
registers, addressing modes
including direct, indirect, indexed,
and auto-increment, and a context
switch mechanism that allows both
rapid interrupt response and user
extension of the hardware
instruction set. The TMS9900 is a
single chip processor that is
comparable to the PDP-11 in most
respects, and exceeds it in several
important ways, notably the
provision of 16 registers instead of
8 and the fact that the user can
have as many independent register
sets as he wants, and need not be
limited to hardware registers within
the CPU.

The purpose of this paper will be to
explore the problems of interfacing
a fully parallel 16 bit processor to
the S-100 bus, concentrating on the
peculiarities of the S-100 bus
structure and the solutions that
were developed to surmount them.
The performance tradeoffs and
compromises made in the design will
be discussed, and finally some
recommendations regarding standards
for certain S-100 signals will be
presented.

Since this paper will be discussing
signals generated by both the 8080
and the TMS9900 processors, as well
as S-100 bus signals, some
conventions have been adopted to
lessen confusion. Signals generated
by either CPU chip are referred to
by the name given them by the chip

manufacturer. Which chip is being
referred to should be clear from
context. S-100 signals will always
be suffixed by their pin number in
parentheses. Active low signals
will be identified by a minus sign
following the signal name.

TMS-9900 Bus Structure

The task of interfacing one
processor to a bus designed for
another involves designing logic
that will transform the signals
generated by one processor into the
corresponding signals of the other
processor. The fact that the
processors have different word
lengths increases the complexity of
the solution, but introduces few
conceptual problems. First, let us
look at the signals used by the
TMS9900 processor. The TMS9900 is
one of the simplest microprocessors
to interface, largely because there
is no multiplexing of information on
package pins. Each pin has one
clearly defined function, so the
external decoders and latches
required in many other systems may
be eliminated. The price paid for
this simplicity is the 64 pin
package required to accomodate all
the independent signals. The
signals which are relevant to memory
accesses are the address bus, data
bus, and control bus. The address
bus consists of 15 lines
representing bits 1 through 15 of
the address being referenced. Since
the TMS9900 uses 16 bit wide memory,
the 15 address bits permit
addressing of 64K bytes of memory.
(Byte addressing is handled by
masking data within the processor,
and is of no concern in interfacing
it.) The data bus is a 16 bit
bidirectional bus which behaves
exactly like the data bus of an
8080, except that it is twice as
wide. The control bus consists of
the signals that control the
external memory. These signals all
appear on their own dedicated pins,
so no status latching is required as

for the 8080. The signal MEMEN- is a low-true signal which indicates a memory access is in progress. When it appears, a valid memory address is present on the address bus. The signal DBIN indicates the direction of the memory data transfer. When DBIN is high, the output drivers on the 16 data bus lines have been disabled to allow the memory to place data on the lines in response to a memory read. MEMEN- and DBIN simultaneously active indicate a memory read, while MEMEN- active and DBIN inactive indicate a memory write. The TMS9900 generates a write enable signal, WE-, which goes active during a write cycle after the address and data have been present for sufficient time to satisfy the setup time requirements of most static memories. This signal allows easy interfacing of memories to the processor, but is not used in the M9900 CPU because the memory cycle must be segmented into two independent 8 bit accesses. The READY and WAIT lines permit the processor to delay to accomodate slow memories. After the address has been presented to the memory bus and MEMEN- goes active, READY is sampled. If high, the memory cycle will complete on the next clock cycle. If READY is low, the WAIT signal is brought active and the processor will delay one cycle, at which time it will test READY again, and either proceed or delay again depending on its state. In the M9900 CPU the READY line is used to cause the processor to wait for the two cycles on the S-100 bus to be completed. The WAIT line is not used, since the M9900 must generate the S-100 wait signal in response to a wait state encountered on either (or both) of the 8 bit accesses. The above signals comprise the complete memory interface of the TMS9900, and seem to be the minimum orthogonal set of signals allowing control of variable speed memories with no external decoding. The TMS9900 has no complex timing relationships between signals. Every signal generated by the processor is related to one of the four clock phases, and there is a uniform propagation delay, typically 20 nanoseconds, from the clock edge to the signal.

8080 Bus Structure
_____

The memory bus of the 8080 is similar in concept to that of the 9900, but more complicated in practice because the control signals relevant to memory are divided into command and control signals which appear directly on processor pins and status signals which are output on the data bus during the first state of each machine cycle. A memory cycle in the 8080 consists of three or more machine states. During the first state, the 8 bit data bus outputs the status bits which indicate what type of cycle is being performed. The processor generates a signal, SYNC, which identifies the first state of the cycle and may be used to latch the status bits for use later in the cycle. The status bits, properly decoded, identify the type of cycle. The ones relevant to memory and I/O cycles are MEMR, WO-, INP, and OUT. The MEMR bit identifies a memory read cycle (except when it comes on during a halt cycle, which will be ignored henceforth). The INP and OUT bits identify I/O read and write cycles, respectively. The WO- signal is active when a write is being done to either memory or an I/O device: external logic must generate the write strobes for I/O and memory based upon the OUT bit. The rest of the status bits are used to implement the 8080 interrupt scheme, to indicate when the machine is halted, to flag accesses to the stack, and to indicate when an instruction byte is being fetched. The command and control signals are almost identical to those of the TMS9900 in name and function. The DBIN signal indicates that the data bus is in input mode to receive data, WR- is the delayed output data strobe, READY is the line that may be used to stretch the length of a memory or I/O cycle, and WAIT is the signal that confirms that the processor is waiting. The address bus on the 8080 is 16 bits wide, since the 8080 uses byte-addressable memory, and the data bus is an 8 bit bidirectional bus. I/O on the 8080 provides 256 ports. An IN or OUT instruction referencing one of these ports causes a bus cycle identical to a memory cycle, except that it is identified as I/O by the presence of the INP or OUT bits on the status bus. In an I/O cycle, the 8 bit port address is presented on BOTH the high and low 8 bits of the

address bus. (This "address mirror" feature is intended to reduce bus loading by dividing devices between the high and low bytes of the address lines in system without address buffering.) The timing relationships of the 8080 signals are quite complicated. The 8080 is driven with a two phase clock. The two phases must not overlap, and have different widths. The command and control signals may be delayed as much as 120 nanoseconds from their controlling clock edge, and the address and data may not be stable for as much as 220 nanoseconds after the clock edge.

S-100 Bus Structure

The S-100 bus is largely the result of simply buffering the signals generated by the 8080 and directly presenting them to the bus. It will help to break up the bus signals into groups and discuss them independently.

Clocks. There are three clock signals on the S-100 bus. The first two are the Phase 1 and Phase 2 processor clocks, presented as TTL levels, PH1(24) and PH2(25). The third signal is named CLOCK-(49), and is an inverted Phase 2 clock, delayed about 60 nanoseconds (an artifact of the original Altair design). Both PH2(25) and CLOCK-(49) are commonly used as timing references by peripheral boards.

Address Lines. The 16 address lines appear on the bus, buffered directly from the processor address lines. The line ADDR DSBL-(22) will cause the address lines to go to high impedance when pulled low.

Data Lines. The 8 bit bidirectional processor data bus is split by logic on the CPU board into an 8 bit data in bus and an 8 bit data out bus on the S-100 bus. Whether the data in bus is applied to the processor data bus is controlled by the DBIN signal from the processor. Since the data out lines are dedicated to output, their drivers are always on, except if disabled by pulling the line DO DSBL-(23) low.

Command and Control Lines. The direct control lines from the processor appear on the command and control S-100 lines. The signal PSYNC(76) is the SYNC signal from the processor that identifies the first state of each machine cycle. PWR-(77) is the processor delayed write strobe. PDBIN(78) is the DBIN signal from the processor. Most (but not all) S-100 boards only drive the data in bus when PDBIN(78) is high. PINTE(28) is the signal from the processor (INTE) which indicates that the processor has interrupts enabled. PWAIT(27) is the WAIT signal that indicates the processor is waiting for the READY signal from a peripheral. PHLDA(26) is the HLDA processor signal which acknowledges a DMA request. (Refer to the section below on DMA control signals for more information on the PHLDA(26) signal.) The six command and control lines may be forced to high impedance by pulling the line CC DSBL-(19) low.

Status Lines. When the status signals appear on the processor data bus during the first state of a cycle, they are latched and presented on a separate set of status lines on the S-100 bus. The status bus remains stable for the duration of the machine cycle. The SM1(44) line indicates the current cycle is fetching the first byte of an instruction. The SOUT(45) bit identifies an output cycle (initiated by the OUT instruction). The SINP(46) pin identifies an input (IN instruction) cycle. SMEMR(47) indicates a memory read cycle, and SHLTA(48) indicates that the processor is halted. The SWO-(97) signal indicates a write-type operation: if SOUT is low, it is a memory write. If high, it is an I/O write. The SINTA(96) signal identifies the special 8080 cycle that responds to an interrupt by requesting an instruction to be executed by the processor, and the SSTACK(98) signal flags the current memory cycle as referring to the stack. The status bus signals may be forced to high impedance by pulling the STAT DSBL-(18) line low.

Memory Control Lines. These lines are a collection of signals relating to memory and I/O cycles. The PRDY(72) line is applied through a buffer to the READY pin on the 8080, and is used by slow memory and peripherals to stretch a cycle that references them. The MWRITE(68)

signal is a copy of the PWR-(77) write strobe that only goes low on memory writes (e.g., when SOUT(45) is low). This is the most commonly used write control signal in memory boards. It is also quite confusing in that many S-100 machines do not generate it on the CPU board. The Altair and IMSAI, for example, generate this signal on the front panel board, with the result that memories which require this signal will not run if the front panel is removed. To compensate for this, many of the "reset and go" PROM/RAM boards contain logic to generate this signal, since they are replacing the front panel in a system. This signal is logically a CPU signal, and really has no business being generated anywhere else, but the weight of history is great, so for the time being confusion will reign. The UNPROT(20), PROT(70), and PS-(69) signals respectively clear, set, and contain the status of the memory protect flip-flop on the currently addressed memory board. Since memory protect is little used, they will not be discussed further. The PHANTOM-(67) signal is used by many systems and boards to allow an automatic power-on start from a nonzero address. The memory board at address zero is strapped to be deselected regardless of the address on the bus when PHANTOM-(67) is low. A PROM board then can, when triggered by a reset signal, disable the memory at address zero, enable itself, and cause the CPU to execute instructions from the PROM even though the CPU "thinks" it is executing from address zero. The PHANTOM-(67) line is normally turned off once the processor has jumped into the PROM itself. The memory at address zero may then be referenced normally.

Interrupt Signals. The lines VI0-(4) through VI7-(11) are the vectored interrupt requests. Pulling one of these lines low will cause an interrupt at the corresponding level, assuming that interrupts are enabled and the processor is not servicing a higher priority interrupt. VI0-(4) causes an interrupt which is not distinguishable from a reset of the processor, and hence should be used with discretion. Since the early S-100 processors did not provide

vectored interrupts on the CPU board, the signal PINT-(73) was defined to permit a separate vectored interrupt controller board to request an interrupt from the processor. In such a system, the vectored interrupt controller board received the 8 vectored interrupt lines, chose the highest priority, then requested an interrupt using the PINT-(73) line and supplied the processor with an interrupt instruction when the SINTA(96) line went high. Because of the way the 8080 CPU interrupt structure worked, systems without a vectored interrupt board could provide a single level interrupt simply by pulling the PINT-(73) line low.

Front Panel Signals. The following signals provide the interface between the front panel and the boards on the bus. Since some systems have complex front panels and others have only a reset switch, most of these signals may be absent in a particular system. The only signals in the front panel group present in all S-100 mainframes are PRESET-(75) and POC-(99). PRESET-(75) is pulled low when the reset switch on the front panel is activated. The CPU board normally responds to PRESET-(75) by resetting the processor and pulling POC-(99) low. POC-(99) is also driven by logic that keeps it low on a power-up operation until the supply voltages have stabilized. Thus, if peripheral boards use POC-(99) as their clear signal, they will be properly reset both by initial application of power to the system and by the reset switch. The following signals are present only in those systems with elaborate front panels. The RUN(71) signal is used to indicate that the front panel RUN/STOP switch is in the RUN position. The SSW DSBL-(53) signal is used by some front panels to implement data input to the processor from the front panel switches. The EXT CLR-(54) signal resets I/O devices independent of the processor. What it does and when it is activated depends upon the peripheral and mainframe being examined. The SS(21) signal is used by certain front panels to implement a single step function. It allows the front panel to place data directly on the processor data bus (this requires a connection from the

front panel to the CPU card). The XRDY(3) signal allows the front panel to stop the processor without interfering with the PRDY(72) signal used by memory and peripherals. The typical CPU ANDs XRDY and PRDY to develop the READY signal for the 8080.

DMA Control Signals. Direct Memory Access (DMA) can be implemented on the S-100 bus through use of the PHOLD-(74) signal. When this signal is pulled low, the processor suspends execution at the end of the current instruction, and raises the HLDA line, which appears on the bus as PHLDA(26). Once the requesting peripheral sees this signal, it can pull the lines STAT DSBL-(18), CC DSBL-(19), ADDR DSBL-(22), and DO DSBL-(23) low and assume control of the bus itself. It should be noted that pulling the CC DSBL-(19) line low forces the PHLDA(26) signal itself to high impedance. If this signal is being used as a direct enable by the DMA board, its floating may lead to havoc, especially because line 25 right next door is the Phase 1 2MHZ clock, which puts a lot of noise on the floating line. Pulling up PHLDA(26) to +5 with a resistor neatly solves this problem, but few CPU boards do this.

Power and Ground Lines. Power to the S-100 bus is unregulated, since each board regulates the raw supply to its desired logic levels. Pins 1 and 51 supply a nominal +8 volts. Pin 2 supplies +16 volts, and pin 52 supplies -16 volts. There are no specifications and few conventions about the voltages actually found on these pins. Specifically, some very high voltages occur in certain mainframes. If the +8 line goes to 12 volts, the dissipation in the on board regulators may exceed the capacity of their heat sinks, forcing thermal shutdown of the regulator and malfunction of the board. One might point out as an aside that since these supplies are unregulated they contain ripple, and at least one CPU board counts on this! The ripple is amplified with an op-amp and used to provide a line frequency real time clock. Pins 50 and 100 are the common ground return for all supplies, and the logic ground reference. Whether or not this is tied to safety (green wire)

ground depends upon the mainframe. Pin 55 is defined as chassis ground, but very few mainframes actually connect this pin.

The reader might have noticed that it took a lot more words, explanations, and hedging to explain the S-100 bus than to explain the 8080 processor signals themselves. This is the case because one of the many different CPU boards or "compatible" I/O boards gives lie to just about any definitive statement regarding the bus one might choose to make (except possibly that it has 100 pins).

Interfacing to the S-100 Bus

The job of interfacing the TMS9900 to the S-100 bus fell largely into three phases: mapping the signals from the 9900 bus to the S-100 bus, defining the subsystems that would make up the CPU card, and logic design and debug. First of all, the memory cycles of the TMS9900 and the 8080 were drawn out side by side, and a mapping was defined between the two. It was found that every signal corresponded in both directions closely enough that once 16 bit TMS9900 bus was multiplexed onto the 8 bit S-100 bus, all the other signals could be translated. In this phase of the design, it was decided to implement S-100 I/O by providing memory mapped I/O in a page of the TMS9900 addressing space. This was done because the unique TMS9900 I/O scheme did not lend itself to adaptation to existing S-100 peripherals, and because the memory bus interface that had to be designed for the TMS9900 already contained all the logic required to implement the I/O cycle. Once it became clear that the problem was solvable, the following subsystems that made up the TMS9900 CPU board were defined.

Clock Generator. This subsystem contains the generator for the four phase clock required by the TMS9900 and the logic that transforms the 9900 clock pulses into a synchronized replica of an 8080 clock. A properly delayed inverted signal is generated for the CLOCK-(49) signal.

Address Bus Driver. This subsystem takes the address bus of the 9900

and drives the address lines on the
S-100 bus. Depending upon whether
the address on the address bus is
within the memory mapped I/O region,
this logic either drives the address
directly onto the bus (for normal
memory accesses) or places the low 8
bits of the 9900 address bus on both
the high and low 8 bits of the S-100
address lines (for I/O accesses).
For normal memory references, the
upper 15 address bits on the bus are
supplied by the 9900 chip, and the
low order bit is generated by the
memory bus controller subsystem (see
below).

Data Bus Driver. This subsystem
contains the drivers that route data
to and from the data in and data out
lines. The data out lines are
driven by two sets of drivers that
can route either the high or the low
byte of the 9900 16 bit data bus to
the data out lines. The data in
lines are received by two 8 bit
latches that save the data appearing
on the data in lines for parallel
presentation to the 9900 as a 16 bit
data word. In addition, extra
drivers provide the data paths
necessary to use the data in and
data out lines as a 16 bit parallel
bidirectional data bus for use with
special Marinchip high-performance
memories (see the description of
"Sixteen bit mode" below).

Memory Bus Controller. This
subsystem is the heart of the M9900
CPU. When a memory access is
initiated by the TMS9900, the memory
bus controller is started.
Depending upon whether the address
presented by the 9900 is within the
memory mapped I/O region or not, the
controller generates an I/O or two 8
bit memory cycles. While the
controller is operating, the READY
line of the 9900 is held low to
force it to wait for the data to be
transferred on the S-100 bus. The
memory bus controller is a random
logic implementation of the 8080
memory timing logic, which generates
PSYNC(76), PWR-(77), MWRITE(68),
PDBIN(78), and PWAIT(27) as if an
8080 were directly connected to the
bus. PRDY(72) is honored with the
same timing as the 8080 expects.
The memory bus controller also
generates the signals to the data
bus drive subsystem that route data
to and from the S-100 bus and the
9900 data bus, and the strobes that

latch data from the S-100 bus at the
same time an 8080 would have sampled
it. When the memory bus controller
gets the PRDY(72) for the final byte
of a transfer, it raises READY to
the 9900, allowing it to proceed
with its execution. By anticipating
ready it is possible to complete the
16 bit memory cycle in the same time
an 8080 would have taken to fetch
two bytes (six machine states). I/O
cycles take three machine states.
The PINTE(28) signal, which
indicates that interrupts are
enabled in the 8080, is always high,
since the 9900 does not provide this
status on an external pin. The
PHLDA(26) signal, which acknowledges
a DMA request is simply the HOLDA
pin of the 9900 buffered to the bus.
A pull-up resistor is provided on
this line to prevent the "floating
PHLDA(26)" problem mentioned above.
The DMA request line, PHOLD(74), is
latched before being passed to the
HOLD line on the 9900. While this
signal is internally latched in the
9900, the internal latching is done
at a different time in the 9900 and
the 8080, so the external latch
accomodates any device that counts
on the 8080's timing. This may be
paranoid, but better to be over
compatible than to discover the CPU
doesn't work with somebody's "sloppy
disc" controller.

Status Bus Generator. Logic decodes
signals from the 9900 and the memory
bus controller to generate the
status bus signals. SWO-(97),
SHLTA(48), SOUT(45), SINP(46), and
SMEMR(47) are generated compatibly
with the 8080. Since SSTACK(98) and
SINTA(96) represent concepts not
applicable to the 9900, they are
always low. On an 8080 system, the
status bus signals may also be read
from the data bus when PSYNC(76) is
high (an artifact of the way the
8080 supplies these signals). The
M9900 CPU does not include logic to
place these signals on the data bus,
since it was felt that no board
designer would count on this quirk
of the 8080. Few do.

Reset and Load Generator. The reset
and load generation subsystem
generates the two nonmaskable
interrupts for the 9900. The
subsystem is designed so that the
reset switch on the S-100 mainframe
may generate either signal. The
9900 reset traps to address zero,

while the load causes a trap to address FFFC, so by strapping the CPU board properly, the user may run with ROM at either end of memory. The reset and load subsystem also decodes the 9900 instructions that force a reset or load operation, and generate signals equivalent to performing the actions via the external switches. A power-up reset circuit causes the CPU to start automatically at the selected address after power is stable. The power on clear signal, POC-(99), is generated.

Interrupt Controller. The 9900 features 16 level priority interrupts within the chip, so only a latch to stabilize the eight interrupt requests on the S-100 bus and a priority encoder are required to provide 8 level interrupts. The other 8 9900 interrupts were not used, and the 8080 trick of requesting an interrupt via the PINT-(73) line was not implemented. The user can easily enough strap PINT-(73) to one of the vectored interrupt lines if this is required.

Once the problem had been subdivided into the above subsystems and the functions of each subsystem clearly defined "all" that remained was to design the logic within each subsystem. This was done, a prototype was built, and the inevitable debug phase was begun. Other than straightforward design errors, several problems began to crop up which seemed to indicate problems inherent in the S-100 standard itself: problems that will beset any designer of a non-8080 CPU board. It is these problems that will be discussed below:

Wait State Timing Reference. In peripherals that stretch the bus cycle with the PRDY(72) signal, there is no convention regarding which pin defines the start of a memory or I/O cycle. Some boards use PSYNC(76), others use the status bus signals or MWRITE(68). The M9900 CPU originally generated the status bus signals immediately at the start of the memory cycle rather than waiting 200 nanoseconds like the 8080, and this caused some boards to miss their wait states because they were starting one-shots to time the wait state when the status bus signals changed. The

early status signals from the M9900 caused the one-shot to expire before the processor sampled PRDY(72). The final M9900 design delays the status bus signals for compatibility. Since there are many ways to decide what kind of cycle is being performed from the signals on the bus, there are correspondingly many ways to time the cycle. It seems unreasonable to expect every processor to reproduce all the 8080 signal timing relationships exactly, so a simple definition of what is to be used would be a great boon to CPU and peripheral board designers both. It would be nice if we decided to make MWRITE(68) universal. Then we could count on SINP(46), SOUT(45), SMEMR(47), and MWRITE(68) as four simple signals that identify the current cycle.

DMA Protocol. The whole area of DMA protocol on the S-100 bus is very loosely defined. The statement that the DMA device must keep PHOLD-(74) low until the processor responds with PHLDA(26) would ease one area of confusion. The existence of separate disable lines for each of the buses is too ingrained in ingenious bootstrap schemes to do away with at this late date, however, some statement such as "The disable lines will never be activated unless the PHLDA(26) line is active" would be a step in the right direction.

Utility Clocks. An 8080 on the S-100 bus provides perfectly good 2 Mhz clocks on three separate pins. Boards which need a clock for general timing such as generating communications baud rates or implementing a real time clock have no particular reason to choose one over the other. This is a problem when trying to use a processor that runs at a speed other than 2 Mhz. We should define PH1(24) and PH2(25) as CPU clocks which run at the processor clock rate and to which memory timing is relative. CLOCK-(49) should be defined as a 2 Mhz utility clock signal which bears no specific timing relationship to the CPU clock.

Unique Features of the Design

The M9900 CPU has succeeded in interfacing the TMS9900 to the S-100 bus. The CPU is a single board

system that will run with most
existing S-100 peripherals and
memories. It is, in fact, more
compatible with 8080 peripherals
than many Z-80 boards, particularly
in interfacing to dynamic memories.
The major innovation in S-100 bus
architecture introduced by the M9900
CPU is its ability to use the S-100
bus for 16 bit parallel data
transfers.

Sixteen Bit Mode. Obviously,
forcing a 16 bit parallel processor
to do all of its memory accesses in
byte-sized chunks reduces its
performance compared to a fully
parallel system. In most
applications, the increased power of
the TMS9900 instruction set results
in such a performance gain that the
overhead introduced by the extra
memory accesses is overcome. For
those applications which require the
greatest CPU power, the M9900 CPU
allows the use of 16 bit parallel
memory in the S-100 chassis. The
key to this option is the signal
that Marinchip Systems has
designated SXTN-(60). When a memory
cycle starts, if this signal is
pulled low by the addressed memory
board within 125 nanoseconds, the
SYNC cycle will be aborted, and the
memory transfer will be performed 16
bits parallel, using the S-100 data
out bus for the high byte and the
S-100 data in bus for the low byte.
A memory cycle performed in 16 bit
mode will complete in one
microsecond (assuming no wait
states), compared to three
microseconds if conventional S-100
memories are used. This can result
in performance gains approaching
three-to-one. Since the memory
itself informs the processor if it
is capable of 16 bit operation, it
is possible to mix conventional 8
bit and 16 bit memory in the same
chassis. Obviously, a 16 bit memory
is a rather special beast,
especially so if it is capable of
operating in 8 bit mode for DMA
transfers. None the less, the
design of such a memory is a
straightforward task beside the
design of the M9900 CPU itself.

CRU I/O Signals. Little has been
said about the TMS9900's unique
bit-addressable I/O scheme, the
Communications Register Unit (CRU).
This mechanism performs bit-serial
transfers from the processor to
external devices within a 4096 bit
address space. This approach
permits peripheral chips to be built
with few costly pins and no external
logic. For example, the UART for
the 9900 family is an 18 pin device,
compared with the 28 to 40 pins
required with other microprocessors.
Since the CRU I/O was not used in
interfacing to the S-100 bus, it was
simply brought to the bus on three
unused pins. Future 9900
peripherals may be built to take
advantage of this unique I/O system.

Summary

The M9900 CPU is a happy marriage of
the advanced 16 bit architecture of
the TMS9900 and the widely accepted
S-100 bus. The interfacing of the
9900 revealed several
characteristics and problems of the
S-100 bus that deserve to be
standardized. The design of the
M9900 CPU took a very conservative
approach to the S-100 bus, and tried
to generate virtually all the
signals provided by an 8080, with
the same timing relationships.
Careful work by a standards group
would have permitted a much more
straightforward design, and will
greatly ease the task of others who
wish to interface foreign processors
to the bus. The extensions to the
S-100 bus made by Marinchip Systems
were to provide the unique
performance and features of the 9900
within the existing bus
architecture. The author would
welcome communications regarding
standards for 16 bit (and wider)
operation on the S-100 bus.

SINGLE CHIP MICROCOMPUTERS FOR THE HOBBIEST

John Beaston, Microcomputer Applications
Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

Abstract

Recent advances in solid state technology have allowed integrated circuit manufacturers to place all the elements of a computer system on a single silicon die. Although most of these single chip microcomputers are ROM (Read Only Memory) based and primarily intended for high volume users, EPROM (Erasable, Programmable ROM) based versions have also been developed to overcome the high development and rework costs associated with their ROM counterparts. This EPROM technological fallout opens up a whole new world of versatile, low cost, dedicated control for the computer hobbiest. As an introduction to single chip microcomputers, a specific microcomputer, the EPROM based Intel 8748, is discussed in detail with both the hardware and software aspects explained. Finally, a typical application example is presented.

Introduction

One of the few areas not often addressed by computer hobbiest is that of control applications for microprocessors. A sample of such applications might include burglar/ fire alarms, radio controlled models, printer controllers, and ham radio accessories. There are probably several reasons for the interested hobbiest not tackling such projects. The greatest deterent is the non-availability of specialized hardware. The hobbiest using an S-100 based computer as a burglar alarm is probably guilty of processor overkill, however designing and building a dedicated microprocessor based alarm system would be expensive and very hardware intensive, so there really has been no alternative. Portability is another consideration. Many control applications like radio controlled models preclude the use of most microprocessor based systems simply due to high parts count and power supply demand. Handling multiple tasks is also a problem. If one processor is to control several tasks such as a printer and keyboard in addition to the alarm system, the software must be written with this in mind. Adding additional tasks is usually a difficult task in itself. And finally, it is well known that even a minimal system containing the required amount of I/O

and memory for the usual control application is not inexpensive. With these deterents, it is no wonder that hobbiests have not pursued the more control-oriented applications. But never fear, a need has been seen and filled.

Recent advances in semiconductor technology have enabled integrated circuit manufacturers to pack all the elements of a control-oriented microprocessor system on a single silicon die. These microcomputers contain a CPU, RAM, EPROM/ROM, I/O, timers, and a clock generator all on one chip. Most of these microcomputers are capable of stand-alone operation while some operate as a slave processor to a more powerful master processor.

These microcomputers remove all of the deterents mentioned earlier. They have obviously low parts count. Their power requirements are correspondingly reduced. Multi-tasking is easy since each task may have its own controller. And their cost is significantly below that of a microprocessor based system with equivalent capabilities. With these deterents removed those hobbiests interested in control applications can now come "out of the closet" and invent and build to their heart's content.

Desirable Control Characteristics

As an introduction to single chip controllers, let's examine several characteristics which are desirable in single chip control applications.

1. Single 5v supply

2. Code efficient instruction set

3. Timing control

4. Interrupt capability

5. Expandable memory and I/O

6. Compatible ROM/EPROM versions

The single supply characteristic is almost universal in the present and upcoming generation of both microcomputers and microprocessors. Its advantage is obvious - fewer power supply hassles.

All instruction sets, whether in microcomputers or microprocessors should be code efficient. However, this characteristic is particularly important for single chip controllers. With all memory on-chip, each byte is precious. So every method to conserve code is desirable. One method of achieving code efficiency is to limit the addressing range to, say, 4K bytes rather than the usual 65K. (Since control applications rarely require more than 1K of program memory, this is not really a restriction.) This allows memory reference, Jump, and Call instructions to not have to carry the cumbersome 16 bit address with it. These instructions can now be compacted into 2 bytes rather than the 3 bytes found in microprocessor coding.

Several additional characteristics of the instruction set are important for code efficient control. Many control applications require only one bit wide I/O; switches are read, relays and lamps are driven. Thus the bit manipulation capabilities of the instruction set are important. The instruction set should be able to set and reset individual output port bits and to read and test individual input port bits.

Some controllers are often called upon to drive displays and read keyboards. Therefore, the instruction set should have binary and BCD arithmetic capabilities. Decimal adjust and nibble swapping instructions provide an efficient means of handling BCD data.

The third control characteristic is timing control. Timing control involves two functions: sequencing and event counting. In sequencing, the controller generates accurate time intervals or pulse streams to replace cams and gears in mechanical systems, or for external timing to multiplex displays, etc. In event counting, the controller is used to count events or to keep track of an external timing source such as 60Hz for time-of-day computations or where long time intervals are to be maintained. In most microcomputers, the timing control function is implemented with a programmable on-chip timer/counter. Having the timer on-chip decreases parts count and alleviates the need for time consuming software delay loops and port polling.

Going almost hand in hand with timing control is interrupt capability. Timing

control frequently involves interrupting the processor to execute a specific task, i.e., time-of-day computation. Also, since single chip controllers are usually used in real-time asynchronous systems, interrupt capability is often mandatory to respond quickly to asynchronous events. Interrupt capability is code efficient since it eliminates the need for lengthy software polling loops.

There are times when neither the on-chip memory nor I/O is sufficient for the application. For these times, an expansion bus should be provided. This bus should typically have a multiplexed address and data structure to save valuable package pins. It should provide a microprocessor-like structure to allow interfacing to standard RAM and ROM/EPROM. It should also allow the use of intelligent peripherals such as USARTs and PPIs to supply functions the controller does not have time or code space to implement.

The final characteristic is the one which has the greatest impact on the hobbiest: compatible ROM and EPROM versions. At least two manufacturers are producing, or are about to produce, EPROM equivalents of their ROM controllers. The ROM versions are for high volume production. However, users are usually reluctant to commit code to ROM before it has been fully debugged. To allow the complete debugging of both the hardware and software, the EPROM equivalents were developed. Now the user can develop his hardware and software using the EPROM version and know it will still work once committed to ROM. This EPROM/ROM compatibility is a significant asset to the high volume user.

It is this compatibility byproduct which is a boon to the hobbiest. Using the EPROM based single chip controllers, specialized hardware and software can be developed for applications with a volume of one. This can be done at a cost significantly lower than what would be required utilizing a microprocessor based system.

The Intel 8748

Let's look at one particular microcomputer, the Intel 8748, to see how these control oriented characteristics are actually implemented. First, let's discuss the 8748's general features.

Key features of the 8748 are:

1. Single 5v supply

2. 2.5 µs cycle time - all instructions

execute in 1 or 2 cycles

3. 96 instructions - 70% single byte

4. 1Kx8 UV erasable/programmable program
    memory - single pulse programming
    and single location programming

5. 64x8 RAM data memory

6. 26 I/O lines

7. 8 bit interval timer/event counter

8. single level interrupt

9. easily expanded memory and I/O -
    multiplexed bus

10. on-chip clock generator - xtal, or RC,
    or LC

11. single step function

The 8748's resident program memory consists of 1024 words 8-bits wide which are addressed by the program counter. This memory is UV erasable and user programmable. Individual words are programmed with a single programming pulse. There is no restriction on how many words may be done during a programming session. Three locations in the program memory have special importance; locations 0, 3, and 7. Reset vectors the program counter to location 0. An external interrupt vectors the program counter to location 3. And a timer interrupt causes a vector to location 7.

While the resident memory is 1K bytes, the 8748 program counter allows expanded addressing up to 4K bytes. External ROM/EPROM furnish the additional memory by way of the expansion bus which is discussed shortly.

The resident data memory is organized as 64 words by 8 bits. All 64 locations are indirectly addressable through either of two RAM pointers which reside at addresses 0 and 1 of the RAM array. In addition, the first 8 locations (0-7) of the array are designated as working registers. These registers are directly addressable and are usually used to store frequently accessed intermediate results. Locations 24-31 are also designated as a second set of working registers. These two sets of working registers are selected using a Register Bank Switch instruction. This second bank may be used as an extension of the first bank of may be reserved for use during interrupt service routines. The first two locations in the second bank also serve as RAM pointers for indirect addressing.

Of course, if the second bank is not used, those locations are still addressable as general purpose RAM. RAM locations 8-23 serve a dual purpose in that they contain the program counter stack during subroutine calls. This provides a maximum nesting of 8 subroutine levels. If less than 8 levels are used, the remaining stack locations are available as general purpose RAM.

Since the RAM pointers are 8 bits wide, they are capable of addressing 256 RAM locations. If necessary, this additional RAM may be easily added thru the expansion bus.

The 8748 contains 26 lines which can be used as either input or outputs. These 26 lines are arranged as 3 8-bit ports plus 2 test inputs. Ports 1 and 2 have identical characteristics. They are called quasi-bidirectional. This structure allows each individual line to serve as an unlatched input, a latched output, or both, even though the outputs are statically latched. The third 8-bit port is BUS. BUS is a true bidirectional port with associated input and output strobes. If bidirectionality is not needed, BUS may serve as either a non-latching input port or a statically latched output port. Input and output cannot be mixed as with the other ports however.

BUS also forms the expansion bus allowing interface to external RAM, ROM/EPROM, and peripherals. Four control signals are provided: RD (Read), WR (Write), PSEN (Program Store Enable), and ALE (Address Latch Enable). When used for expansion, the BUS lines (DB0-7) are multiplexed with address and data. ALE provides a means for external circuitry to de-multiplex the bus when standard RAM, ROM/EPROM, and peripherals are used. Multiplexed combination peripherals such as the 8155 Combination RAM/I/O/Timer and 8755 Combination EPROM/I/O, interface directly. PSEN is used to enable external program memory. RD and WR are the normal control lines for RAM and peripherals.

The T0 and T1 pins serve as testable inputs. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. T0 also serves as a clock output whenever an ENT0 CLK instruction is executed. T1 serves as the event counter input when the on-chip counter is used in that mode. When neither of these special modes are being used, T0 and T1 are general purpose testable inputs.

The INT (INTerrupt) pin is another test-able input. It may serve a function identical to T0 and T1 or it may be used as an interrupt input in the traditional sense. In the interrupt sense, activating INT causes a "jump to subroutine" to location 3 in the program memory. As in any CALL, the program counter and status word are saved on the stack. Location 3 generally contains an unconditional jump to the interrupt service routine. At the completion of the routine, a RETR instruction is executed to return the program to its pre-interrupt location. Of course, the instruction set contains instructions to disable and enable inter-rupts under software control.

The final hardware element to discuss is the on-chip timer/counter. The 8-bit counter is presettable and readable through the accumulator. Instructions are provided to load, read, start, and stop the counter. The counter contains an over-flow flag. This flag is set whenever the count raps around from FFH to 00H. The flag may be tested or used as an interrupt source. A counter interrupt vectors the program to location 7 in the program memory. The counter interrupt output is internal-ly OR'd with the external interrupt pin, INT. The counter can be configured to operate as either an event counter or as an interval timer. These modes are selectable via the software. In the event counter mode, the T1 input supplies the events to be counted. In the inter-val timer mode, the internal clock (ALE) is prescaled by 32 and is used as the counter input. In either mode, the load, read, start, and stop instruction control the operation of the counter.

Now that the hardware has been discussed, let't look at a small subset of the instruction set which illustrates the 8748's instruction set efficiency in control applications.

The 8748 contains a comprehensive set of single bit manipulation instructions. These instructions apply to the accumulator as well as the output ports. The output port instructions are representative. To manipulate a bit in an output port, there are AND and OR port instructions. These instructions utilize a mask to define which bits are to be set or reset. For the AND instruction, the mask contains a 0 where a bit is to be reset and a 1 where no change is desired. In the OR instruction, the mask contains a 1 where a bit is to be set, and a 0 for no change. Thus individual bits in any port being used for output may be set or reset using these instructions. For input port bit testing,

the port is first read into the accumula-tor. The instruction set includes instructions to jump on the condition of any bit in the accumulator. One of these instructions is then executed. Thus the condition of switches can be read and the appropriate action taken. Also included is an instruction for n-way branches based on the contents of the accumulator. The contents of the program memory location pointed to by the accumulator are sub-stituted for the lower 8 bits of the program counter. This results in an indirect jump within a page of program memory.

Loop control is accomplished with the DJNZ instruction. This instruction implements a decrement and jump if not zero function. It may be applied to the contents of any working register. This instruction saves code space and shortens the timing spent in various software loops.

The MOV P3 A,@A instruction eases look-up table handling. This instruction uses the accumulator as the offset to fetch a byte from page 3 in the program memory. This effectively allows the CPU to "look-up" the corresponding code (an ASCII character, for instance) in a single, code-efficient operation.

Special instructions are provided for BCD and binary arithmetic. The DA A instruction decimally adjusts the accumu-lator. SWAP A swaps the two 4-bit nibbles of the accumulator. And the XCHD instruc-tion exchanges the lower nibbles of the accumulator and any working register.

This completes our discussion of some hardware and software details of the 8748.

Application Example

An application example was given as part of the presentation.

Conclusion

This presentation has hopefully been an introduction to the relatively new world of single chip microcomputers. We have covered the general characteristics desirable in single chip microcomputer used in control oriented applications. And as an example of one such controller, the Intel 8748 was discussed.

THE DISYSTEM: A MULTIPROCESSOR DEVELOPMENT SYSTEM
WITH INTEGRATED DISC-ORIENTED INTERCONNECTIONS

Claude Burdet, Systemathica Consulting Group, Ltd.
4732 Wallingford St., Pittsburgh, PA, 15213, (412)621-8362

Abstract

The microcomputer architecture of the DISYSTEM features innovative solutions to several limitations--8-bit capacity, 64K maximal memory, fixed addressing--of the 8080 microprocessor family. The modular design of the mainframe is especially suited for implementation of business and industrial applications on turnkey systems. It also furnishes a possibility to expand an existing microcomputer system into a powerful parallel processing configuration which will, in effect, perform 16-bit multi-byte operations.

Section 0.1 Overview

This article describes the general structure of a new type of microprocessor based computer system; one of its distinguishing features lies in the use of two processors which communicate through a linkage module to become a versatile dual CPU, operating in parallel processing mode, or independently from one another in multiprocessing mode (see Appendix A).

The building blocks of the DISYSTEM mainframe are described in Section 1: two single board computers, and a communication board comprising linkage and arbitration modules with the FD controller.

Section 1 also presents a special memory design for improved compact storage and better usage of memory banks through dynamic addressing. A 24-bit addressing scheme increases direct RAM/ROM access to a maximum capacity of several megabytes, along with a sharing scheme for access of the same memory banks by both CPUs.

In Section 2 several typical hardware configurations are listed, illustrating the modular versatility of the hardware. A fully expanded DISYSTEM operates as a virtual machine using special system commands to define different types of operating configurations of the dual processor system: two independent CPUs, one parallel processing dual unit, or various Master-slave multiprocessing configurations. A multiprocessing flow chart illustrates the virtual machine's ability to change structure at each instruction; these special commands are used to define the state of the virtual machine and can be viewed as an extension of the instruction set of the 8080.

The DISYSTEM linking approach introduces new programming possibilities in the 8080 family. With its two CPU chips the virtual machine is a 16-bit computer whose characteristics differ from those of conventional 16-bit microprocessors--in fact, depending on the application and the software, double 8-bit parallel processing can be superior to 16-bit arithmetic.

0.2 Introduction

The DISYSTEM was originally developed at Systemathica to fill a gap in the computer industry. Off-the-shelf microcomputer systems do not have the kind of power and hardware versatility required for OEM development of turnkey systems in the small to medium size range. The 8080A is chosen because it is a well known chip with the best speed-to-cost ratio; as a result, the DISYSTEM delivers minicomputer performance at the price of micros.

A wide range of hardware options are built into each DISYSTEM board, so that the same standard element can be used for a variety of different hardware applications without additional interface boards and modifications. This contributes to hold turnkey system development costs to a minimum and to guard against premature obsolescence. For example, DISYSTEM memory units are designed to operate with either 8-bit or 16-bit systems so that the substantial investment represented by a large memory unit can be preserved when a system is upgraded from an 8-bit to a 16-bit CPU.

The next objective of the DISYSTEM project is one of technical innovation for a low cost system with features found only in large systems, putting

the accent on three areas: intelligent control of peripheral devices (see Appendix B), increased size and better utilization of memory space, and faster number-crunching and file management by the processing unit.

The result is: a new single board computer concept with larger memory and I/O capability; a multi-megabyte memory box with dynamic addressing; a dual, parallel-processing, Central Processing Unit which simultaneously operates two microprocessors; and a super-intelligent Floppy Disc controller with built-in I/O module and DMA for temporary storage of large files.

The question of hardware and software compatibility is one where Systemathica differs most from other manufacturers. DISYSTEM architecture is largely universal: it interfaces directly with a variety of different bus structures, such as Intel's MDS or SBC, and S-100 standards. The same philosophy applies to the software. In addition to software packages specifically developed for the DISYSTEM to take advantage of its parallel- and multi-processing capability, any software written for the 8080 family will run on DISYSTEMS including popular disc operating systems such as ISIS or CPM.

The motivation behind the development of the DISYSTEM is a natural consequence of the need for better overall performance of small but relatively powerful systems (usually disc oriented). The intrinsic speed of a processor is often much less relevant than one assumes; in an interdependent system, speed is limited by the slowest component, and this is almost never the processor itself. I/O devices are much slower and should be given particular attention--floppy discs also fall in this category. Overall processing efficiency of a disc-oriented system therefore requires at least two processors: a Host CPU for main processing and number crunching, and a sloave CPU for handling the peripheral devices. Microcomputers presently offered on the market are either of the single processor kind or contain a second processor which is permanently dedicated to a specific peripheral task. The DISYSTEM architecture, on the other hand,maintains complete multiprocessing symmetry for increased programming flexibiliby, including the possibility to simultaneously service several I/O devices (viz. floppy disc, terminal and printer). This introduces a much higher level of peripheral intelligence than could be attained with a dedicated

slave processor.

The dual 8-bit processing structure of the mainframe rivals conventional 16-bit processors through its versatility: 16-bit arithmetic is obtained in the form of a double-byte word where each byte is processed on a different processor; the hardware links are used primarily to convey "carry" signals. Alternately, the dual machine can be set up for parallel programming where both processors may execute different operations while communicating through an 8-bit channel. This dual CPU displays more than twice the power of a single CPU chip due to new software options arising from the byte oriented link which is, in fact, a new register shared simultaneously by both microprocessors.

DISYSTEM hardware also interfaces other existing microcomputer systems; small system owners can upgrade their facilities towards 16-bit processing and intelligent peripheral control without fundamental changes to their present installation.

Section I   The DISYSTEM

The overall architecture of a DISYSTEM is shown in Fig. 1. It features two main processors which function as independent microcomputers, sharing the same floppy disc and memory; alternately, each processor can operate as a slave to the other, converting the floppy disc controller into a super-intelligent device. Configuration changes are under software control, and processor assignment can be dynamically reconfigured using commands of the DISYSTEM extended instruction set. Thus the dual CPU operates as a virtual machine which can become any of several machine tvpes through an internal command. The following configurations represent the basic options which can be chosen by the virtual machine for either CPU (see Section II for an example).

I.1   DISYSTEM Architecture

The single-board computer concept is critical to the multiprocessing design of the DISYSTEM. The various standard configurations listed below reoresent different phases defined by the software during program execution.

Configuration C1) This is the basic single board processing configuration. A micro-unit operates alone with its 8K RAM/ROM on-board memory

and through its on-board I/O. The main bus is cut off by "floating" the corresponding drivers to the high impedance state. In this mode the micro-unit will typically perform minor satellite processing tasks such as editing, formatting, or word processing.

Configuration C2) In Fig. 1 each micro-unit is connected to a memory box (shown immediately below the micro-unit). When the micro-unit controls its off-board bus, it has direct access to this memory box, becoming a fully expanded single processor microcomputer. The DISYSTEM includes two such computers which can operate independently. Communication between the two systems may occur through a latched 8-bit communication port; data transfer is then performed in handshake mode. At all times a status word can also be read by either subsystem, indicating the current status (i.e. configurations) of the entire DISYSTEM.

Configuration C3) This is the basic configuration of a disc oriented microcomputer system. A micro-unit gains access to the floppy discs by turning on the appropriate buffers of the Floppy Disc module. The FD 1771 chip automatically handles all primary disc functions, including a Cyclic Redundancy Check, while the CPU is dedicated to servicing the FD 1771. Since disc operations are slow (milliseconds range) as compared to the CPU (microseconds range), overall performance of a single CPU-disc system is reduced due to the inherently slow procedures of a disc operating system.

Configuration C4) The buses from both sides may be linked and become one by turning on the arbitration and memory cross-access buffer/drivers. To avoid conflict, one of the micro-units is then "floated" and constrained to on-board operation, while the other now has access to both memory boxes (i.e., 120K of memory). The 8080 address structure only allows a maximum of 64K of direct addressing, but a special Virtual Memory assignment module is used to expand on-line memory addressing. This also permits a DMA type of data transfer between the CPUs of (up to) 32K in less than 10 microseconds, i.e., more than 100,000 times faster than a conventional memory data transfer.

Configuration C5) Here one CPU has both memories and the FD controller

(i.e. Configuration C3 and C4 combined), while the other CPU is in Configuration C1.

DISYSTEM hardware is flexible enough to incorporate other microprocessor systems as one or both of its linked CPUs. Configuration C3, for example, is a super-intelligent disc controller which can be connected to an existing microcomputer (not necessarily 8080 based).

Bus Structure There are two separate buses, linked by the arbitration board; access to and from the buses is through tri-state buffers with plenty of power for noise reduction. Each memory box also possesses an internal bus with the same characteristics.

I.2 Micro-unit (8080-MU-1001 board) see Figs. 2 and 7

This single computer comprises the following modules:

Central Processor Module Consists of an 8080 CPU set (8080 CPU, 8228/38 controller, 8234 clock driver, 2.0432 MHZ clock) and associated logic.

I/O Module Two serial ports (independent 8251 USART) each with selectable Baud rate varying between 75 and 19,200. One can operate either a 20ma current loop, an RS 232 CRT terminal, or both simultaneously. The I/O also contains six independent parallel 8-bit ports organized in two 8255 chips. A pair of 8216/26 buffer/drivers is provided for either output or input buffering.

RAM/ROM Memory Module With up to 4K of static RAM (21L02) and up to 4K of ROM (2708), the micro-unit is well equipped to perform most satellite processing tasks. The memory module also contains an automatic start-up relocation submodule which sets the on-board 8K block of memory at any of the 00, 20, 40, 60, 80, A0, C0, or E0 address boundaries. ROM is always relocated to the top, RAM to the bottom of the 8K page so that all on-line RAM remains continuous.

Bus Buffer/Driver Module All address, data, and control lines entering the micro-unit are buffered; all signals leaving the micro-unit are boosted by powerful drivers (8097 type or 8216/26 with a fan-out of up to 30

TTL loads).

I.3 Memory Box (for 8- or 16-bit words, up to 128K words) see Fig. 6

The DISYSTEM contains two 64K RAM/ROM memory boxes which may be organized either as two (physically) separate subsystems or as one single unit totaling 128K. In any case, RAM is placed on 16K boards (2102-M-816) where each 8K RAM block can be individually assigned to one or the other CPU. A single system command will reconfigure the entire memory so that, in effect, each CPU has direct access to a 128K workspace.

A memory box plugs into the address and data bus of either side of the DISYSTEM (see Fig. 1); CPU #1 accesses memory box #2 through the memory cross-access module of the arbitration board; similarly for CPU #2 and memory box #1. Thus, each CPU has 64K of natural memory and an additional 64K of "virtual" memory. Virtual memory allocation is not limited to 128K: several megabytes of RAM or ROM can be appended in this manner, connecting several memory boxes in parallel on either side. This spacious memory design has been developed to allow future use of new memory technologies, such as low cost bubble devices. It also allows several (more than two) systems to share a joint ROM source or a 64K joint RAM workspace (scratch pad).

I.4 Mother Boards (2102-M-107X) see Fig. 6

DISYSTEMS are usually mounted on two universal mother boards which accommodate 4 memory boards and 4 system boards. Memory boxes are also available as separate units (a 64K unit or a 128K unit), with or without virtual memory logic which is wired onto the mother board.

The system mother board is universal in the sense that any bus line can be (jumper) rerouted to any other pin of a connector so that DISYSTEMS may incorporate boards from other manufacturers (0.156-inch spacing). A wire-wrap area and up to 30 (16-pin) DIP positions are provided for user-designed system implementations or signal conversion. The main purpose of this universal mother board is to ensure compatibility with other microcomputers. It will, in particular, allow one to generate S-100 bus signals for direct interface of a memory box, a floppy disc controller, or a DISYSTEM Configuration C3 (super-intelligent con-

troller) with an S-100 microcomputer. There are also additional slots between standard connector positions; thus, if space and ventilation are not a problem, DISYSTEM mother boards will accept additional boards.

I.5 Arbitration and FD Controller Board see Figs. 3 and 4

FD Controller Module Tri-state drivers govern the access to the controller's internal bus from each CPU. The controller is seen by the CPU as an I/O device which can be referenced as B0 to BF; all special commands to the FD 1771 are triggered by such I/O commands. All signals to the disc drives are powered by open-collector logic.

Status Latch Module A set of 10 fully independent flip/flops are used to record the current status of the controller. It is collected in one byte of information which can be inspected by either CPU at their request (see Table 1). There is also one pair of flip/flops attributed to each CPU for handshake communication.

Data Latch Module Data communication between both microcomputers is handled by a pair of 8-bit latch/drivers. In interrupt or handshake mode, each byte is latched by one micro-unit and subsequently read by the other. In cross-access DMA mode, the ports act as transparent buffers. All system control signals are accompanied by a message byte which is automatically latched in the data communication ports, at the disposal of the other micro-unit (see Table 2).

Data Separator The FD 1771 possesses an internal data separator which may be used; however, this is not recommended and, for added reliability, the FM serial data/clock line is presented to an external data separator in the controller module.

I.6 Interrupt Structure

The interrupt structure of the DISYSTEM is completely symmetrical and under software control by either CPU. There are three internal interrupt sources: CPU #1, CPU #2, and the controller chip FD 1771. Over 200 internal and external interrupt calls may be serviced. Interrupts between CPUs may

be inhibited by a system command in order to assure absolute priority to the floppy disc controller during disc data transfer. Interrupt calls and acknowledgements are recorded in the status latch, so that no ambiguity may occur. In addition to the hardware signal, each interrupt is accompanied by an 8-bit message (latched by the caller), which contains the priority and identification of the requested service. Each CPU can entertain over 200 individual interrupt calls with detailed priority management and associated service vectors. Entry points are listed in a 512 byte jump table (in RAM or ROM) which is itself directly referenced by the (RST 7) hardware call (see Fig. 5). Since it is controlled by two processors (not merely an interrupt controller chip), this scheme can easily be cascaded to accomodate arbitrarily many interrupt services. The DISYSTEM interrupt structure also accomodates interrupt signals from external devices. Such interrupt calls are presented to the same priority manager whose individual service vectors eliminate the need for separate interrupt management.

Section II   Operations

During the course of its operations a DISYSTEM will continuously change structure, assuming any of the Configurations C1-C5; a system command is required for each change of configuration. Perfectly symmetrical, this architecture replaces the more conventional master-slave configurations with dedicated CPU, offering more possibilities for hardware and software design to increase the efficiency of the total system. The example described here shows how the virtual machine evolves: the Host (CPU #1) communicates with a complete disc operating microcomputer slave. Thus the main processor need not be concerned with file management since his requests to the slave subsystem can be issued at the file name level. Furthermore, virtual memory allocation allows the Host to access disc files in a time comparable to RAM access time. All DISYSTEM commands have I/O names and no memory location need be reserved for internal activities.

II.1   Example (see Appendix C, Table 3)

Consider the following installation:

Peripherals   serviced by CPU #1:

-CRT at 9600 Baud
-Cassette tape unit at 300 Baud
serviced by CPU #2:
-Teletype at 110 Baud
-Line printer at 4800 Baud

The teletype and printer are serviced on a time sharing basis, similarly for the CRT terminal and the tape unit.

ROM   Both ROM #1 and ROM #2 are automatically relocated to page F0-FF (last 4K).
-Rom #1 contains a start-up monitor and utility routines, cassette tape operating system, and a smart CRT editor; the 4K onboard RAM buffer is used as a scratch pad (that's more than two full CRT screens!).
-ROM #2 contains a start-up monitor, and a printing editor for both the teletype and the line printer; thus micro-unit #2 can operate as an independent satellite with a 4K on-board RAM buffer.

System RAM   #1 memory box = main programs and workspace area.
            #2 memory box = disc operating system and file storage area.

A typical multiprocessing chain of operations is described in Table 3; in its initial status, the virtual machine consists of two independently running CPUs, each with its own memory, i.e., as in the parallel programming Configuration C2. Numbers in ( ) indicate the sequence of events.

This example also presents an illustration of virtual memory operations. The memory area ADDR occupies the same physical location (in memory box #2) during the entire process, but it is referenced either as natural memory by CPU #2 during phase 4 or as virtual memory by CPU #1 during phase 7. Instead, a double DMA operation could be made by CPU #1 to transfer the data block ADDR into memory box #1 and back. The method used in the example, however, is much faster, as CPU #2 is engaged in on-board processing during phase 7.

Disc files appear, to the Host, to reside in RAM. The only requirement for this is that the fetch request be made in advance; but in view of the fact that the FD controller has access to as many memory banks as needed, this is no severe restriction--disc file requests can be timed so that CPU #2

has enough time to load the file into virtual memory before CPU #1 issues an actual data reference within that file. File access time now becomes essentially the same as RAM access time. Thus, the use of virtual memory banks has much the effect of converting a floppy disc into fast memory.

In Table 3, communication between CPUs is interrupt driven, and multi-level nested interrupt calls may occur during multiprocessing. But the processors will automatically resume their "interrupted" task immediately after the interrupt has been serviced, so that little overhead is required to control the flow of operations.

The "program" is a straightforward application of the improvements built in the DISYSTEM. It shows that virtual memory operations eliminate lengthy memory-to-memory data transfers, and that to a large extent they give a RAM flavor to disc files. The importance of such improvements need not be emphasized to the user of disc oriented microcomputers.

This example was chosen because it demonstrates the super-intelligence of the FD controller system, i.e., the half of the DISYSTEM which is in Configuration C3. In the DISYSTEM, it is linked to a Configuration C2, but, as such, it will yield the same improvements when combined to any microcomputer system.

Further multiprocessing programming details, parallel programming routines, multibyte arithmetic functions, and other software possibilities of the DISYSTEM are described in Reference 3 . Some standard software packages are furnished with DISYSTEMS allowing the user to take advantage of the parallel and multiprocessing mode. Application software packages include large scale linear programming, simulation programs as well as business programs. DISYSTEM related software and programming methodology can be obtained directly from the manufacturer.

Conclusions

Building a system with off-the-shelf components is a difficult task as soon as a certain level of complexity has to be reached; many microcomputer boards are readily available on the market, but hardware compatibility is not as pure as magazine ads make one believe. The major difficulty lies in the system design area. Off-the-shelf boards are fine as "add-on" products,

but a collection of them still does not constitute a sound concept for a well integrated modular architecture.

The DISYSTEM architecture follows the opposite philosophy: it is one integral concept which is capable of major extensions in several directions: memory, central processing, discs or other I/O devices. System expansion represents no more than an implementation of options already built into the hardware and the system commands.

The development effort centered around this principle has proved successful: the DISYSTEM is one of the lowest priced systems, even when compared to less ambitious personal computers; and yet, an expanded version will hold its rank in the field of minicomputers.

The prices listed below convincingly demonstrate this fact.

The 8080A-MU-1001 single board computer is available for $119.00 kit, sockets are $15.00 extra; assembled $1500.00; the additional chip set with CPU, 1K UVPROM, RAM, serial port costs $110.00.

The 16K memory board costs $60.00, $30.00 for 16K RAM (21L02, 450ns); assembled $260.00. The 64K memory box motherboard with virtual memory logic is $58.00 (kit), $103.00 assembled; 128K is available for $93.00 kit, $185.00 assembled.

The arbitration and FD controller 1771-AC-1051 is available at $171.00 kit, sockets are $25.00 extra, $230.00 assembled; the additional FD chip set is $71.00.

Standard OEM quantity discounts apply. Delivery is 0-45 days.

Assembled and tested systems are priced as follows (disc drives and enclosures excluded):

-64K virtual memory system (450ns): $1100.00
-Microcomputer (Configuration C2) with 20K RAM: $520.00
-Super-intelligent FD controller (Configuration C3) with 36K of RAM and disc operating system: $1200.00
-DISYSTEM with 72K RAM: $1950.00
           with 120K RAM: $2600.00

Appendix A

The term _parallel processing_ is used to characterize a single program performing (possibly different) operations on several processors.

The term _multiprocessing_ is used here in the restricted sense where several machines operate independently from one another, occasionally communicating through interrupt calls, for instance.

Parallel processing example:

Scalar Product: $c = ab$

CPU #1 operations

$$c_1 = a_1 b_1$$
$$c_3 = c_1 + a_3 b_3$$
$$c_5 = c_3 + a_5 b_5$$
$$\vdots$$
$$c_{n-1} = c_{n-3} + a_{n-1} b_{n-1}$$
ship c_n to CPU #2

CPU #2 operations

$$c_0 = a_0 b_0$$
$$c_2 = c_0 + a_2 b_2$$
$$\vdots$$
$$c_{n-2} = c_{n-4} + a_{n-2} b_{n-2}$$
$$c = c_{n-1} + c_{n-2}$$

Appendix B

Intelligent means that the device is accessed through a microprocessor based controller; we will also call the device _super-intelligent_ when it is controlled by a microcomputer system with full I/O, a substantial memory, and resident operating system.

Appendix C

Table 1: Status Word

Each CPU has permanent access to the following status information:

Bits 0 and 7: a handshake command has been issued
Bits 1 and 6: handshake data is latched

Remark: Bits 0 and 1 are for data transfer _from_ the CPU, and Bits 7 and 6 are for data transfer _to_ the CPU

Bit 2: interrupt caused by the other CPU or some external device
Bit 3: interrupt stems from the FD controller
Bit 4: the CPU has access to the FD controller
Bit 5: the CPU has control over the buses on both sides

Appendix C

Table 2: DISYSTEM Bus Controls

The following I/O commands are used for

Communication between the Micro-units:

Hardware function

OUT D0: send Handshake command byte
OUT D2: send Handshake data byte
OUT D4: interrupt other CPU
OUT D6: return its natural bus to other CPU
OUT D8: disconnects FD controller from itself
OUT DA: CPU disconnects itself from its natural bus
OUT DC: disable interrupt calls from other CPU
OUT DE: unused

All these commands issue a message byte which is automatically latched in the Data Latch Module. The following commands are used to read the appropriate message as well as perform the indicated hardware function.

Hardware function

IN D0: read the status word
IN D2: read the latched data byte & clear Handshake flip flops
IN D4: acknowledge interrupt; clear the interrupt flip flops and read the latched data byte

Table 2: continued

IN D6: gain control of both natural
buses and disconnect other CPU
from its natural bus
IN D8: gain access to the FD controller
bus
INDA: disconnect other CPU from its
natural bus
IN DC: enable interrupts from other CPU
or an external device
IN DE: unused

Communication with FD controller:

Hardware function

OUT B0: send command byte to controller
register
OUT B1: send track ID   "   "
OUT B2: send sector ID   "   "
OUT B3: send data byte   "   "
OUT B4-B7: send configuration byte
(latched in disc control
latch): selects drive and
FD 1771 op code
OUT B8-BF: unused

IN B0: read FD controller command
register
IN B1: "   "   "   track
IN B2: "   "   "   sector
IN B3: "   "   "   data
IN B4-B7: force CPU into wait state,
until data request signal
from FD 1771 (an interrupt
or a reset) reactivates it
IN B8-BF: unused

Appendix C

Table 3:   A Multi-processing Applica-
tion: File update by the
super-inteligent FD
controller

The table below presents the
flow of major operations and con-
figuration changes occuring during
a "file update" procedure. This
description does not contain all
instructions in order to highlight
main events better than a complete
program listing.
It is assumed that both CPUs
are linked asynchronously; most
"acknowledge" interrupt calls would
be superfluous with a synchronous
DISYSTEM.
The label * resume processing *
indicates that the activity which
is taking place at that moment is
not directly related to the "file
update" routine.

CPU #1		CPU #2	
"Config: Timing: Operation"		"Config: Timing: Operation"	
C2	Main processing, independent or parallel mode	C2	Independent satellite processing or main parallel processing
C2:(1):	Send FD service request and assign other CPU to FD service (interrupt call)		
		C2:(2a):	Acknowledge (interrupt call)
		C2:(2b):	Grab FD controller bus (system command DB D8)
C2:(3):	Issue a "fetch FILENAME" request, with desired address location = ADDR (interrupt with message)		
		C3:(4a):	Acknowledge (interrupt call)
		C3:(4b):	Get "FILENAME" from the appropriate disc drive designated by the resident disc operating system. Store "FILENAME" into RAM location ADDR (memory box #2)
	* resume processing *		

CPU #1	CPU #2
"Config: Timing: Operation"	"Config: Timing: Operation"

CPU #2:

C3:(4c):Signal "file ready at address ADDR" (interrupt with message)
C2:(4d):Quit FD control (system command D3 D8)

CPU #1:

C2:(5a):Acknowledge; request access to virtual memory box #2 (interrupt call)

CPU #2:

C1:(5b):Acknowledge (interrupt call)
(5c):Release system bus #2 (system command D3 DA)

CPU #1:

C2:(6a):Reorganize memory #1 (system command D3 00 with message)
C4:(6b):Grab system bus #2 and gain access to virtual memory #2 (system command D3 D6)
C4:(6c):Reorganize memory #2 (system command D3 10 with message)
C4:(7a):Process and update "FILENAME" at address ADDR in memory #2 (main program)
C4:(7b):Reorganize memory #2 (system command D3 02 with message)
C2:(7c):Release system bus #2 (system command D3 D6)
C2:(7d):Signal to CPU #2 that system bus #2 is free (interrupt call)
C2:(7f):Reorganize memory #1 (system command D3 00 with message)
C2:(8): Issue a "store FILENAME" request (interrupt with message)

CPU #2:

* resume processing *

C2:(7e):Grab system bus #2 (system command DB D6)

C2:(9a):Acknowledge (interrupt call)
C3:(9b):Grab FD control (system command DB D8)
C3:(9c):Store "FILENAME" according to the directions of the resident disc operating system
C3:(9d):Signal "FILENAME stored" (interrupt with message)
C2:(11):Release FD controller (system command D3 D8)

CPU #1:

C2:(10):Acknowledge (interrupt call)

CPU #1:

C2:   Main processing, independent or parallel mode

CPU #2:

C2:   Independent satellite processing or main parallel processing

Appendix D

Captions for Figures One thru Eight

Figure 1 Architecture of the DISYSTEM

A DISYSTEM consists of two single board computers, an FD controller and linkage board, and two memory units each containing up to four 16K RAM boards; all boards are mounted on two system mother boards in a single or two separate enclosures.

Each CPU has access to the FD controller and to both memory boxes, for a maximum of 120K of RAM; each also has its own on-board I/O and memory module. Information is shared through an 8-bit latch and an 8-bit transparent communication channel. All commands of the DISYSTEM extended instruction set are executed through hardware logic for maximum speed.

Figure 2 Microprocessing Unit (single board computer)

The 8080-MU-1001 board is for stand alone operation as well as multiprocessing. It is an ideal low cost general purpose controller, with 8K of on-board memory and multiple serial and parallel I/O. Simultaneous service of several independent devices such as typewriters or printers converts them into intelligent word processors and editors. This micro-unit is available fully assembled and tested, as a kit, or a bare board, with or without sockets. It will also operate normally without all of its I/C ports and memory chips for maximum cost efficiency in applications requiring less processing power.

Figure 3 Floppy Disc Controller Module

The Floppy Disc controller has access to both CPUs and both memory units for added programming flexibility; it features its own clock generator and is capable of controlling either Standard FD Drives or Mini FD Drives. The FD controller can be given absolute interrupt priority, disabling all other interrupts. A 1771 controller chip handles all disc signals, performs automatic CRC checks, and has several formating capabilities, including IBM formats.

Figure 4 Cross-Access Arbitration Module

The arbitration module is the heart of a DISYSTEM. It combines two independent microcomputers into one central processor, operating under an extended instruction set. The resulting machine then operates either in parallel processing mode with 16-bit capacity or as an interrupt driven multiprocessor. All Address, data, and control bus lines are transmitted from one system to the other, with an additional latched 8-bit bi-directional communication channel. The arbitration module also provides DISYSTEMS with priority interrupt control for over 200 separate calls with individually vectored service.

Figures 3 and 4

The FD 1771-AC-1051 board contains both the floppy disc controller and the arbitration module. Tied to a micro-unit, it forms a super-intelligent FD controller system with its own processor, I/O, and memory.

Figure 5 Interrupt Service Structure

The Data Latch furnishes the LSB of the Jump Table address to the RST 7 subroutine (for instance 9A). In turn, the jump table delivers the MSB (for instance 22 contained in memory location F39A) and the LSB (B7 from location F49A) of an entry point (i.e. 22B7) for the requested interrupt service.

Figure 6

Memory box unit showing its four positions for 16K memory boards (2102-M-816), totaling a maximum of 64K words. A memory board is inserted in the last slot, displaying two separate 8-bit 8K banks; thus the memory unit can be configured for 8 or 16 bit operations.

Figure 7

The micro-unit 8080-MU-1001 with a maximum of 4K RAM (21L02), 4K ROM (2708), 2 serial ports (8251) and 6 parallel ports (8255). This single board computer also possesses 10 free 16 DIP positions for dedicated applications.

Appendix D continued

Captions continued

Figure 8

    The disc-oriented microprocessor
connection board FD 1771-AC-1051 con-
tains a bus linking module and data
communication channel (center), a
floppy disc control module with double
access (left), and a system command
logic module (right) which defines the
state of the dual processor virtual
machine.

References

[1] Intel MDS-800 Microcomputer
    Development System Reference
    Manual, October 1975.

[2] George A. Anderson and E. Douglas
    Jensen: "Computer Interconnection
    Structures: Taxonomy, Character-
    istics, and Examples," Computing
    Surveys, vol. 7, no. 4, pp. 197-
    213, December 1975.

[3] Systemathica, DISYSTEM Software
    Development Manual, 1977.
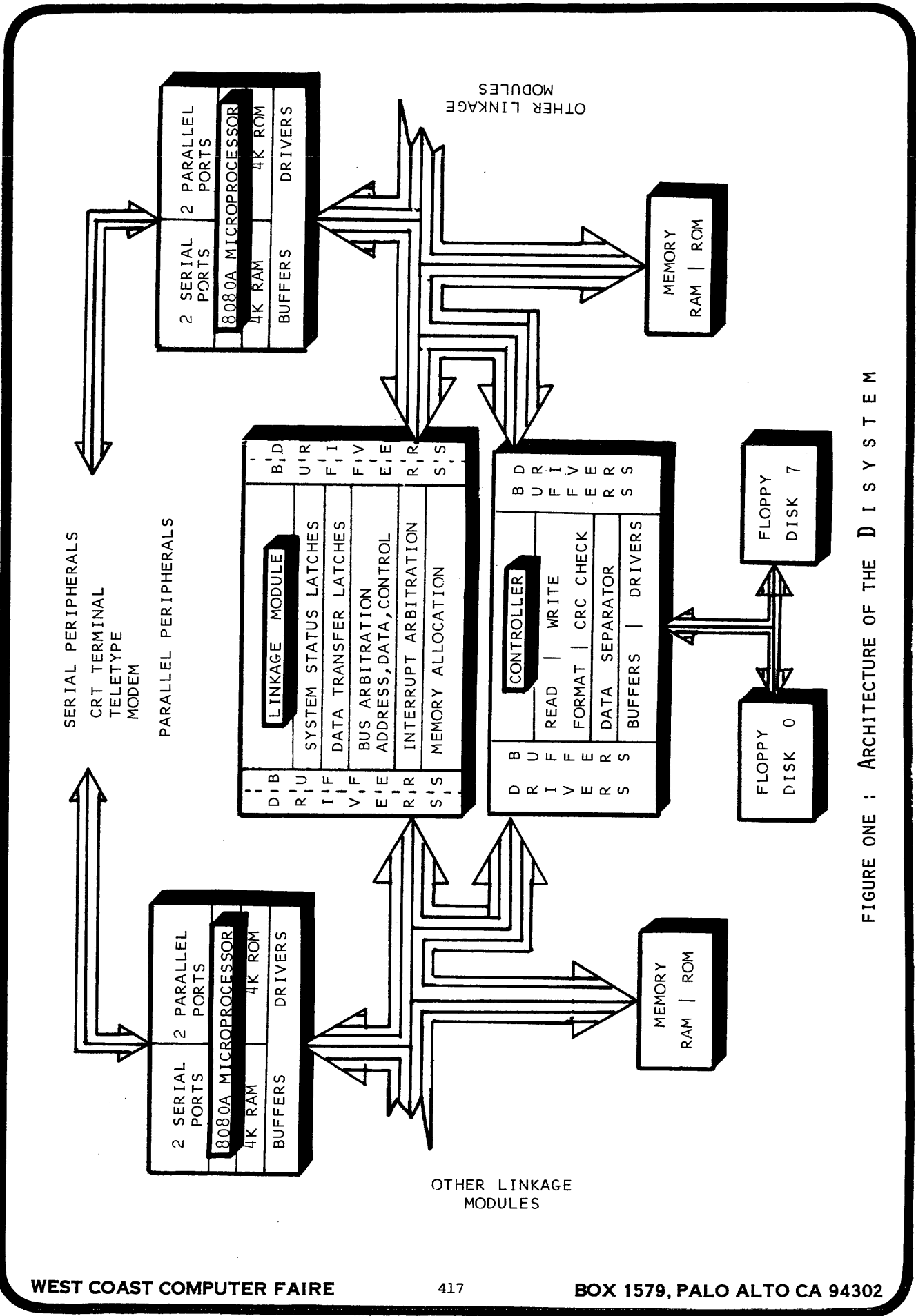
FIGURE ONE : ARCHITECTURE OF THE D I S Y S T E M
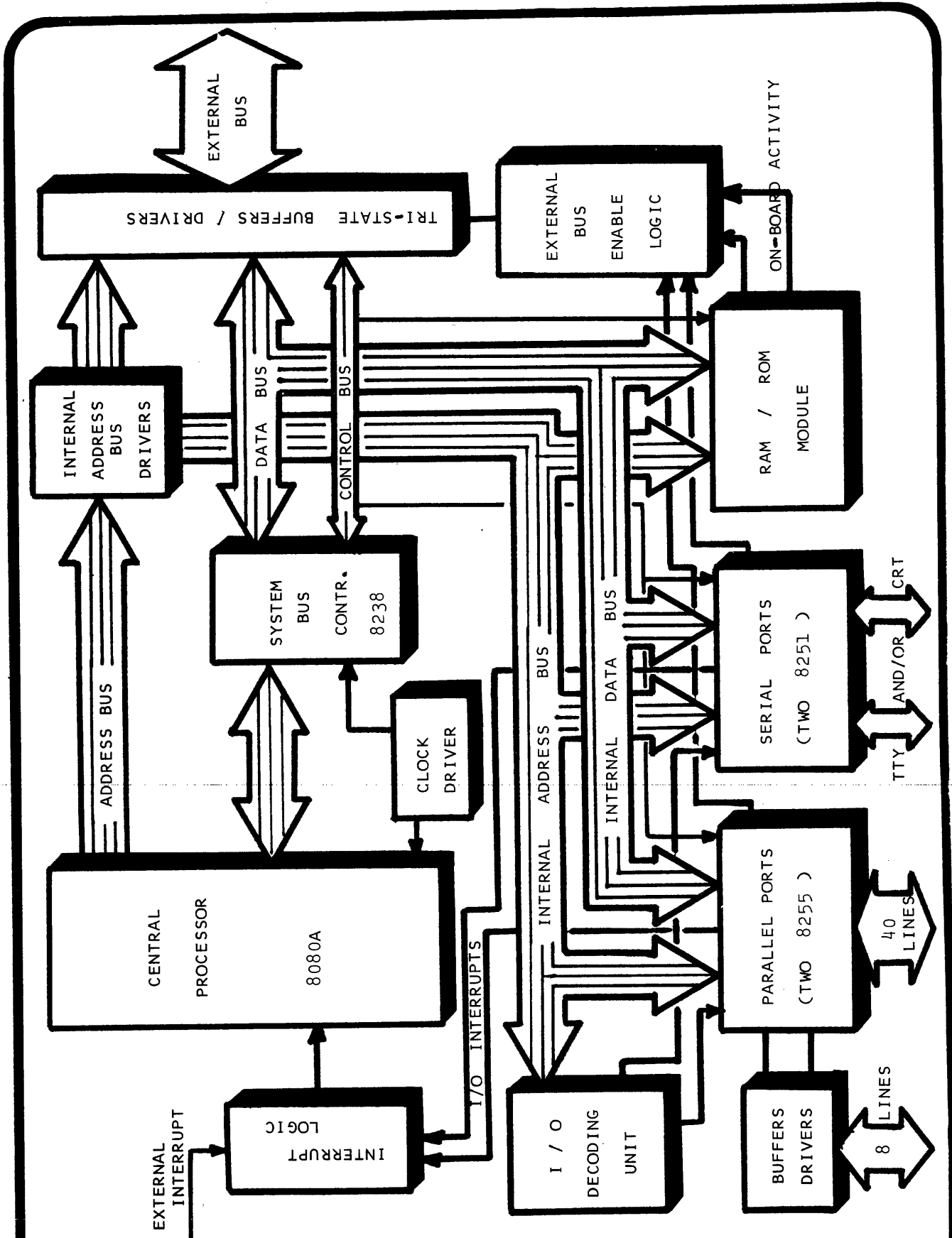
FIGURE TWO : MICROPROCESSING UNIT (SINGLE BOARD COMPUTER)
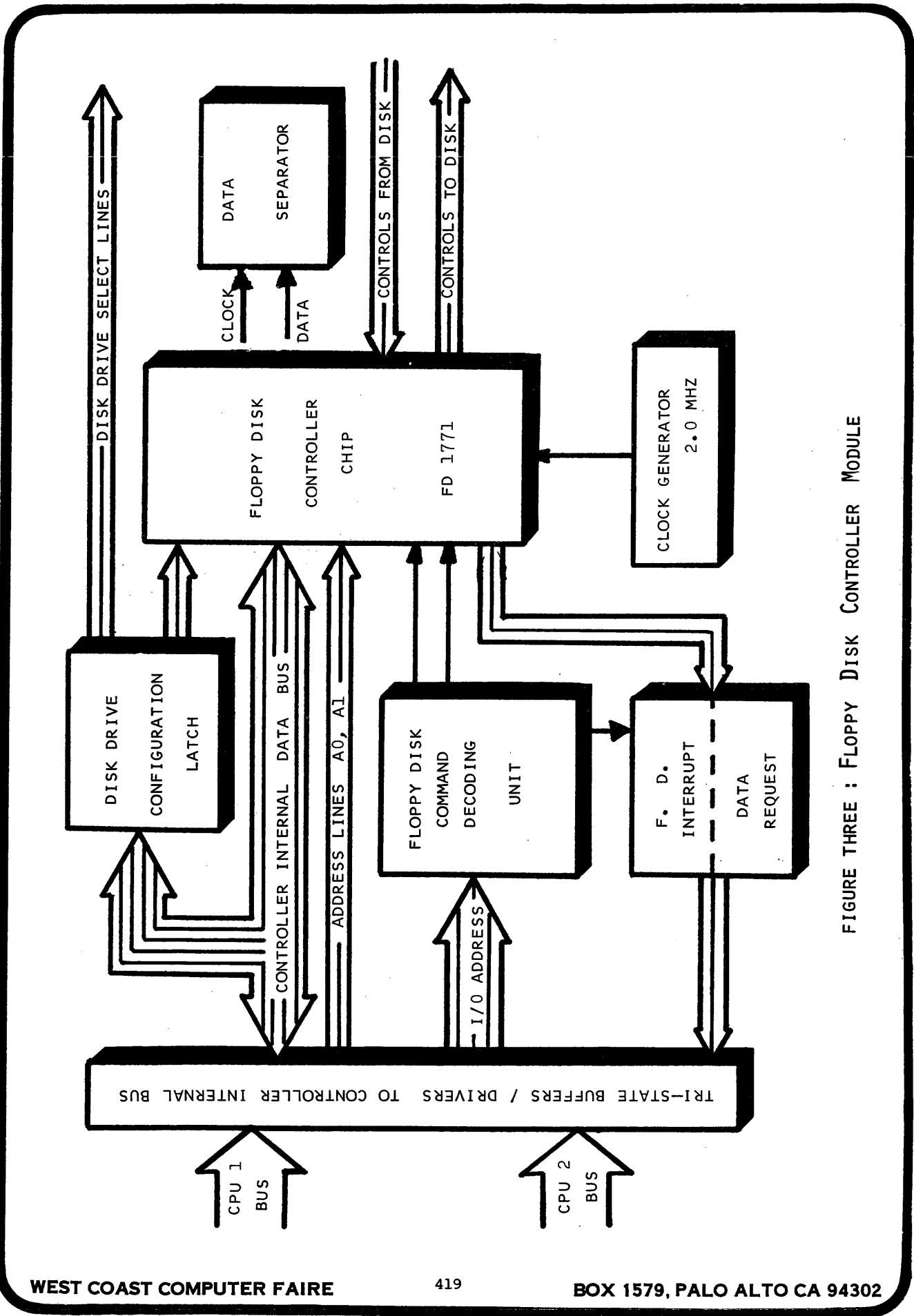
FIGURE THREE : FLOPPY DISK CONTROLLER MODULE
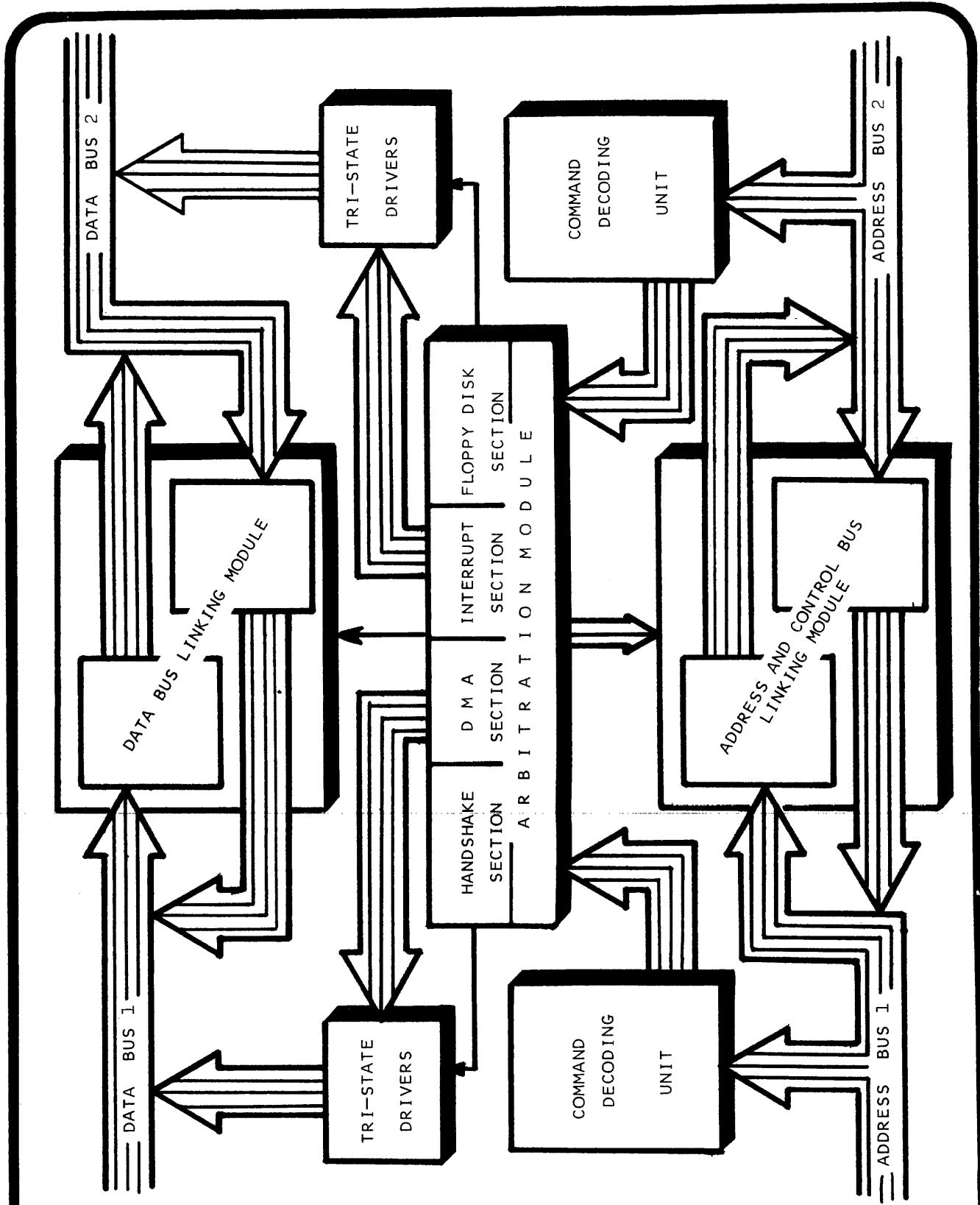
FIGURE FOUR : Cross-access Arbitration Module
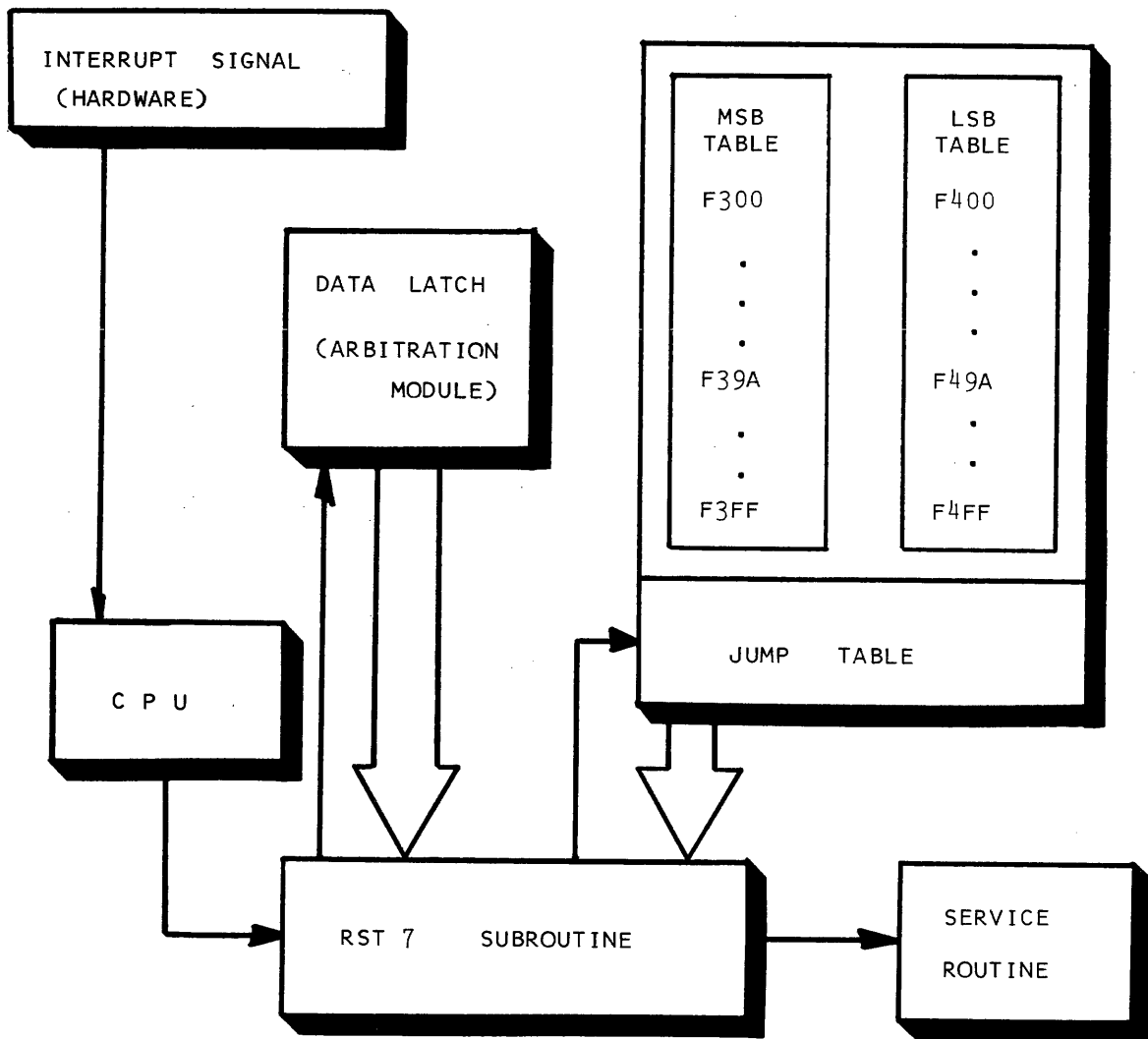
Figure 5: INTERRUPT SERVICE STRUCTURE
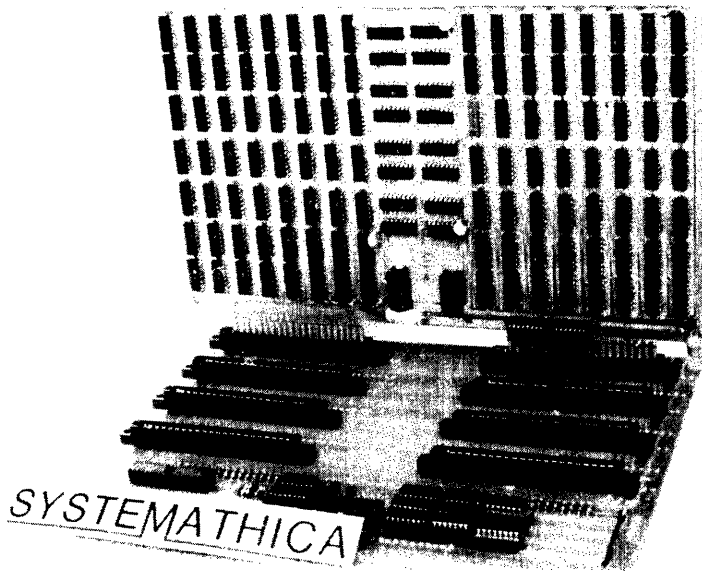
SYSTEMATHICA

C. Burdet
The DISYSTEM...
Figure 6: Memory box unit

SYSTEMATHICA
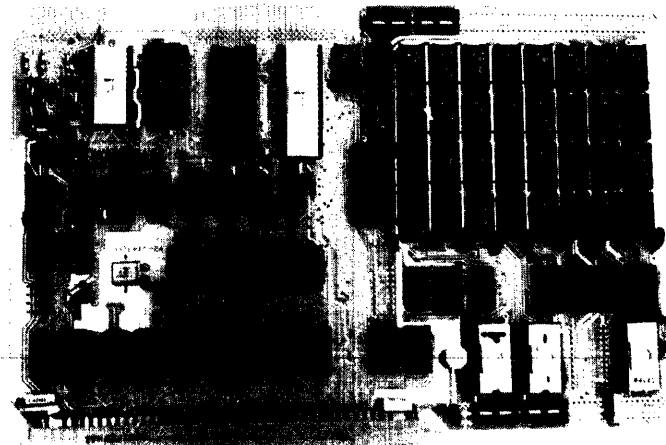
C. Burdet
The DISYSTEM...
Figure 7: micro-unit
          8080-MU-1001
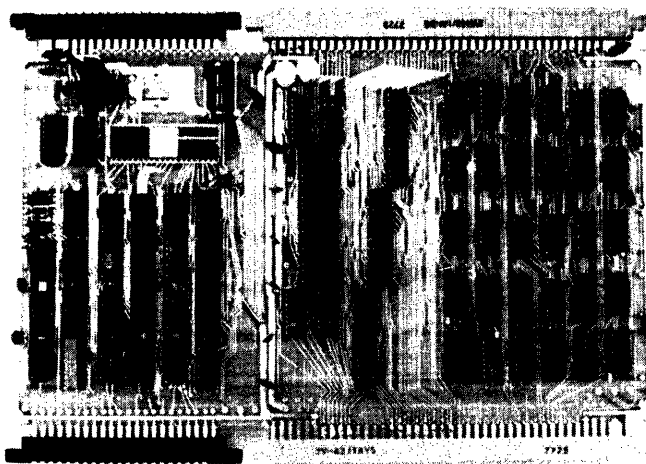
C. Burdet
The DISYSTEM...
Figure 8: arbitration and con-
          troller board
          FD 1771-AC-1051

SYSTEMATHICA

A POINT-OF-SALES NETWORK

Samuel A. Holland, Director of Research and Development
Extensys Corporation, 380 Bernardo Avenue, Mountain View, CA 94040

A Point-of-Sales network is a classic example for implementation of distributed processing architecture. This paper defines a point-of-sales environment in terms of the capabilities that are needed to effectively handle the information exchange. Next the EX1000 embodies as its primary architectural concept "distributed processing". Each of the components in the point-of-sales network is then described using the EX1000 as a basis. This emphasizes the modularity as well as efficiency of a well designed distributed processing system.

Point-of sale applications with computerized systems generally require a wide variety of hardware and software designs to provide a totally integrated system. Some of the factors to be considered in point-of-sale systems include:

● Diverse data inputs at the acutal point-of-sale: keyboards, cash registers, bar code readers, credit card inputs, interface to mechanical devices and totalizers.

● Ability of multiple point-of-sale terminals to access a regional data base with input/output capability in real-time.

● Ability to update the information bank in the regional data base from the point-of-sale terminal with current transaction data....along with ability to modify the regional data base from a corporate home office main-frame computer (pricing, inventory codes, acceptance and verification data, etc).

● Capability to batch stored information in the regional data base to a home office system on demand.

● Ability to easily add and subtract remote terminals as required.

● Human factors particularly at the point-of-sale for ease of input, accuracy, and confirmation of entry.

● Ability for the system's software architecture to allow application code to be written, entered, and modified.

● Low capital investment to more easily justify a full network with maximum capability.

● Record retention considerations.

These generalized point-of-sale requirements along with installation-specific considerations lend themselves to a distributed processing technique. Under such an architectural scheme, the full capability of the electronic circuitry and associated software can be focused on each diverse section of the overall system to optimize system capability and flexibility while minimizing cost.

The Extensys EX1000 Computer System embodies architectural concepts found, before now, only in costly large scale computers. The primary concept that has been used throughout the EX1000 is distributed processing. The EX1000 distributes system processing functions to those system components that are best suited to perform those tasks. This dedicated function type of architecture is just coming of age. It is a state-of-the-art technology which was prohibitive until the recent introduction of cost effective computational plus control microprocessor chips (costing as little as $8) along with programmable peripheral circuits to perform preassigned tasks. Coupling these cost effective products with the distributed processing techniques proven by computer companies such as Control Data Corporation, Extensys Corporation has been able to provide an extremely powerful and flexible system in the EX1000. Through the efficient use of system components and their modular structure, the EX1000 system allows system configurations that meet a wide variety of particular application needs as well as offering expansion capability to satisfy increasing usage demands.

The Extensys EX1000 Computer System is ideal for use in a point-of-sales network environment. A-1 illustrates how a point-of-sales network system could be designed using the EX1000. The use of distributed processing concepts allows for the placement of processing in those elements which can provide a highly cost effective and efficient operation. The system functions that are distributed to the various processing elements are:

● The Home Office System provides data retention and processing for the entire data base.

● Regional Systems are used for concentrating data from up to 64 (or more) terminals within a geographical region,

with area data retention as required.
● Regional Systems also use a Store and
forward system to communicate periodic-
ally with the Home Office System.
● Remote Terminals have intelligence
for editing and formatting of data.
Extensys is able to distribute the proc-
essing requirements of the network sys-
tem through the proper selection and
placement of microprocessor elements in
junction with software to perform and
control those tasks.

The Home Office System which retains
the entire data base, would normally
consist of a large mainframe computer,
such as an IBM 370/168, a UNIVAC 1110 or
a CONTROL DATA CYBER 175. These units
would have large disk storage capacity
for entire data base retention. Addi-
tionally they would be used for statis-
tical information processing and report-
ing.

The Regional Systems would consist
of an Extensys EX1000 Computer System.
A typical Regional System is illustrated
in A-2. It is composed of several re-
mote terminal processing elements, a
host or central processor, a system mem-
ory, disk storage and a local terminal
processing element. The remote terminal
processing elements can accomodate up to
8 remote terminals each. By incorporat-
ing 8 terminal processing elements, up
to 64 (or more) remote terminals could
be serviced by a single Regional System.
More systems could be added to increase
the regions capabilities even further
The flexibility to add or delete termin-
als allows for the efficient structuring
of each Regional System to meet current
as well as future needs.

The host or central processor is
used to execute application programs.
These programs would be executed in the
system memory with each terminal sharing
the same programs. System memory modul-
es can be added, providing from 16K 8-
bit bytes to 1 megabyte of RAM storage
to accomodate a wide variety of applica-
tion programs. The disk storage element
controls from 256K bytes to 2 megabytes
of on-line removable disk storage. Add-
itional disk storage units can be added
to a Regional System to expand the disk
storage capacity of a Regional Data Base.
A local terminal provides the capability
to monitor the data base and control the
operation of the system.

A Remote Terminal is comprised of
selected EX1000 processing elements. A-3
illustrates its composition. A Remote
Terminal consists of the local terminal
processing element of the associated Re-
gional System coupled with a communica-

tions modem. The Remote Terminal has
enough processing horsepower to be able
to perform local editing of information
as well as formatting and compression of
information to accomplish efficient
data transfering to the Regional System.
Software within the Remote Terminal
could step the person initiating a tran-
saction through a given set of instruc-
tions in order to provide ease of oper-
ation and accuracy of transaction.

The Flow of Transactions in the
point-of-sales network system using the
EX1000 as a basis is depicted in A-4.
Transactions are inputted at a Remote
Terminal by keyboard, credit card reader
or other convenient means. The remote
terminal with dual microprocessors and
4K bytes of buffer storage performs ed-
iting and formatting functions and con-
centrates the data prior to transmitting
it via modems to the Regional System.
The Regional System receives the trans-
actions through a modem attached to a
multiplex controller board which has a
dedicated microprocessor and buffer for
each modem. The transactions are then
combined with other transactions on the
APU-100 processing unit in an 8K byte
buffer. These transactions are then
passed to the Regional Data Base through
the central memory of the system. The
APU-100 processor then allows the tran-
saction data to be efficiently transfer-
red from the RAM buffer and stored on
the removable disk.

All transactions would be stored at
Regional System sites and periodically
or on demand would be forwarded to the
Home Office System. This would be accom-
plished by buffering large volumes of
transactions off the Regional System
disk storage through the central memory
of the system to a communications multi-
plex controller board. This board then
passes these transactions to the Home
Office System via communication lines.

The Extensys EX1000 Computer System
is ideal for point-of-sale network en-
vironments. It incorporates flexibility
through distributed processing to allow
for structuring systems to specifically
meet current needs as well as offers ex-
pansion capability as the demands on the
system increase. It provides an effic-
ient operation since only those proces-
sing elements that are required for any
operation are involved, freeing the re-
maining elements to perform other tasks.
The EMOS software, in addition to the
hardware, is distributed throughout the
complete system for effeceint handling
of transactions. As a result of the tot-
al hardware/software distribution, only

application programs would need to be
developed. This flexible, efficient
operation coupled with the cost effect-
iveness of using microprocessors makes
the EX1000 an excellent choice for a
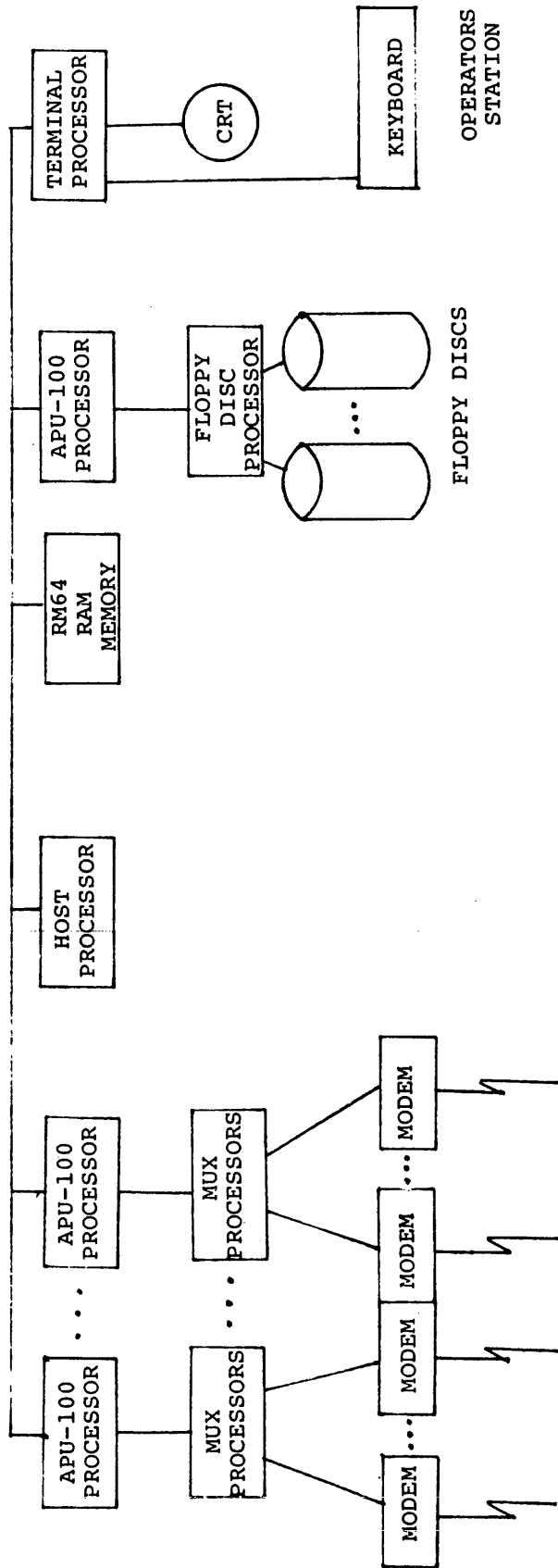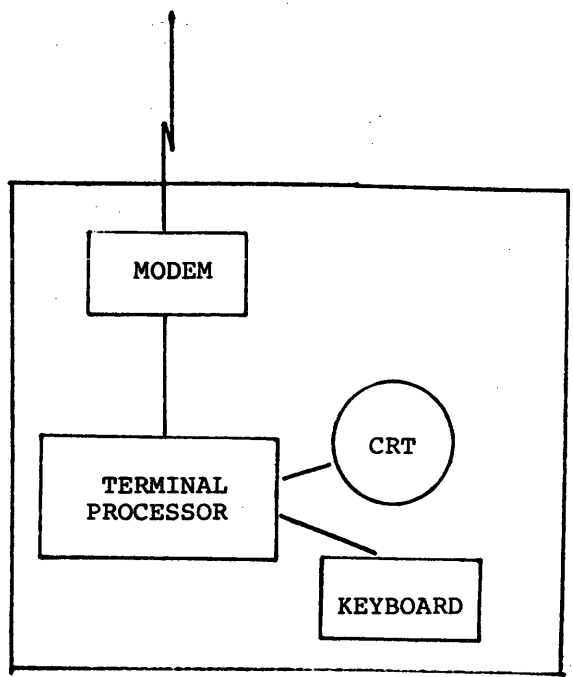point-of-sales network system.

POINT OF SALES NETWORK SYSTEM
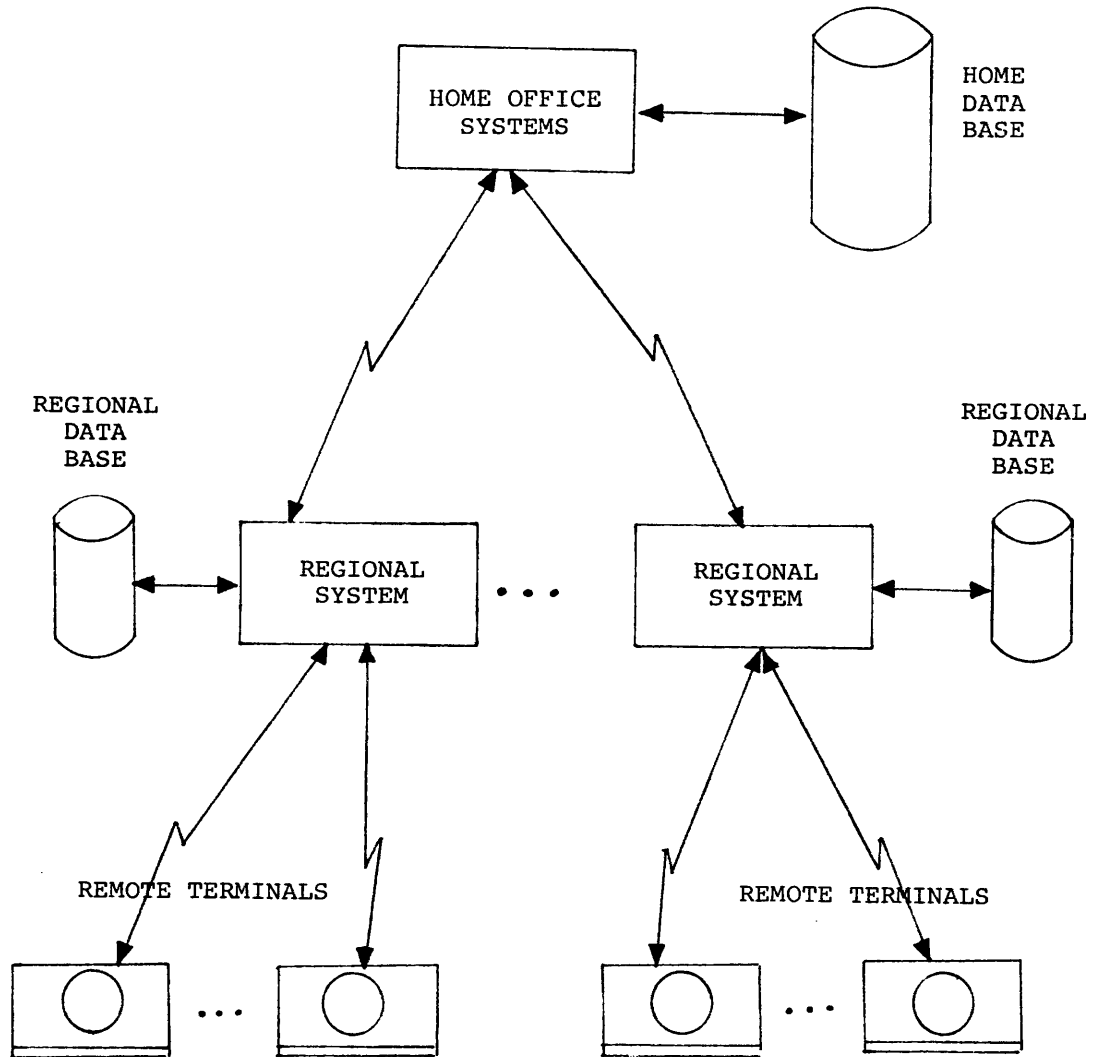
A-1

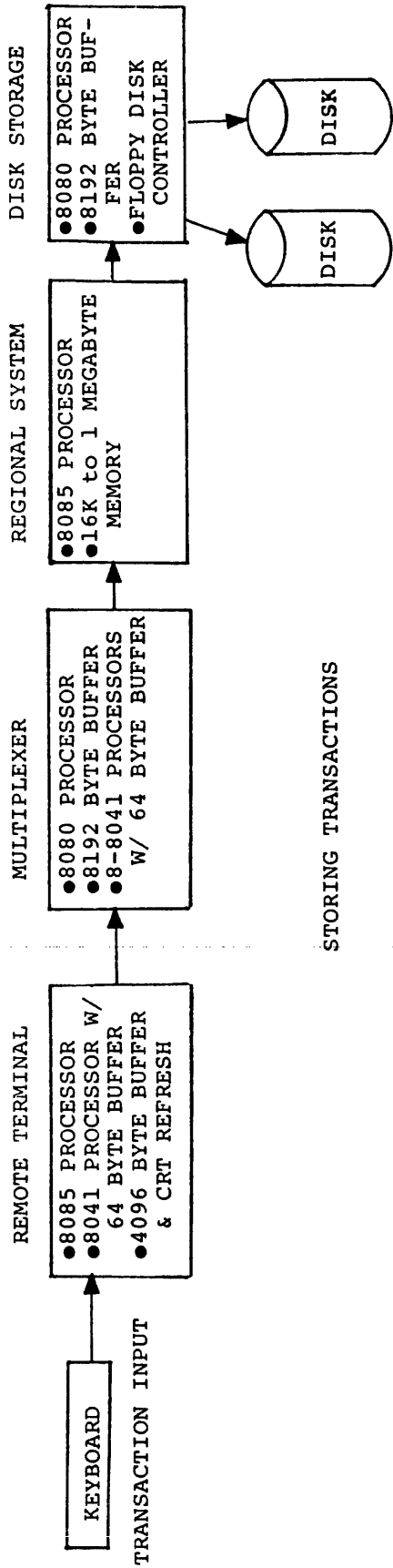REGIONAL SYSTEM CONFIGURATION

DATA BUS

OPERATORS STATION

TERMINAL PROCESSOR

CRT

KEYBOARD

APU-100 PROCESSOR

FLOPPY DISC PROCESSOR

FLOPPY DISCS

RM64 RAM MEMORY

HOST PROCESSOR

APU-100 PROCESSOR

APU-100 PROCESSOR

MUX PROCESSORS

MUX PROCESSORS

MODEM

MODEM

MODEM

MODEM

MODEM

COMMUNICATION LINES TO UP TO 64 TERMINALS

A-2

REMOTE
TERMINAL

A-3

STORING TRANSACTIONS
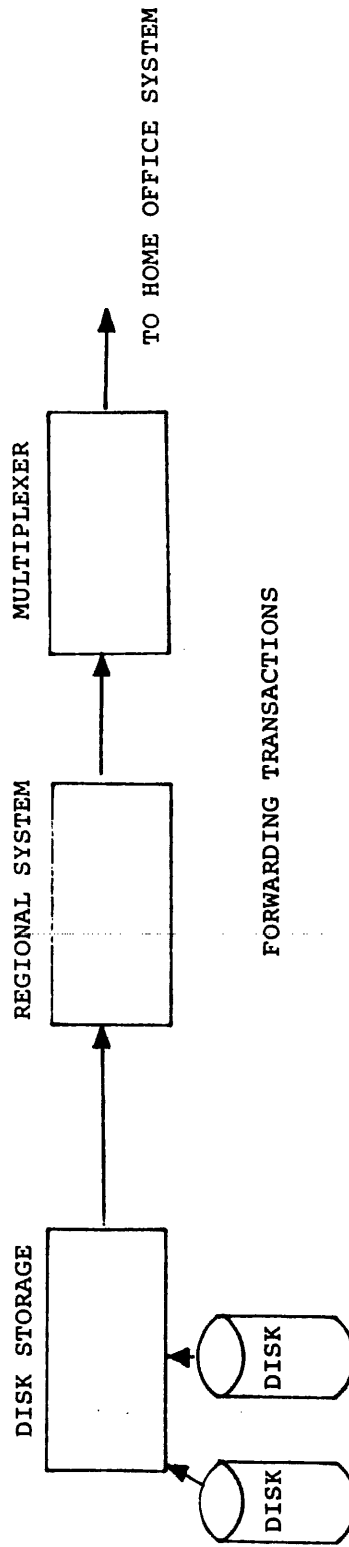
FORWARDING TRANSACTIONS

A-4

A SHORT NOTE ON HIGH LEVEL LANGUAGES

AND MICROPROCESSORS

by Sassan Hazeghi *
and Lichen Wang

STANFORD LINEAR ACCELERATOR CENTER

Abstract

In this note, some of the practical aspects of bridging the gap between high level programming language and computer hardware are discussed. Several possible strategies are considered and the method of half-compiling-half-interpreting is studied. In dealing with address space limitation (or tight memory situation) and slow speed of micro processors running an interpreter, a measurement and analysis technique is suggested. This analysis not only gives a good estimate of the timing and storage requirement before the actual implementation, it also helps to optimize the speed and storage usage of the implementation. The note concludes with some results concerning the implementation of the programming language PASCAL on a family of micro processors.

The question of high level language versus assembly language programming is far from resolved, and it is unlikely that it will ever be settled to the satisfaction of everyone involved. However, it can safely be said: a high level language is an essential programming tool in large scale software projects. For large and medium scale computers, the major applications that were written in assembly language, namely system software such as compilers, interpreters, operating systems and even assemblers, are now, to a very large extent, being written in high level languages.

In the case of microprocessors, the rather primitive nature of the instruction sets of these machines and their inherent address space limitation makes them a less desirable target for compiler writers and compilers. Nevertheless, starting

----

with the first generation of the microprocessors there has been a great deal of interest in implementing existing high level languages and even in defining new languages for these new devices.

The early versions of microprocessor compilers and assemblers were intended to run on available medium or large scale computers which generate machine code for the target microprocessor. These programs, generally referred to as cross compilers and assemblers, (themselves benefactors of the already existing software tools on their host computers) were made available to the microcomputer programmers through time sharing networks or in-house host computers, and thus were not easily accessible to the small users who had no extra support but their microprocessors.

The resulting pressure from the user community prompted the microprocessor manufacturers, as well as independent software establishments, to provide resident software for the microcomputers. Unfortu-

nately, the existing cross compilers cannot usually be used to "bootstrap" themselves into a "resident" version, and the limitations of the microprocessors make the task of writing good resident compilers a rather difficult one. An alternative approach to this problem is the use of "interpreters" which do not try to translate the set of actions specified by the user program into machine code, but instead execute these actions in their own way.

There are two important characteristics of the interpreters which make them particularly suitable for microprocessors. The first is that the high level language representation of a program is generally more compact than its assembly language form. In other words, high level languages result in a denser "encoding" of the program. The famous APL "one-liners" are an extreme example of such compact encodings. The second characteristic is that it is generally easier to mimic the actions specified by a program than to translate that program into a sufficiently different language, in particular a machine language. An observation which supports this idea is that interpreters are usually much smaller than "full" compilers for high level languages.

Due to these and other similar considerations, the first high level language which gained widespread use on microprocessors was an interpretive BASIC. The availability of very compact interpreters, as well as the simplicity of the language, made it the microprocessors' "universal" high level language. This is particularly evident amongst computer hobbyists with limited resources and little concern for speed and (in)efficiency of the programs. These interpreters, in general, maintain a copy of the program in its original textual form, with little or no change in its representation. Although this results in ease of modification of the program and meaningful diagnostic messages, it is still far from optimal in terms of the storage requirements, in addition to being very inefficient in terms of running time.

Implementing an interpreter or a

compiler in assembly language is a long and tedious project. Moreover, the work has to be repeated for each microprocessor, and more often than not, some differences are introduced in the underlying high level language. The standard method of bootstrapping a self-compiling compiler (a compiler written in the language it implements) into a resident compiler appears to be a very attractive approach in dealing with these problems. However, before starting any actual implementation, some careful planning is needed in order to overcome the problem of limited memory space. Therefore, as an experiment in the microprocessor software design, we decided to try implementing a compiler for the programming language PASCAL on a family of microprocessors.

One of the existing PASCAL compilers, the so called P_Compiler [1], is specifically designed to simplify the task of generating code for different target computers. This is accomplished by generating code for a hypothetical ("universal") Stack Machine which is to be mapped into the target computer machine code. The stack machine is designed to simplify the compiler itself [2], [3], by leaving the details of the (machine dependent) register assignment and utilization problem to the final implementor. Another characteristic of the stack machine code (also called "zero address" code) is the relatively short average instruction length due to lack of register specification fields within the instructions. For these reasons, we adopted a modified version of the P_Compiler stack machine as our virtual intermediate machine, with the idea that this machine could be implemented (i.e., interpreters written) on a variety of microprocessors with reasonable efficiency. Table-1 shows this modified instruction set, which basically satisfies the following requirements:

a) The instruction set of the I.M., referred to as the Intermediate Language (I.L.), is quite small. Many studies of the instruction sets of different

computers [4] have shown that the bulk of the programs are composed by only a few different instructions, which also account for a large fraction of the program's running time (in general, the opcode entropy is on the order of 3-4 bits). But instructions are usually an integral number of storage units and a large number of different opcodes tends to increase the average instruction length (Table-2, 8080 frequencies).

b) The I.L. is relatively machine independent and it can be easily implemented on a variety of microcomputers. Even though the I.L., as generated by the P_Compiler, could be used on the target machine in a number of different ways (e.g. direct translation, macro expansion, etc.), in its final encoded form it is intended for an interpretive implementation.

c) The level of complexity of the instructions represents a compromise between the source language and the potential target machines. To this end, the I.L. includes "floating point", "set" (as used in PASCAL) and address computation instructions, but does not provide bit manipulation operations such as "Shift", "Rotate", and "Bit Test".

d) The I.M. code is flexible enough so that it can be relocated in memory or divided into segments or pages.

e) The I.L. includes provisions for optional runtime checking so that the runtime errors can be easily traced to the source statements which caused them.

The runtime environment of the I.M. closely resembles that of PASCAL and other block structured procedural (ALGOL-like) languages. Such an environment is highly useful to the compiler writer, and could also be used by programs written directly in the I.L. thereby simplifying the storage management problem.

A previous experience in mapping the I.L. into IBM 370 code [5], using a moderately optimizing transformation, resulted in an average length of 4 to 5 bytes per translated I.L. instruction. The PASCAL P Compiler itself, a 4000 line PASCAL program, was translated into about 15000 I.L. statements. This turned out to be equivalent of about 73000 bytes of the IBM 370 code, and did not include the data area required while running the compiler. It is clear that a direct translation of the I.L. into machine code, or even a straight-forward mapping of the I.L. into the microprocessor's memory, would far exceed the available storage on most of these machines.

To deal with this problem, we developed a set of programs to trace and analyze the I.M. code which provide static and dynamic information about the I.M. instruction set. By static information, we mean the statistical distribution of various instructions in a given program as it is loaded into memory. Dynamic information, is based on the frequency of execution of various instructions as the program is executed. These distributions are obviously program dependent and the dynamic properties are, in addition, data dependent. We considered the P_Compiler itself to represent a extreme case in terms of size and complexity both as a program and as data (while being compiled). The P_Compiler was therefore used as a benchmark to compare different revisions of the I.L. with regard to the projected size and speed of the encoded program.

The standard way to cut down on the "opcode space" of the instructions is to use some form of the Huffman-encoding of the opcodes, in which the most frequent opcode has the shortest representation. Howerer, practical considerations, such as operand and memory alignment requiments, complicate the problem. For example, the "LOD" and "LDC" (loading variable values or constants on top of the I.M. stack) are the most frequently used instructions (Table-3). The typical "LDC" instruction should be able to specify a full integer constant (a 2-byte quan-

tity) and the "LOD" operation should include both a "level number" (typically in the range 0 to 7) and a sufficiently large "offset" field. Thus even with Huffman-encoding, LOD and LDC would still require about 2-3 bytes per instruction. In the case of the compiler, these instructions alone would use up 10-15 K bytes of memory. A closer look, however, shows that the operand distributions are even more skewed than the opcode distribution (Table-4). This suggests that, if needed, the operand fields could also be Huffman-encoded. Since the above two instructions are also heavily represented in the dynamic instruction counts, we decided on a fairly simple encoding, namely, dividing these into different groups, depending on the size of the operand.

For the "LDC" instruction, we noticed that the great majority of the (integer) constants used in programs are positive, and, in the case of the P_Compiler, about 80% of these constants are in the range 0 to 31. Thus if we use a 3-bit opcode and a 5-bit operand field, a large fraction of LDC instructions could be represented by a new one byte instruction. This was in line with an earlier decision to align instructions on byte boundaries but to allow operands to cross over byte boundaries, whenever this would not create a major runtime penalty in extracting the operand fields. Likewise, for the LOD instructions, it is known that most of the references are to the absolute Global variables (those defined in the main program) and the most Local ones (those defined in the very procedure in which they are being referenced). The Local variables are, most often, simple variables (as opposed to "structured" variables in PASCAL terminology) and few in number, so their offset value in the address field is small. For the Global variables, however, we noticed that a few "hyperactive" variables are overly represented in the reference pattern. (One can observe this phenomenon by looking at a cross reference listing of any program, whether written in assembly or high level language.) Reord-

ering the Global variables (performed either by the programmer or the compiler) so that simple variables precede larger (structured) variables and are ordered by their reference frequencies, results in a short form (1-byte) "LOD" instruction which can replace the majority of the original "LOD" instructions. Of course, one has to provide the long form of these instructions, so that none of these decisions impose any restrictions on the language. The important point, however, is that these long forms are not employed frequently enough to make a significant contribution to the program size or execution time.

The relatively high frequency of the procedure call instruction "CUP" reflects the current trend in programming style, which emphasizes modular and structured program organization. (The P_Compiler consists of about 100 procedures). The efficiency of the implementation of "Call" and "Return" mechanisms thus becomes an important factor in the efficiency of the overall program. For this reason, it was decided to include the somewhat redundant "MST" instruction which flags the beginning of evaluation of parameters prior to a Call. This instruction helps speed up the "call" sequence in our actual implementation, but it can be removed from the instruction set if storage limitation becomes an issue. The call instruction "CUP", however, presents a more interesting problem. The straightforward implementation of this instruction would include the entry point address of the Called Procedure in the operand field of the instruction. By observing that there are usually many more Calls than Procedures, we concluded that procedure names (addresses) were good candidates for more compact encoding. Using a procedure number (name) with a much smaller range than a procedure address, we can use a 2-byte "CUP" instruction, instead of the 3-byte full address version. Further examination of the call instruction reveals that not all the procedures are alike, since only a few procedures are the target of the bulk of the call instructions (Table-5). Having set aside some

4-bit opcodes in the beginning, we chose to use smaller numbers (shorter representation) for the most frequently called procedures, and larger ones for the less popular ones. This encoding allows up to 16 procedures to be called by 1-byte call instructions (about 85% of the procedure calls in the compiler), while the rest of them are called by the 2-byte call instructions. Note that having a "procedure address table" (PAT) to implement such calls is also essential to the code relocatability at the procedure level. Moreover, we have envisioned a simple overlaying scheme which uses the same PAT to "page" procedures into the main storage from a secondary storage device such as floppy disks. In such a case, the PAT entry will contain the main memory address of the procedure, or its seconday memory address (disk address), as well as appropriate flags. The I.M. code is read-only and paging out is not necessary; one only has to invalidate the PAT entries for the segments overlayed by a paged-in procedure.

The next candidates for the 4-bit opcodes, based on the frequency table, were conditional and unconditional branch instructions. Over half of the conditional branches are forward branches to targets within 16 I.M. instructions from the source of the branch. Although this represents a large number of instructions, we realized that the "average length" of the final encoded instructions would be closer to 2 bytes. Consequently cutting down the ratio of branch instructions with small branch distances. In addition, exact determination of all such short "relative branch" instructions requires a non trivial, multi-pass algorithm, which was considered to outweigh the potential benefit. Instead, we settled on a 12-bit target address which is relative to the beginning of the procedure. This puts a limit of 4K bytes on the size of each procedure which, given the projected code density, was felt to be quite adequate. (The largest procedure in the P_Compiler is encoded in well under 2K bytes). Since branching to a point outside the

procedure containing the branch instruction is not allowed in PASCAL, the "procedure-relative" nature of our branch instructions does not pose any problem. Furthermore, procedures can be easily "relocated" in the main memory with no restrictions and in a completely transparent mode. This feature is also needed to simplify the overlay scheme discussed previously.

Without going into great detail in describing the analysis and encoding of the other instructions, we note that the first 10 instructions of the I.M. instruction code account for over 75% of the static instruction count, and somewhat smaller but still significant fraction of the dynamic count. Consquently, we can afford to be less concerned with the compactness of representation and pay more attention to the efficiency of the implementation of some of the statistically infrequent but, dynamically or otherwise, important instructions. One of the surprises in the static count listing is that the integer add "ADI" instruction is less frequent than the number of procedures. In other words the average procedure in the P_Compiler contains less then one Addition operation, and the number of Subtraction, Division or Multiplication operations are well below the number of Additions. In this sense, the lack of arithmetic instructions for operations like multiplication and division, at least as far as the compiler writer is concerned, is not a major loss. However, the software implementation of these instructions should favor the running time in the space/speed compromise to make them more attractive to numerical applications.

After completing the design of the instruction set, we were able to encode the P_Compiler and get an accurate measure of the program's total memory requirement as well as the contribution of individual instructions to the total sum. (Table-6 summarizes these results.) These figures correspond to a version of the compiler which was adopted for the IBM 370 implementation and includes features (such as alignment of variables on appropriate byte boun-

daries and collection of other information), which are irrelevant to the microprocessor implementation. By eliminating these features, we feel that the P_Compiler will fit in about 25-26 K bytes witnout resorting to the other storage saving options mentioned earlier.

Until tnis point in the design, no actual implementation of the interpreter was attempted. Only after we were reasonaoly satisfied with the encoded size of our bencnmark program, did we decide to write an interpreter for a particular microprocessor. Once tne representation of the I.M.'s Stack and the general plan for implementing the Instruction Fetch and Execution cycles in the interpreter were defined, the coding of the individual instructions proved to be a quite simple and almost mechanical task. The resulting program, excluding its taoles and the section dealing with floating point instructions (not yet implemented), is about 1.5K bytes long and execept for a small I/O interface is entirely self contained.

In order to ootain an estimate of tne performance of tne interpreter we decided to use a Quick-Sort program (due to C.A.R. Hoare and J. Sedgewick) which was extensively used in the evaluation of the Pascal 370 compiler. The PASCAL source of tnis program (including tne I/O and a Random Number generating routine) is about 130 lines long and it compiles into 410 I.L. instructions. Tnis in turn can oe translated into 1240 bytes of tne IBM/370 code or encoded in 580 oytes for tne interpretive execution by tne microprocessor. The program's running time, when sorting N "random" integers, is proportional to N*LOG(N) and takes about 2-seconds on the 370/168 for N = 10000. The 2MHz 8080/Z80 microprocessor Quick-Sort, using tne same program, sorts N = 1000 randomly generated numbers in less than 20 seconds under tne current version of the interpreter. This rudimentary comparison shows that the interpretive microprocessor version of tne sort routine is about 100 times slower tnan the "compiled" 370/168 code.

A more meaningfull evaluation would be the comparison of the timing results of the I.L. interpreter with the alternatives available to tne microprocessor programmer, namely the Assembly Language and the BASIC codings of the Quick-Sort routine. Although the assemoly language implementation of the sort routine is not completed yet, the (integer) BASIC coding of tne program seems to oe about 18 times slower than the PASCAL version. (The Tiny BASIC [6] Quik-Sort program is about 750 bytes long and sorts 100 numbers in 24 seconds. Tnis time increases to about 360 seconds for 1000 random integers.) Even though the performance measurements of the interpreter is at a very early stage at the time of this writing, by an extrapolation of the available results, it seems that the interpreter is efficient enough to be useful in compiling large programs such as the P_compiler. This will make the micro prossesor based compiler a stand alone system without the need for external software support from larger computers.

In conclusion, the "interactive" analysis technique presented here proved to oe a very helpful and effective tool in resolving some important design issues and answering critical questions before any implementation decision had to be made. All too often, these early decision are made prematurely and without enough data. The ability to measure tne cost and effect of such decisions, before being committed to them, is essential in designing software systems which must meet difficult requirements. Although the major concern in our experiment was the limited address space of the microprocessors, one could use a similar approach in estimating the running time (or other dynamic properties) of programs, thereby avoiding potential pitfalls. With the experience gained by implementing the I.L. on the 8080 microprocessor family, we feel that the implementation of such interpreters on other existing or future microprocessors can be an effective way of overcoming the deficiencies and limitations of microprocessors as well as providing a reasonable vehicle

for transporting software across different machines.

AKNOWLEDGEMENT

We wish to thank Len Shustek for his help in reviewing this note and providing the table of 8080 opcode frequencies [5].

REFERENCES:

[1] K. NORI, U. AMMAN, K. JENSEN, H. NAGELI. 'The PASCAL P COMPILER, Implementaion Notes', Berichtes des Instituts fur Informatik, E.T.H. Zurich, Dec. 1974.

[2] D. BULMAN, 'STACK COMPUTERS: An Introduction', I.E.E.E. Computer, May 1977.

[3] N. WIRTH, 'Stcak vs. Multiregister Computers', ACM SIGPLAN Notices Notices, March 1968.

[4] L. SHUSTEK and B. PEUTO, 'Current Issues in the Architecture of Microprocessors', I.E.E.E. Computer, Dec. 1976.

[5] S. HAZEGHI, 'Bootstrap and Adaptation of a PASCAL Compiler on the IBM/370 Computer', Computation Group Technical Memo (in preparation), Stanfor Linear Accelerator Center.

[6] L. WANG, 'Palo Alto Tiny BASIC', DDJ, VOL 1, NO 5, May 1976.

LAB	OPERATION	OPERANDS	EFFECT(S)	
	ABI (ABR)		nt←ABS(t)	{absolute value}
	ADI (ADR)		nt←2t + t	
	AND		nt←2t AND t	
	CHK	P,Q	~(P<=t<=Q)→ERROR	{bounds checking}
	CSP	Q	PUSH(pc); pc←Q	{call stnd. proc.}
	CUP	T,P,Q	PUSH(pc); pc←Q	{call user proc.}
	DEC	Q	nt←t-Q	
LAB	DEF	Q	(psuedo op)	{LAB set to the Integer Q}
	DIF		nt←2t ⊖ t	{Set Difference}
	DVI (DVR)		nt←2t DIV (/) t	
LAB	ENT	T,P,Q		{entering proc. LAB, type T, level P, local stack frame size Q}
	EQU	T(,Q)	nt←2t = t	
	FJP	Q	~t→(pc← Q)	{branch on FALSE}
	FLO		2t←FLOAT(2t)	{FIX to FLOAT conversion}
	FLT		t←FLOAT(t)	" " "
	GEQ	T(,Q)	nt←2t >= t	
	GRT	T(,q)	nt←2t > t	
	INC	Q	nt←t + Q	
	IND	Q	nt←M[t+Q]	{indirect load}
	INN		nt←2t IN t	{set membership test}
	INT		nt←2t ♂ t	{set intersection}
	IOR		nt←2t	t	
	IXA	Q	nt←2t + Q*t	{compute 'base/index' address}
LAB	LAB		(pseudo op)	{label definition}
	LSA	Q	nt←.Q	{load address of string Q}
	LDA	P,Q	t←.M[<P,Q>]	{load <base/level> address}
	LDC	T,Q	t←Q	{load constant}
	LEQ	T(,T)	nt←2t <= t	
	LES	T(,Q)	nt←2t < t	
	LOC	Q	(pseudo op)	{LOCation counter}
	LOD	T,P,Q	nt←<P,Q>	{load from <level:P, offset Q>}
	MOD		nt←2t MOD t	
	MOV	Q	M[2t:Q]←M[t:Q]	{move Q 'locations'}
	MPI (MPR)		nt←2t * t	
	MST			{begin proc. parameter list}
	NEQ	T(,Q)	nt←2t <> t	
	NEW	Q	M[t]←hp;np←np-Q	{NEW stnd. proc}
	NGI (NGR)		nt←-t	
	NOT		nt← ~t	
	RET	T	POP(pc)	{return to calling routine}
	RST		np←t	{release dynamic storage}
	SAV		M[t]←np	{mark dynamic storage}
	SBI (SBR)		nt←2t - t	
	SGS		nt← [t]	{generate a single member set}
	STO		M[2t]←t	{indirect store}
	STP		EXIT(t)	{terminate execution}
	STR	T,P,Q	M[<P,Q>]←t	{store into level: P, offset Q}
	TRC		nt←TRUNC(t)	{TRUNC operator}
	UJP	Q	pc←Q	{unconditional branch}
	UNI		t←2t ⊕ t	{set union}
	XJP	Q	pc←Q + t	{indexed branch}

NOTATIONS .:

← .:.= assignment,   → .:.= conditional,
t .:.= Top stack element,   2t .:.= 2nd Top stack element ,
nt .:.= New top stack element (implies POPing the old one),
pc .:.= program counter,   hp .:.= dynamic storage pointer,
P,Q .:.= instruction operands,   L .:.= current (static) level
T .:.= type of the operand/procedure,
M ::= data storage array,
M[i:j] .:.= locations M[i] through M[i+j-1],
M[i] ::. = M[i:1],   .a ::= address of the entity 'a',
<i,j> .:.= base_level address (i.e. offset j of level i).

TABLE 1   I.M. INSTRUCTION SET

OPCODE		% INSTR.	% CUMM.
LOD	R,M	12.92	12.92
LODI	R	11.65	24.57
LODI	RR	9.63	34.21
LOD	M,R	7.66	41.87
LOD	R,R	7.06	48.93
INC	R	6.39	55.88
ADD	HL	5.71	61.59
INC	RR	5.55	67.14

(PL/M PROGRAM)

OPCODE		% INSTR.	% CUMM.
CALL		13.00	13.00
LOD	R,R	9.06	22.06
JMP	CC	7.96	30.02
POP	RR	6.96	36.98
PUSH	RR	6.85	43.84
INC	RR	4.46	48.30
RET		4.36	52.66
LODI	RR	3.96	56.62
LOD	R,M	3.70	60.32
JMP	U	3.28	63.61
LODI	R	3.17	66.78
CMPI	N	2.91	69.69
XCH		2.89	72.58
LD	HL	2.41	74.99

(ASSEMBLY LANG. PROGRAM)

STATIC 8080 OPCODE FREQUENCIES

TABLE 2

OPCODE	% INST.	% CUMM.	RAW COUNTS
LOD	18.8	18.8	2736
LDC	15.5	34.3	2254
MST	7.1	41.5	1037
CUP	7.1	48.6	1037
FJP	6.0	54.7	877
STR	5.7	60.5	837
UJP	5.2	65.7	756
LDA	4.4	70.1	647
STO	3.0	73.2	445
IND	2.7	76.0	402
EQU	2.6	78.6	338
CSP	2.4	81.1	361
IXA	2.3	83.5	346
INC	2.1	85.6	308
DEC	1.7	87.4	252
NEQ	1.6	89.0	238
MOV	1.6	90.6	236
LSA	1.6	92.3	234
LCI	1.0	93.3	156
ORD	.9	94.3	138
UNI	.8	95.1	123
NOT	.6	95.8	100
RET	.6	96.5	96
ENT	.6	97.1	96
ADI	.5	97.7	81
INN	.5	98.2	80
AND	.3	98.6	49
LEQ	.3	98.9	44
GRT	.1	99.1	25
SBI	.1	99.2	24
XJP	.1	99.4	20
LES	.1	99.5	19
IOR	.0	99.6	13
GEQ	.0	99.7	13
MPI	.0	99.7	9
CHR	.0	99.8	9
DVI	.0	99.8	7
NGI	.0	99.9	4
SGS	.0	99.9	3
MOD	.0	99.9	3
DIF	.0	99.9	2
STP	.0	99.9	1
ODD	.0	99.9	1
ABI	.0	100.0	1

TOTAL COUNT = 14508,   ENTROPY = 4.10

STATIC I.L. OPCODE FREQUENCIES
FOR THE PASCAL_P COMPILER.

TABLE 3

OPERAND RRANGE	COUNT	CUMMU.	% COUNT
0	207	207	9.1
1	227	434	19.2
2	193	627	27.8
4	234	861	38.1
8	305	1166	51.7
16	260	1426	63.2
32	309	1735	76.9
64	74	1809	80.2
128	73	1882	83.4
256	32	1914	84.9
512	3	1917	85.0
1024	0	1917	85.0
2048	0	1917	85.0
4096	0	1917	85.0
8192	0	1917	85.0
16384	1	1918	85.0

NUMBER OF DISTINCT OPERANDS= 148
COUNT= 2254 OR 15.5% OF THE TOTAL.

DISTRIBUTION OF "LDC" OPERANDS

VALUE RRANGE	COUNT	CUMMU.	% COUNT
0	338	338	12.3
1	0	338	12.3
2	181	519	18.9
4	233	752	27.4
8	757	1509	55.1
16	416	1925	70.3
32	230	2155	78.7
64	283	2438	89.1
128	17	2455	89.7
256	224	2679	97.9
512	39	2718	99.3
1024	0	2718	99.3
2048	18	2736	100.0

NUMBER OF DISTINCT OPERANDS= 84
COUNT= 2736 OR 18.8% OF THE TOTAL.

DISTRIBUTION OF "LOD" OPERANDS

BRANCH DISTANCE	INTERVAL COUNT	CUMM. COUNT
-1024	0	52
-512	0	52
-256	0	52
-128	8	44
-64	9	35
-32	11	24
-16	13	11
-8	10	1
-4	1	0
-2	0	0
-1	0	0
0	0	0
1	0	0
2	32	0
4	343	32
8	179	375
16	114	554
32	73	668
64	54	741
128	21	795
256	8	816
512	1	824
1024	0	825

COUNT= 877 OR 6.0% OF THE TOTAL.
FORWARD/BACKWARD BRANCH RATIO = 15.0

DISTRIBUTION OF BRANCH DISTANCES
FOR THE "FJR" INSTRUCTION.

INSTRUCTION OPERAND DISTRIBUTIONS
(FOR THE PASCAL_P COMPILER)

TABLE 4

I.L. STATIC INST. COUNT		I.L. DYNAMIC INST. COUNT		CALLED PROC. DISTRIBUTION		"LDC" OPERAND DISTRIBUTION	
LØD	18.8%	LØD	21.2%	ERRØR	28.8%	LDC 0	10.7%
LDC	15.5%	LDC	11.9%			LDC 1	11.8%
MST	7.1%	FJP	9.2%	INSYMBØ	16.9%	LDC 2	10.1%
CUP	7.1%	STR	7.7%			LDC 4	12.2%
FJP	6.0%	LDA	5.5%	GENO	5.1%	LDC 8	15.9%
STR	5.7%	IND	4.2%	SKIP	4.9%		
UJP	5.2%	IXA	3.9%	NEXTCH	4.6%	LDC 16	13.6%
LDA	4.4%	DEC	3.7%	CØMPTYP	4.3%		
STØ	3.0%	ADI	3.1%	GEN2	3.8%	LDC 32	16.1%
IND	2.7%	NEQ	3.0%	GEN1	3.4%		
EQU	2.6%	EQU	2.8%	LØAD	2.8%	LDC 64	3.9%
CSP	2.4%	UJP	2.4%	EXPRESS	2.1%	LDC 128	3.8%
IXA	2.3%			ENTERID	2.1%	LDC 256	1.7%
INC	2.1%			SEARCHI	1.9%		
DEC	1.7%			LØADADD	1.7%		
				FLOV	1.7%		
				GENLABE	1.7%		
ØTHER	13.4%	ØTHER	21.4%	ØTHER	14.5%		

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

OPCODE/OPERAND DISTRIBUTIONS OF THE BENCHMARK PROGRAM

TABLE 5

OPCODE	OPND FIELD			ABS. COUNT	RELA. COUNT	CUMM. COUNT	INSTR. LENGTH	TOTAL LENGTH	RELA. LENGTH
000XXXXX		0	LDC1	2221	14.8	14.8	1	2221	7.7
110XXXX1		193	LODL	1517	10.1	24.9	1	1517	5.2
01000010		66	MST	1037	6.9	31.8	1	1037	3.6
1000XXXX	D8	128	FJP	877	5.8	37.7	2	1754	6.1
0110XXXX		96	CUP1	836	5.5	43.3	1	836	2.9
1001XXXX	D8	144	UJP	756	5.0	48.3	2	1512	5.2
00110000	D8	48	STRLG	677	4.5	52.8	2	1354	4.7
00100100	D16	36	LOD	627	4.1	57.0	3	1881	6.5
00100000	D16	32	LDC	613	4.0	61.1	3	1839	6.4
01000100		68	STO	443	2.9	64.1	1	443	1.5
01001001		73	ADI	389	2.5	66.7	1	389	1.3
110XXXX0		192	LODG	388	2.5	69.3	1	388	1.3
00111001		57	EQU	383	2.5	71.8	1	383	1.3
00110100	D8	52	LDALG	332	2.2	74.0	2	664	2.3
00110101	D16	53	LDA	315	2.1	76.2	3	945	3.2
0111XXXX		112	CSP1	297	1.9	78.1	1	297	1.0
01001010		74	SBI	296	1.9	80.1	1	296	1.0
1010XXXX		160	IND1	267	1.7	81.9	1	267	.9
1011XXXX		176	IXA1	265	1.7	83.7	1	265	.9
01000001	D8	65	MOV	236	1.5	85.2	2	472	1.6
00111010		58	NEQ	236	1.5	86.8	1	236	.8
00110110	DX	54	LSA	234	1.5	88.4	18	4212	14.6
00100101	D16	37	LOD8	204	1.3	89.7	3	612	2.1
00101000	D8	40	CUP	201	1.3	91.1	2	402	1.4
00100001	D64	33	LDC8	156	1.0	92.1	9	1404	4.8
01000111		71	ORD	138	.9	93.0	0	0	.0
00101010	D8	42	IND	134	.8	93.9	2	268	.9
01010111		87	UNI	123	.8	94.8	1	123	.4
01011101		93	NOT	100	.6	95.4	1	100	.3
			ENT	96	.6	96.1	11	1056	3.6
01000011		67	RET	96	.6	96.7	1	96	.3
00101110	D16	46	IXA	81	.5	97.3	3	243	.8
01010110		86	INN	80	.5	97.8	1	80	.2
01011010	D8	90	NEW	57	.3	98.2	2	114	.3
01011011		91	AND	49	.3	98.5	1	49	.1
00111011		59	LEQ	44	.2	98.8	1	44	.1
00111110		62	GRT	25	.1	99.0	1	25	.0
00110111	D40	55	XJP	20	.1	99.1	6	120	.4
00111100		60	LES	16	.1	99.2	1	16	.0
00110010	D16	50	STRS	15	.1	99.3	3	45	.1
00110001	D16	49	STR	15	.1	99.4	3	45	.1
01011100		92	IOR	13	.0	99.5	1	13	.0
00111101		61	GEQ	13	.0	99.6	1	13	.0
01001101		77	MPI	9	.0	99.6	1	9	.0
00111000	D16	56	CMPM	9	.0	99.7	3	27	.0
01001011		75	DVI	7	.0	99.8	1	7	.0
00101001	D8	41	CSP	5	.0	99.8	2	10	.0
01001110		78	NGI	4	.0	99.9	1	4	.0
01010101		85	SGS	3	.0	99.9	1	3	.0
01001100		76	MOD	3	.0	99.9	1	3	.0
01011001		89	DIF	2	.0	99.9	1	2	.0
01000101		69	STO8	2	.0	99.9	1	2	.0
01010100		84	ABI	1	.0	99.9	1	1	.0
01001111		79	RST	1	.0	99.9	1	1	.0
00111111	D8	63	CMPS	1	.0	99.9	2	2	.0
00101011	D8	43	INDS	1	.0	100.0	2	2	.0

TOTAL COUNT = 14975, BYTES = 28710.

ENCODED INTERMEDIATE LANGUAGE INSTRUCTIONS
FOR THE BENCHMARK PROGRAM (P_COMPILER).

TABLE-6

COMPILER CONSTRUCTION·FOR SMALL COMPUTERS

R. Broucke

Dept. of Aerospace Engineering and Engineering
Mechanics, University of Texas at Austin
Austin, Texas 78712

Abstract

We describe a simple parsing algorithm that can be used in a compiler to translate complex statements in machine instructions. It could be used on any small computer. To describe the Parser in detail we also include a SNOBOL-implementation of it (one page of coding only) and three examples of translated statements.

Introduction

In the present article we describe a simple parser that could be easily implemented on a small computer or even on a microcomputer. The Parser is the heart of any compiler. It translates the complex statements of a higher-level language and decomposes them in small elements, eventually in machine instructions.

In order to be able to describe the parser we assume that the higher-level language is Fortran and that the machine language has only six instructions: Add, Subtract, Multiply, Divide, Exponents and Store. In order to test the algorithm we programmed it in SNOBOL4 which is a well known language for symbol manipulation. We give the complete listing of the program (only one page) at the end of the article, together with a one-page output of the program: the translation of three fairly complex Fortran statements in elementary operations. The elementary operations are preceded by the word CALL. The reader who is not too familiar with SNOBOL4 will be able to understand the mechanism of the Parsing algorithm by carefully studying the three examples.

We hope that this short presentation of a "Simple Parser" will help some of the experienced programmers in building simple compilers for their personal computer.

Description of Parser

We insist only on the principal ideas and precedence rules that make up the body of a parsing algorithm. In fact we are restricting ourselves to the following simple arithmetic expressions.

1. There are only five binary operations: exponents, multiply, divide, add and subtract, identified by the symbols **, *, /, + and - .
2. There may be an arbitrary number of parentheses, but they have to form matching pairs.
3. There are no constants in the expressions, but only variables, (of any length, starting with a letter).
4. There are no subscripts or dimensioned variables.
5. No considerations of type are made (such as integer or real).

The parser program has 60 SNOBOL statements. The first 16 lines are Pattern definitions and the last 16 lines are the main program. The other 28 lines are the four subroutines of the program: EXP, COMPIL, POPT, DROPT. The subroutine EXP is really the heart of the compiler; it transforms any expression without parentheses into a single variable.

The basic principle of the Parser consists in searching for elementary expressions called Binaries. A Binary is a pair of variables separated by an operation such as **, *, /, + or -. The subroutine EXP compiles these binaries by replacing them with a tempory variable inside the statement and simultaneously generating a CALL statement as output.

During the compilation process of any statement, we use the following precedence rules:

All expressions inside parentheses are compiled first; pairs of parentheses are thus removed gradually until a "level zero" expression with no parentheses remains. A call to the subroutine EXP is made for each pair of parentheses. The final expression is then compiled by a last call to the subroutine EXP. Inside the subroutine EXP, the following precedence rules are respected.
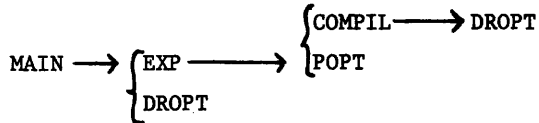
The exponential binaries (of the form A**B) are compiled first, from left to right as they occur in the statement.

The multiplicative binaries (of the form A*B or A/B) are treated next, from left to right.

Finally the additive binaries (such as A+B and A-B) are compiled, again from left to right as they occur in the expression.

A call is made to the subroutine COMPIL for each binary that is found in the expression. Also, a temporary variable is generated (a letter T followed by a 5-digit integer) for the storage location of the result of the binary operation. For instance the result of A*B may be stored in T00012. The label T00012 is constructed by the SNOBOL program. At the end of the compilation of A*B = T00012, the program checks if A, B happen to be temporary variables. If they are, they will

be stored in a stack (last-in, first-out
principle) with available temporary locations
(TAVAIL). This is done by the subroutine DROPT,
with the purpose of optimizing storage locations
in the object program. The subroutine POPT that
finds temporary locations whenever they are needed
will first search in the table of avialable
temporaries. If this table is empty, a new
temporary variable will be constructed. The reader
can understand this feature by examining the three
examples of compiled statements given at the end
of the text. It can easily be seen that the flow-
chart of the program can be represented as follows:

$$\text{MAIN} \longrightarrow \begin{cases} \text{EXP} \longrightarrow \begin{cases} \text{COMPIL} \longrightarrow \text{DROPT} \\ \text{POPT} \end{cases} \\ \text{DROPT} \end{cases}$$

```
            ALPHA = 'ABCDEFGHIJLKMNOPQRSTUVWXYZ'
            DIGIT = '0123456789'
            ALNUM = ALPHA DIGIT
            B     = ' '
            BS = SPAN(B) ! ''
            LP =    BS '('
            RP =    BS ')'
            VE       =      (ANY(ALPHA) (SPAN(ALNUM) ! ''  )) . VAR
            EXX = BS '**'
            MULDIV = BS ('*' ! '/') . MD
            ADDSUB = BS ('+' ! '-') . AS
            BINARYE = BS VE . LVAR EXX      BS VE . RVAR
            BINARYM = BS VE . LVAR MULDIV BS VE . RVAR
            BINARYA = BS VE . LVAR ADDSUB BS VE . RVAR
            PEXP =  LP BREAK('()') . INSIDE RP
            TPAT = 'T' ANY(DIGIT) ANY(DIGIT) ANY(DIGIT) ANY(DIGIT) ANY(DIGIT)
                 DEFINE('COMPIL(OPERTN,A1,A2,RE)')                   :(END.CMP)
COMPIL      OUTPUT = DUPL(' ',30) 'CALL ' OPERTN  '(' A1 ',' A2 ',' RE ')'
            DROPT(A1)
            DROPT(A2)                                               :(RETURN)
END.CMP
            DEFINE('POPT()')                                        :(POPT.END)
POPT TAVAIL (',' BREAK(',') . POP ) = ''                           :S(POP2)
            TEMPNR  = TEMPNR + 1
            POP = 'T' DUPL('0', 5 - SIZE(TEMPNR)) TEMPNR
POP2 POPT  =  POP                                                  :(RETURN)
POPT.END
            DEFINE('DROPT(T1)')                                     :(D.END)
DROPT T1  TPAT                                                     :F(RETURN)
            TAVAIL = ',' T1 TAVAIL                                 :(RETURN)
D.END
            DEFINE('EXP(EXPR)')                                     :(END.EXP)
EXP        EXPR BINARYE  = POPT()                                  :F(L5)
            COMPIL('POW'  ,LVAR,RVAR,POP)                           :(EXP)
L5         EXPR BINARYM  = POPT()                                  :F(L6)
            OP    = 'MUL'
            OP    = IDENT(MD,'/')  'DIV'
            COMPIL(OP,LVAR,RVAR,POP)                                :(L5)
L6         EXPR BINARYA  = POPT()                                  :F(L70)
            OP    = 'ADD'
            OP    = IDENT(AS,'-')  'SUB'
            COMPIL(OP,LVAR,RVAR,POP)                                :(L6)
L70        EXP = EXPR                                               :(RETURN)
END.EXP
            TEMPNR =0
            TAVAIL =  ','
L1         EXPR = TRIM(INPUT)                                       :F(END)
            OUTPUT = EXPR

            EXPR VE . LEFTV = ''                                    :F(E1)
            EXPR ( BS '=') = ''                                     :F(E2)
L8         EXPR PEXP = EXP(INSIDE)                                  :S(L8)
            EXPR  = EXP(EXPR)
            EXPR BS VE = ''                                         :F(L81)
            EXPR = EQ(SIZE(EXPR),0)  ''                             :F(L81)
            OUTPUT = DUPL(' ',30) 'CALL STORE(' VAR ',' LEFTV ')'
            DROPT(VAR)                                              :(L1)
L81        OUTPUT = 'INCORRECT EXPRESSION ON RIGHT SIDE'            :(L1)
E1         OUTPUT = 'ERROR ON LEFT SIDE OF EQUAL SIGN'             :(L1)
E2         OUTPUT = 'INCORRECT EQUAL SIGN '                         :(L1)
END
```

```
Y=WE*R+KL-T**R-X+X/R**TR*RE+A*Z/X**Z-A+Z1**U1/U*T+AS/Z-QE/ZX-Q123K
                              CALL POW(T,R,T00001)
                              CALL POW(R,TR,T00003)
                              CALL POW(X,Z,T00006)
                              CALL POW(Z1,U1,T00005)
                              CALL MUL(WE,R,T00004)
                              CALL DIV(X,T00003,T00002)
                              CALL MUL(T00002,RE,T00003)
                              CALL MUL(A,Z,T00002)
                              CALL DIV(T00002,T00006,T00007)
                              CALL DIV(T00005,U,T00006)
                              CALL MUL(T00006,T,T00005)
                              CALL DIV(AS,Z,T00006)
                              CALL DIV(QE,ZX,T00002)
                              CALL ADD(T00004,KL,T00008)
                              CALL SUB(T00008,T00001,T00004)
                              CALL SUB(T00004,X,T00001)
                              CALL ADD(T00001,T00003,T00004)
                              CALL ADD(T00004,T00007,T00003)
                              CALL SUB(T00003,A,T00007)
                              CALL ADD(T00007,T00005,T00003)
                              CALL ADD(T00003,T00006,T00005)
                              CALL SUB(T00005,T00002,T00006)
                              CALL SUB(T00006,Q123K,T00002)
                              CALL STORE(T00002,Y)

Z=G*A+B*((C-D))**A-F*(X)+((A)+X-(((( A)))) * (((A9-B/D))+D))+A**BD
                              CALL SUB(C,D,T00001)
                              CALL DIV(B,D,T00002)
                              CALL SUB(A9,T00002,T00003)
                              CALL ADD(T00003,D,T00002)
                              CALL MUL(A,T00002,T00003)
                              CALL ADD(A,X,T00002)
                              CALL SUB(T00002,T00003,T00004)
                              CALL POW(T00001,A,T00003)
                              CALL POW(A,BD,T00001)
                              CALL MUL(G,A,T00002)
                              CALL MUL(B,T00003,T00005)
                              CALL MUL(F,X,T00003)
                              CALL ADD(T00002,T00005,T00006)
                              CALL SUB(T00006,T00003,T00005)
                              CALL ADD(T00005,T00004,T00003)
                              CALL ADD(T00003,T00001,T00004)
                              CALL STORE(T00004,Z)

Z = ( A+B*C)/(F*G-(D+E)/(H+K))
                              CALL MUL(B,C,T00004)
                              CALL ADD(A,T00004,T00001)
                              CALL ADD(D,E,T00004)
                              CALL ADD(H,K,T00003)
                              CALL MUL(F,G,T00005)
                              CALL DIV(T00004,T00003,T00006)
                              CALL SUB(T00005,T00006,T00003)
                              CALL DIV(T00001,T00003,T00006)
                              CALL STORE(T00006,Z)
```

TABLE DRIVEN SOFTWARE

AN EXAMPLE

Val Skalabrin,
Second Source

ABSTRACT

A mini data base management system (MDBMS) is implementable
in any computer language which has or can be given, the
capability to perform byte manipulation within a data record.
This capability is definable as the ability to move a
contiguous segment of bytes between the data record and a
work area. With it, a single set of programs can be described
which will allow the user to create, update, and report on
data from virtually any data file. The system will also
allow a data base to be altered in format without requiring
any program modification. The report option of the system
is a version of the usual general-purpose-select-whatever-
you-want program found in most larger installations.

DEFINITION

Mini Data Base Management and MIS is a set of
computer programs, user manuals, user training and
support which allows the manager's staff to
create, maintain and add to their own data file.
MDBMS gives the manager the capability of generat-
ing reports on selected members of his data base.
MDBMS also allows the the manager to change the
format of his data base, dropping or adding types
of information kept in the files, without requiring
additional DP center programming.

WHY MDBMS?

Firms need to maintain on-line computer files to
capture daily activity, and to produce on-call
reports, but, management also reserves the right
to expand and change the contents of the records
kept in that data file - "The county's new regula-
tion means that we'll have to keep an Ethnic
Origin code on all of our employees. Can you DP
boys stick that in all your programs by Monday?"
Plus there are generally several middle management
people asking if you can't keep their thermofro-
cator inventory on the computer.

The computer manufacturer would like to help his
clients by furnishing a file management system for

thermofrocator inventory, but he is geared towards mass production and cannot afford to tailor his software to each installations needs.

The requirements in most cases are very similar in system design:

1. create a file
2. maintain the file
3. update, add, delete records
4. report generation
5. be ready for a change in record format.

The differences in most cases are also very similar:

1. number of records in a file
2. number of items in a record
3. names of the items
4. lengths of the items
5. length of the record
6. data item types (alpha or numeric)
7. report formats and headings.

The Mini Data Base Management System and MIS Report generator discussed in this paper attempt to provide one answer to the requirements listed above.

MDBMS FUNCTIONAL OUTLINE

File Creation

The Driver File - The design philosophy behind MDBMS rests on describing the size and make-up of a data record in a fashion which will allow the subsequent update and report programs to build a core table of each data item's:
- size, in bytes
- type, alpha or numeric
- name

This formation of the makeup of a data record is the task of the first module of MDBMS, CREATE. The output of CREATE is a sequential, small file called the Driver File. We will use BASIC and an example to clarify the operation of CREATE.

FILENAME?   (CREATE initializes PEOPLE.DRV the Driver file)

NUMBER OF DATA ITEMS PER RECORD?   6

           (CREATE will now loop through the item specification questions 6 times)

THE FIRST DATA ITEM MUST BE THE KEY OR RECORD IDENTIFIER ITEM. LIMIT EACH ITEM NAME TO 10 CHARACTERS.

ITEM #1 Name?   SOC SEC #
A (ALPHA) OR N (NUMERIC)?   A
NUMBER OF CHARACTERS?   11

           (The entries in PEOPLE.DRV at this point are:
006
SOC SEC #   A011

ITEM #2 NAME?   NAME
A (ALPHA) OR N (NUMERIC)?   A
NUMBER OF CHARACTERS?   30

           (PEOPLE.DRV now has:
006
SOC SEC # A011
NAME   A030)

ITEM #3 Name?   SEX
A (ALPHA) OR N (NUMERIC)?   A
NUMBER OF CHARACTERS?   1

           (006
SOC SEC # A011
NAME   A030
SEX   A001)

ITEM #4 NAME?   AGE
A (ALPHA) OR N (NUMERIC)?   N
NUMBER OF DIGITS?   2

           (006
SOC SEC # A011
NAME   A030
SEX   A001
AGE   N002)

ITEM #5 NAME?   SALARY
A (ALPHA) OR N (NUMERIC)?   N
NUMBER OF DIGITS?   5

           (006
SOC SEC # A011
NAME   A030
SEX   A001
AGE   N002
SALARY   N005)

ITEM #6 NAME?   LOVER
A (ALPHA) OR N (NUMERIC)?   A
NUMBER OF CHARACTERS?   20

           (006
SOC SEC # A011
NAME   A030
SEX   A001
AGE   N002
SALARY   N005
LOVER   A020)

"PEOPLE" HAS BEEN CREATED, YOU MAY NOW RUN "UPDATE" TO ENTER DATA RECORDS INTO YOUR FILE AND "REPORT" TO SELECT AND PRINT FROM "PEOPLE."

CREATE has finished its job and built the driver file, PEOPLE.DRV, and initialized the key (PEOPLE.KEY) and data files (PEOPLE.DAT).

Digression for Overview - We now are discussing the three files which makeup the MDBMS system:

1. PEOPLE.DRV, the driver file
2. PEOPLE.KEY, the key file (do-your-own ISAM)
3. PEOPLE.DAT, the data file.

The Driver file contains a description of the format of each data item in a record, and therefore of the entire record itself.

The Key file will contain the user's record identification key (Social Security Number in the example) and will also contain the physical record number that the data is stored in PEOPLE.DAT. The Key file is sortable.

The Data file will contain the data record with the remaining data (Name, Sex, Age, Salary, Lover) and is never sorted. We will also discuss the tag file, and how to sort, later.

File Maintenance and Update

__The Update Program__ - UPDATE is the update program (surprise!) and is best described by continuing our example.

FILENAME? PEOPLE

> (UPDATE looks for a driver file named PEOPLE.DRV, and finding it builds core tables with an arbitrary max of 100 items. In this case there are only six items in each. There are:
> 1. an Item Name table of 6 names
> 2. an Item Type table of 6 types
> 3. an Item Length table of 6 lengths
> UPDATE now will ask whether you wish to Add, Change, or Delete a record.)

After every record transaction in the following example, the contents of PEOPLE.KEY and PEOPLE.DAT will be displayed.

```
ACTION? ADD
SOC SEC #? 531-33-1111
NAME? SKALABRIN VLADOMIR
SEX? M
AGE? 33
SALARY? 42000
LOVER? MOTHER NATURE
```

```
    people.key        people.dat

0002
A531-33-11110001    Skalabrin Vladomir M3342000
                    Mother Nature
```

What has occurred is now instantly replayed.

Upon receiving the user's file name, PEOPLE, our UPDATE program opened the driver file, PEOPLE.DRV, and retrieved:

Number of items = 6
and then
```
           °For I=1 to Number of
            Items
           °INPUT---Name(i), Type(i),
            Length(i)
           °NEXT I
```

UPDATE then asked for the action to be performed, it was ADD - a command to collect from the user all six items in this new record. UPDATE branches to the Add-a-Record routine and:

```
                   °For I=1 to Number of
                    Items
                   °PRINT Name(i);
                   °IF Type(i)="N" branch to
                    Accept-Numeric
                   °INPUT Alpha-String
                   °GO TO Continue
Accept-Numeric     °INPUT Numeric Value
Continue           °CONTINUE
                   °Perform a routine to
                    concatenate (append)
                    sufficient blanks to pad
                    the response to at least
                    Length(i) bytes and then
                    isolate the first Length(i)
                    bytes.
                   °Perform the move of this
                    contiguous segment of
                    bytes into the record -
                    length work buffer.
                   °NEXT I
                    Perform the routine which
                    separates out the key
                    (record identifier - the
                    first Length(l) bytes.
                   °Perform the routine which
                    finds the next available
                    slots in PEOPLE.KEY and
                    PEOPLE.DAT and write the
                    records out.
                   °RETURN
```

UPDATE now repeats its request for the action to be performed.

```
ACTION? ADD
SOC SEC #? 518-18-1818
NAME? ALBRECHT ROBERT
SEX? M
AGE? 38
SALARY? 15000
LOVER? YES
```

```
         people.key           people.dat

0003
A531-33-11110001      Skalabrin Vladomir M3342000
                      Mother Nature
A518-18-18180002      Albrecht Robert     M3815000
                      Yes
```

```
ACTION? CHG
SOC SEC #?  531-33-1111
ITEM NAME? AGE
AGE = 33 = ? 38
ITEM NAME? /END
```

```
         people.key           people.dat

0003
A531-33-11110001      Skalabrin Vladomir M3842000
                      Mother Nature
A518-18-18180002      Albrecht Robert     M3815000
                      Yes
```

After reading this file, UPDATE had:

        Number of Items = 6
        A six element Name table
        A six element Type table
        A six element Length table
        Record Length = 69 bytes (summing the
        lengths)

When AGE (item 4) was requested during the CHG
operation, UPDATE totaled the lengths of items 1,
2, and 3 (those preceding AGE), added 1 and had
the position of the Starting Byte of AGE.

Item Value = MID$ (Work Buffer, Starting
Byte, Length (4)) then returns the value of AGE
from the 69 byte work buffer cum combination data
record.

After accepting the new AGE in Response and perform-
ing the Pad-it-out-to-length(4)-bytes routine, the
Work Buffer is reformed by:

Work Buffer = LEFT$ (Work Buffer, Starting Byte -
               1) + Response + RIGHT$ (Work Buffer,
               69-(Starting Byte -1 + Length
               (i))).

The CHG option is the Update option, and upon its
choice, UPDATE will:

                  °PRINT Name (1);
                  °INPUT Response
                  °Perform the Search Key
                   File routine for this
                   identifier and move the
                   appropriate records into
                   the work buffer.;
        Start   °PRINT "ITEM NAME";
                  °INPUT Response
                  °FOR I = 2 TO Number of
                   Items
                  °IF Name(i) = Response
                   branch to Continue
                  °NEXT I
        Continue °CONTINUE
                  °Perform routine to move
                   Length(i) bytes out of
                   the appropriate area of
                   the work buffer into Item
                   Value.
                  °PRINT Name(i); "="; Item
                   Value; "=";
                  °Accept Response as before
                   and move back into the
                   work buffer.
                  °GO TO Start

Detail of Moving To and From

In our example, the driver file PEOPLE.DRV contains:

        006
        Soc Sec #   A011
        Name   A030
        Sex   A001
        Age   N002
        Salary   N005
        Lover   A020

This moving out of n bytes beginning at position
m, and moving in of n bytes beginning at position
m is the critical capability the computer language
must have to implement the MDBMS concept.

MIS - The Report Option

The value of a general purpose, select, format and
report on-line program is very great for any
computer installation providing data processing
services to its customers.

The picture that keeps returing to mind is of the
innocent eyed business manager who just requested
a non-existent report for tomorrow, and when told
that no program existed and it may take two weeks
to find the time, remarks:

        "You've got the data in your computer, I'm
        just asking you to print it out."

MDBMS and its MIS report option does just that.

Let us demonstrate the use and function of REPORT
(the report program) by assuming we've entered a
number of people into our file PEOPLE and contin-
uing our example.
FILENAME? PEOPLE

PLEASE ENTER THE ITEM NAMES, THE COMPARISON TESTS
AND TEST VALUES TO BE USED.

        ITEM NAME?   SEX
        COMPARISON?   EQ
        TEST VALUE?   M

        ITEM NAME?   AGE
        COMPARISON?   GT
        TEST VALUE?   30

        ITEM NAME?   AGE
        COMPARISON?   LT
        TEST VALUE?   40

        ITEM NAME?   /END

We have just specified a report of all male People
between the ages of 30 and 40.

The mechanism of REPORT works thusly:

REPORT builds core tables of item names, types,
and lengths just as UPDATE did. REPORT then
builds, through terminal interaction, selection
tables of:

1.   item name to be tested

2.   comparison test to be used

3.   value to test against

When the user signifies that he has entered all of
her selection criteria by typing "/END", REPORT
selects the records for printing by:

```
                    °LET T$ = "GO"
                    °Read a record (sequential
                     file reads)
                    °FOR I=1 to Number of
                     Tests
                    °FOR J=1 to Number of
                     Items
                    °IF Name to be Tested
                     (I) = Item Name (J) GOSUB
                     Selection Routine
                    °NEXT J
                    °NEXT I
                    °IF T$ = "GO" PRINT
                    °GO TO START

Selection Routine   °Perform routine to move
                     item (J) from record
                     buffer to Item Value.
                    °Convert Comparison (I) to
                     comparison number (1
                     through 6).
                    °GO TO eq, ne, gt, ge, lt,
                     le, ON comparison number.
           eq       °IF Item Value < > Test
                     Value (I),  LET T$ =
                     "NO"
                    °RETURN
```

The "PRINT record" used above may be replaced with
GOSUB Print Formatting routine. This would be the
case in a full blown REPORT program which allows
the user to specify report headings, define a
subset of a record's items to print, and allow the
formation of new report items formed from record
items and/or previous report items.

This type of sophisticated REPORT program might
have the following starting dialogue:

HOW MANY COLUMNS IN THIS REPORT?   4

COLUMN #1 HEADING? Employee Name
DATA (D) OR FORMULA (F)? D
DATA ITEM NAME? NAME

COLUMN #2 HEADING? AGE IN MONTHS
DATA (D) OR FORMULA (F)? F
NUMBER OF COMPONENTS? 1
COMPONENT NO. 1 FROM? D
DATA ITEM NAME? AGE
MULTIPLIER? X12

COLUMN #3 HEADING? ANNUAL SALARY
DATA (D) OR FORMULA (F)? D
DATA ITEM NAME? SALARY

COLUMN #4 HEADING? SALARY/AGE RATIO
DATA (D) OR FORMULA (F)? F
NUMBER OF COMPONENTS? 2
COMPONENT NO. 1 FROM? D
DATA ITEM NAME? SALARY
MULTIPLIER? X1
COMPONENT NO. 2 FROM? R
REPORT COLUMN NO.? 2
MULTIPLIER? X1
COMPONENTS 1 & 2 OPERATION? /

The REPORT program can be given these capabilities
of selecting records for printing, formatting the
output, forming report items from combinations of
data items and previous report items and could
also be equipped to print grand totals of columns,
subtotals for columns, and counts of the reoccurrence
of specific items.

The actual print formatting would be done by
defining a 132 byte output record (for line printers)
and performing the "Move n bytes into the record
beginning at positon m" routine.

Sorting the Report

The need to sort a report into a particular order
introduces a redesign into MDBMS and MIS components.
Considerations of remaining general purpose,
increasing processing speed, etc., dictate the
REPORT program outlined above to be segmented into
a SELECT program which performs the selection of
records to be processed, and a format-and-print
only REPORT program. One alteration is made to
each of SELECT and REPORT, and a new program SORT
is defined.

> °SELECT writes the record key (Social Security
> Number in our example) and the corresponding
> data record number into the last 20 bytes of
> a 120 byte TAG file, and blanks the remainder
> of the TAG record.
> °If the selected records are to be reported in
> a sorted order, SORT will determine the sort
> keys by item name, from major to minor,
> access the data file by the record numbers in
> the TAG file, and move the corresponding item
> values into the first 100 bytes of the TAG
> file.
> The TAG file is sorted as though the entire
> 120 bytes were one sort key.
> °REPORT sequentially uses the record keys and
> data record numbers from the last 20 bytes of
> the TAG file to read data records and print
> its report.

Do Your Own ISAM

The discussion presented in the preceding pages
mentions the Key File and record keys several
times. While MDBMS and MIS is operable completely
with sequential files, it performs as though it
had Indexed Sequential Access Method (ISAM) capa-
bilities.

ISAM gives the user and programmer the capability
of referencing the storage location of a data
record by its user-defined key (Social Security
Number, Catalog Order Number, Course Code, etc).
The only restrictions imposed are a maximum size
(bytes) and uniqueness of record keys (no two
alike).

The large system ISAM's are markedly faster than
the do-it-yourself concept outlined below, but
often have severe operational limitations. Some
of these are:

1. Files _must_ be initially built via an ISAM loading utility.
2. When sufficient records have been added to fill the allocated ISAM file or its overflow area, it must be rebuilt.
3. Some ISAM's (e.g., IBM 1130) do not allow adding a record in the same program which is updating the file.

In particular, most small systems do not have an ISAM utility, and those that do have restrictions which make their use in a user controlled environment prone to operational errors.

MDBMS fulfills the ISAM concept in its Key File and search routines.

Consider the Key File as a unique sorted file of 20 byte records, each record containing a unique record key (15 bytes) and an associated data record number (5 bytes). Set Key File record #1 aside as a special record to be considered later.

ISAM Search

Each program in the MDBMS series must be able to locate the data record defined by a user key. In our example, this means that each program must be able to take the Social Security Number 518-18-1818 and locate the corresponding data record
 ALBRECHT ROBERT   M3815000 Yes.

At least two search schemes are available:

1. Perform a binary search of the Key File.
    For- fairly concise coding and needs no extra core tables.
    Against- requires a significant amount of extra disk accesses (10 for a 1,000 record file).

2. Build a in-core directory of highest-key-in-this-disk-segment values.
    For- fast, only requires one disk access to locate a key where segment = buffer read.
    Against- may require a large core table if the file is large and the segment is small, or trade-offs of table size versus number of segments per high-key, and the subsequent increase in disk accesses.

In either case, the logic of the search routine is the same.

The physically first record of the Key File contains a count of how many records (including #1) are in the Key File. This record is maintained with sufficient leading blanks to ensure that it will always sort to the front and remain the first (#1) record. The record count is updated by UPDATE if records are added or deleted.

An initialized Key File for a system capable of dynamically adding records would contain:

(15 spaces ....) 00001

An initialized Key File for a system requiring allocation of file space would contain:

(15 spaces ..) 00001
zzzzzzzzzzzzzzz00001
zzzzzzzzzzzzzzz00002
zzzzzzzzzzzzzzz00003
zzzzzzzzzzzzzzz00004

The search and write routines for an ADD would be:

1. read the #1 record and get COUNT (done once).
2. read the COUNT+1 key file record and get the record number of the first available data file slot. Place this data file record number in RECNUM.
3. Write the user key for this new record in the Key File at COUNT+1, and set COUNT = COUNT + 1 in core. No further accesses of the #1 record would be required until UPDATE ends if this is a single user environment.
4. after completing the formation of all the data required for this new record, write the data into the Data File at RECNUM.

Both the binary and directory search techniques require that the Key File be sorted at some time, and also need an additional routine to handle that unsorted portion of the file created by ADD's while UPDATE is in operation.

Briefly, the search routine must be capable of:

1. fast searching the sorted portion of the Key File, and
2. sequentially searching the add-on portion.

SUMMARY

The Mini Data Base Management System described on these pages can be implemented in any computer language having the capability of moving an arbitrary (defined at run time) segment of bytes from and to a buffer area.

It is a single set of programs, CREATE, UPDATE, SELECT, SORT, REPORT and TRANSFER*, which allow the creation and maintenance of user defined data files, and the selective report generation from these files.

This system has been implemented in BASIC, COBOL (with a macro), and RPG (with an assembly subroutine).

*TRANSFER looks at the old and new Driver files, and moves selected data from the old Data file to the new Data file.

DESIGN CONSIDERATIONS IN THE IMPLEMENTATION OF A HIGHER-LEVEL LANGUAGE
Including Details of the Internals of Cromemco "16K BASIC"

William F. Wilkinson
Shepardson Microsystems, Inc.
20823 Stevens Creek Blvd., Building C4-H
Cupertino, California, 95014

Purpose and Scope

This paper will try to serve several purposes: First, it will introduce at a tutorial level some of the more important considerations that must be weighed in the design of a compiler or interpreter. It will also present a few of the techniques commonly employed in various language processor implementations. And, finally, it will discuss in some detail an actual and successful BASIC language interpreter.

As noted, a sizable portion of the material herein will be of interest to those who might be relative novices in the field of system-level programming. On the other hand, it is hoped that more advanced programmers will appreciate the information presented relating to the internals of Cromemco's new stand-alone and disk-based BASICs.

Terminology

Some of the terms used within this paper may be unfamiliar to many readers. As an aid to better understanding of the contents herein, a glossary of some of the more important and/or complex terms has been included as an appendix.

The first usage of a word or phrase that may be found within the glossary will be flagged by a double asterisk (**) for the reader's convenience.

Types of Language Processors

In subsequent subsections, the advantages, disadvantages, relative memory requirements, and relative user-program execution speeds of the following types of language processors will be briefly considered:

    COMPILERS
        1. One Pass
        2. One Pass plus Assembly
        3. Multi Pass
    INTERPRETERS
        1. Unaltered Source
        2. Keyword Token (Partially Syntaxed)

        3. Fully Syntax Checked
        4. Incremental Compilers
        5. Compile and Go

An example of each type as represented by an available software product will be given. However, this author can not claim with absolute certainty that the examples given are correct since, in some cases, external indications were the only indications as to type.

Compilers

A compiler is a type of language processor which reads a user's source program one or more times and produces (as final output) actual machine language code.

Typically, programs which are compiled need 'support' from system 'libraries' of subroutines. These subroutines are used when the compiled program needs to perform a function too complicated to be included directly in the machine code. Examples of routines found in system libraries include I/O (Input/Output) routines, transcendental functions (Log, Sin, etc.), and array processing. On machines such as IBM 370's which include floating point arithmetic in their instruction set, such instructions are obviously included in the object code. But for a microprocessor lacking these capabilities, the routines for floating point arithmetic must also be included in the system libraries.

Advantages of compilers include generally lesser memory requirements and faster user-program execution speeds (because the output code really is machine language). Note, though, that some of the speed of compiled code is lost on microprocessor-based systems where significant amounts of floating point arithmetic is performed. Even on small systems, however, there is no denying the advantage of compilers when performing integer based routines.

Disadvantages of compliers include the fact that the machine code output is often very difficult to debug and that turn-around time between program revisions is longer (since both the debugging and the compliling consume significant amounts of time).

Sophisticated (especially disk-based) compilers can overcome some of the disadvantages noted by including an equally sophisticated debug program that is aware of where variables** are stored in memory, where program lines have been compiled, etc. This can produce an interactive debugger with trace capabilities similar to an interpreter. Even the best of such debuggers still cannot make program modification as easy as an interpreter can.

One Pass Compilers are often a good choice for microcomputer systems, especially those without a disk. They read the source code** only once, simplifying input from serial devices such as teletypes and cassettes. Nevertheless, the better ones are often capable of local optimization** on at least a per-statement basis. On a system with a given amount of memory, one-pass compilers often cannot handle source programs as large as those compiled by other techniques; this is especially true if the compiler is expected to output its object code** directly to memory instead of to a mass storage device.

An extremely good one-pass compiler is a true example of the system designer's (and programmer's) art: after all, such a compiler is actually performing several functions at once (see next section).

The Microsoft-produced Fortran IV Compiler for the 8080 and Z80 is a one pass compiler. It is available from several sources now, including Cromemco and, of course, Microsoft.

One Pass plus Assembly Compiler. This is also a type of compiler that reads the user's source code only once. But it then produces an assembly language intermediate "source" program which then must be assembled by a more or less standard one or two pass assembler**.

The chief advantage of this type of compiler is that it is the easiest kind to design and implement! In fact, given a system with a good macro-assembler, an enormous amount of "housekeeping" is removed from the compiler-writer's bailiwick.

Often, this type of compiler will do little or no optimizing. The extra passes through the assembler can be a nuisance to the user if not fully automated.

On the other hand, the person familiar with assembly language can often take advantage of the availability of the assembler listing and thereby significantly reduce debugging time.

Also, since the compiler is simpler, it generally requires less memory to run than a one-pass type. If the assembler used is similarly compact, significant memory savings may be realized at the expense of compile time.

An interesting note here is that Data General's Fortran IV compiler is a one-pass-plus-assembly type (two pass assembly, at that) and was, for quite some time, the only Fortran compiler offered by the company! This contrasts sharply with their "Fortran V" compiler discussed in the next section.

Sidelight: one pass plus assembly type compilers were especially popular in machines of a few years back because of their reduced memory requirements. When memory was extraordinarily expensive and disks were cheap (compared to memory), the approach made a lot more sense than it does in today's cheap-memory market.

Multi-Pass Compilers. These compilers, as their generic name implies, must read the user's source program several times before producing machine code output.

The primary reason for writing a multi-pass compiler is usually to facilitate global optimization**. They are typically capable of significantly more efficient object code than the one-pass types (though some two-pass compilers might only be simplifications of one-pass types). The object code may or may not be more compact, but it is virtually always significantly faster at execution time.

The only real disadvantages of multi-pass compilers is that the compiler itself is usually a huge program which requires much machine time to operate. A secondary disadvantage is that once global optimization has taken place the dis-assembled object code might in no way resemble the input source: this can be a real handicap at debug time, especially if the compiler has a bug in it! (Which actually happened to this author.)

IBM's Fortran Level H is an example of a multi-pass optimizing compiler. In fact, with this compiler, the user may even specify one of three of levels of optimization to be attempted. Data General's Fortran V compiler is rumored (rumored!) to be a thirteen pass optimizing compiler. In any case, it produces remarkably efficient code for a minicomputer compiler.

Some Interim Notes

Before proceeding to a description of types of interpreters, a comment on the compiler versus interpreter controversy seems appropriate.

Generally speaking, the only programs that really need computational speed are

those doing a large amount of "number crunching". But, if a program involves any significant amount of floating-point arithmetic (especially transcendental functions), the time required to perform such routines is usually significantly more than the time spent on the supportive control functions. Similarly, programs involved in primarily I/O processing on devices no faster than floppy disks are generally seldom bound by the speed of the control operations being performed.

So what kind of program really needs the extra measure of speed provided by compilers versus the interpreters? There really are many such programs: real-time process control, music processors, programs which need to run on memory-limited systems, and others. But the hobbyist should probably carefully consider whether the programs he is writing fall into any of these categories. If not, he should seriously consider using an interpreter, for reasons which will be noted in the next section.

As a final note on compilers, it should be noted that one feature of Fortran compilers is they may be made to process a particular subset of Fortran (ANSI standard Fortran), and other compilers on other systems should be able to handle the code with little or no modification.

Interpreters

The primary difference between compilers and interpreters is that the latter were designed to facilitate interactive computer usage while the former come to us from the days when batch processing represented the only computing power available.

Generally, an interpreter will allow the user to sit down at his terminal and/or computer and enter a program one line at a time. He may enter several lines and then RUN the resultant partial program, verifying that it is performing the function assigned to it. He may proceed in this manner, adding a few program lines at a time and checking that they function properly, until the whole program is entered and checked.

Depending on the interpreter being used, several additional aids to debugging may be available: Most Basics allow the user to execute several (if not most) statements in an immediate or direct mode. For example, in Cromemco Basic, only the DATA, DEF (user function), and FOR/NEXT statements may not be directly executed. Virtually all interpreters

allow the user to halt program execution through some break or escape key, after which he may examine the value of variables, status of flags, etc. Many basics even allow the user to change, delete, and insert program lines after executing such a break and then continue execution of the modified program!

But various types of interpreters have varying abilities, some of which are discussed in the sections which follow.

Unaltered Source is a term used herein to mean that the program as actually typed in by the user is stored in essentially unaltered form (i.e., extraneous spaces might have been removed, etc.) in memory. The interpreter then executes statements by actually reading the source and operating on the commands therein. No validity check is made on the source until execution is attempted. Some systems use this "feature" to advantage: since no checking is performed, any data may be entered and treated as a program line. Since Basic (for example) will rearrange incoming lines in numerical order, this allows the user to implement a poor man's text editor.

The disadvantages of this method of interpretation are numerous: chief of which is relatively slow execution speed (although even here a clever designer can earn his money). Another significant problem is that the user has no idea that he has made an error in entering a line until it is actually executed, which may not occur for some time in a complex program.

There are several so-called "tiny Basics" available which are implemented in this fashion. Despite their name, many of these Basics actually can handle some very sophisticated programs.

Keyword Token (or Partially Syntaxed). This term is used to classify a type of interpreter that scans incoming lines of program for the keywords defined in the syntax** of the language, converting them to internal format bytes (tokens). However, a Basic interpreter of this type is liable to allow a statement of the following form without even protesting about the utter nonsense of it:
    100 NEXTSTEPFORPRINT
Such a Basic would be very likely to list that statement back out as:
    100 NEXT STEP FOR PRINT
It was capable of recognizing all those keywords, but did not realize that they were used incorrectly. (Of course, the nonsense would be found at execution time and an error message would result.) It seems obvious, then, that in some ways there is little advantage in this type of

interpreter versus the unaltered source variety. Were we only concerned with validity checking, that would be true; however, the keyword token concept allows somewhat higher execution speeds, since there are fewer bytes to scan to determine statement type and since the appropriate keyword token can immediately vector off to its processing routine rather than having to first decode it and then vector.**

Both the keyword token and unaltered source interpreters are capable of performing all their feats in the direct mode, although many interpreters nevertheless exclude certain functions as direct because they may not truly make sense unless executed within a running program.

Many, many of the available microprocessor Basic's are of the keyword token type. Prime examples are most (all???) of the Basic interpreters written by Microsoft, including,as a specific example, Applesoft as distributed by Apple Computer.

A sidelight: This type of interpreter is popular with system programmers because it is reasonably fast at execution and yet does not require the double level of complexity necessary to the type of interpreter discussed next. Shepardson Micro-systems, for example, felt much more constrained in space than in speed when its 6800 Basic was designed, and hence chose this method.

Fully Syntaxed is a term used to describe a type of interpreter which actually completely takes apart the user's input line and reassembles the pieces as internal codes (tokens), checking for syntax errors right then and there. As an example, assume a Basic which accepts only standard Dartmouth variable names (single letter, optionally followed by a single numeral); consider the result when the user enters the following line:
    530 PRINT FORB
The keyword token type interpreter will most probably syntax that as:
    530 PRINT FOR B
It sees the keyword "FOR" and processes no further. The fully syntaxed type of interpreter, however, would realize that a FOR keyword cannot appear after a PRINT statement. Depending on whether the interpreter in question allowed logical expressions using the keyword OR, it might syntax that line as:
    530 PRINT F OR B
If,however, it did not allow logical operators, it would flag that statement

as being in error. Another level of complexity is added to the problem if the interpreter allows both logical operators and long variable names: is FORB the variable "FORB" or is it the expression "F OR B"? Since both constructs are legal, it is the designer's job to decide on one or the other and then clearly state his choice in the user's manual.

Generally, a fully syntaxing interpreter enjoys all the advantages of the keyword token type (i.e., faster execution speed through token vectoring) plus the added advantage that the user is made aware of his syntax errors as soon as he types them in.

An example of this style of interpreter is Apple Basic, the integer-only Basic placed in ROM in the APPLE II. This is a Basic complete and fast enough to enable the user to program video games directly in Basic.

Incremental Compiler is actually a type of interpreter which takes the fully-syntaxed concept a step further. An incremental compiler not only fully checks for correct syntax as the source is being entered, it also immediately resolves any references to undefined items! The internal code produced may or may not be actual machine language; in fact, it is more likely that a more generalized form of tokens will be produced.

The CROMEMCO 16K BASIC interpreter is an incremental compiler, and details on its inner workings will follow in later sections. Another example of this type is Data General's Basic.

The primary advantage of the incremental compiler is execution speed. Since variable and line number references are already resolved, no execution time is spent searching tables for (for example) variable name match. User's of Cromemco's Basic enjoy excellent execution speed even though the system provides them with as many, if not more, features as any other microprocessor-based Basic.

The primary disadvantage of this type of interpreter is that the interpreter itself requires significantly more memory than other types. For example, Cromemco's Basic requires 16K bytes for the stand-alone version and somewhat more for the disk-based.

Compile and Go is a type of interpreter which is often an interpreter in name only. Some of this type might be more accurately described as interactive compilers. A typical method of implementation involves a two-step interpreter: the first part functions only as an editor, allowing the user to change his program. It may or may not be a partially or fully syntaxing editor.

The second part of this interpreter is
the actual compile-and-go part: it
accepts the edited source (which may
have even been converted into tokens
for ease of compiling) and resolves
all line number and variable references
and then runs the resultant object.
This type of interpreter is easy to
recognize: it is the type that, when
the user types RUN, responds immediately
with error messages regarding missing
END statements, GOTOs to non-existant
lines, etc. (The other interpreters
discussed usually cannot give these
error messages until the line in ques-
tion is actually executed.)

The obvious advantage of this
type of interpreter is that it is
fast. The HP2000 series time-shared
BASIC systems use this methodology
and are often able to successfully
support significantly more users than
(for example) Data General can with its
incremental compiler, even though the
DG machine may be faster and more pow-
erful than the HP CPU.

The biggest disadvantage of
the compile-and-go interpreter is that
usually very few operations are avail-
able in direct (immediate) mode. On
the HP2000F and below, for example, the
user cannot even print out the values
of variables after encountering a pro-
gram break or error. In this respect,
the user is often more uninformed than
he would be with even a straight compi-
ler, and debugging can sometimes be
somewhat time-consuming.

More On Interpreters

We have seen that the primary
advantages of interpreters over compi-
lers are in the area of user interaction
and debugging. Their major disadvantage
lies in their memory requirements: not
only must the interpreter be memory
resident, even tokened source code tends
to occupy more space than truly compiled
code. Of course, speed is also a con-
sideration; but many, this author in-
cluded, feel that speed is often not
overwhelmingly important in hobbyist
applications. Of course, nobody likes
to wait forever for a program run, and
so Cromemco Basic was designed to be
as efficient as possible while retaining
virtually all other desirable features.

Another point worth mentioning
is that memory prices are still drop-
ping at a fantastic rate, so complex
programs are coming into the reach of
more and more people.

And finally, with regards to
what can be called the state of the

art in microprocessor-based BASIC inter-
preters: Watch this space.

CROMEMCO 16K BASIC

The name of this interpreter is
somewhat of a misnomer in the case of the
disk-based version: it currently occupies
18.5K and is likely to grow. Cromemco
Basic was designed from the start to be
many things for many people. It can def-
initely be considered a business-oriented
product: it has easily accessible random
and sequential files, three different
precisions of arithmetic (including 14
digit decimal floating-point), and user-
program trappable processing of errors
and escape requests. All this was accom-
plished with little sacrifice in speed:
the interpreter handles double-precision
arithmetic as fast as most interpreters
do single-precision. Of course, all these
features added to the memory requirements,
but the biggest memory-eater was the imple-
mentation used, namely the incremental
compiler method.

An incremental compiler requires
a number of tables both in the code of
the interpreter and in user program space.
These tables will be discussed in later
sections and then an example program line
will be followed through the entry and
execution phases.

Syntax Related Tables

As each program line (or direct
command statement) is entered from the
keyboard (or some other I/O device), it
is checked for correct language syntax
and converted into an internal format.
Actually, the routine which does the
syntax checking also converts source items
into internal "tokens" at the same time.

Basic's syntaxer is actually driven
by a set of syntax tables, the purposes
of which are described following.

The Reserved Name Table contains a list
of all statement, function, and operator
names that are recognized by Basic. This
table is used to equate the user's ASCII
input to a particular internal token.
Thus, in addition to the reserved name,
each entry contains the token to be used.

The Main Syntax Table. Once the first
reserved name in a statement has been found,
the statement may be classified as to type.
The type directs Basic to an entry in the
main syntax table, where an entry is simply
made up of a list of required and/or op-
tional further items (actually their equated
tokens) that must be found in the statement.
If a required item is not found in the

proper position in the input line, a syntax error is generated. This table is really quite complex, as even such things as arithmetic expressions in all their variations must be covered. There are even syntax subroutines included here, so that a statement type may require (for example) a subscripted string name by calling for a subroutine which in turn requires a string variable, a left parenthesis, subscripts, and a right parenthesis. (Please note that this is not actually how the search is performed; it is actually more segmented and more highly organized than stated here.)

The syntax table contains other information about statement types, such as whether they are allowed in direct mode, within a program, or both places. It can also declare that a statement must be the last one in a line or that subsequent statements on the same line are permitted.

The Binding Strength Table is more properly described as a run-time table, since it determines the priority in which operators (+,-,AND,NOT,etc.) are to be executed. It is included here because it is used to relate the internal tokens for the various operators to their priorities.

On Listing a Program

After a program has been converted to internal token form, it is obviously desirable to be able to implement the LIST statement. On unaltered source types of interpreters, this is simply a matter of dumping the memory contents back out to the list device. In Cromemco Basic, the problem is significantly more complex: Each token must be reconverted to its ASCII form and (as we shall see later) constants must be reconverted to external format.

Tables in User Program Memory Space

Generally, these types of tables may be referred to as "run-time" tables, though many of them are established at program entry time. They are the fundamental tables which the interpreter uses in executing a user program.

The Statement Table is not strictly speaking a table. It is simply an area of memory where all program lines are stored after they have been converted to internal format. The most distinguishing feature of this table in

Cromemco Basic is that the last line entered is always at the end of the table. Only when a line is deleted or modified is it necessary to "slide" the table (in order to recover the space occupied by the deleted parts). Also note that the program's line numbers are not stored in the statement table. Instead, the line numbers are found in the

Line Number Table. This table is also known as the Label Table. Herein are stored the actual line numbers. Also in this table each entry has the address of the beginning of its corresponding program line within the statement table. The third item in each entry in this table is the address within the table of the next logical succeeding line (i.e., the line next to be executed unless program flow is altered by a GOTO, GOSUB, or NEXT statement). Note the implication here: even this table is not organized in line number order! As each new line is entered, its label entry is simply appended to the table and the logical successor pointers are updated as needed.

Variable Table. When a variable is first defined (which occurs as the first line using is entered), an entry is created for it in this table. Types of information contained herein are many: All variables have their name ('A1','A$','E') and variable type (string, integer, short or long floating point, and/or array) stored here. In addition, scalar variables (non-array numerics) have their actual value stored here. Strings have their dimension and current length stored also. Arrays have the number of dimensions in use noted as well as the maximum value of each dimension. In addition, arrays and string entries contain the address of the actual location of the data for each. The data so addressed is located within the

String/Array Table. Again, this "table" is not so much a table as simply a section of memory reserved for the storage of array and string data. Of the tables mentioned so far, this one is unique in that is is allocated (expanded) at runtime. That is, since statements of the form
        310 INPUT J
        320 DIM A$(J),B3(J,2)
are legal, arrays and strings cannot have their space allocated to them until they are actually dimensioned during program execution.

The For/Next Table. An entry is placed in this table whenever a FOR statement is encountered. Information kept here includes the address of the corresponding variable

table entry, the size of STEP requested (or implied), the terminating TO value, and the address of the entry in the label table corresponding to the line logically following the FOR statement. If a FOR statement is executed using a variable with an entry already stored in the FOR/NEXT table, the old entry is deleted, the table is adjusted, and the new entry is appended.

GOSUB/RETURN Table. A very simple table, this consists simply of the address of an entry in the label table which corresponds to the line logically following the GOSUB statement.

User-Defined Functions Table. Since only user-defined functions FNA through FNZ are allowed in Cromemco Basic, this is a fixed length table with 26 entries, each active one of which contains the address of the label table entry corresponding to the line where the function was defined.

AN EXAMPLE

There follows an actual example of what happens internal to Cromemco Basic when a program line is entered (syntax time) and then executed (run time). It is assumed that any existing program and/or data has been removed via a SCRatch command and that the example line shown is the first (and only, though this is not significant) line entered towards creating a new program.
Herewith the example line:

100 IF A2 = 3.7 THEN 300

Syntax Time

The first thing detected is the line number, '100'. This line number is placed in the label table along with the address of the beginning of the statement table (where the encoded line will be stored). Since this is the only entry in the label table, the successor pointer will be set to zero to indicate no successor. Since the label table works from the top down, the pointer to the bottom of the table would be updated to indicate where to place the next entry. So far, then, the label table looks something like this:

2 bytes --pointer to statement in statement table
2 bytes --pointer to successor label (zero, currently)
3 bytes --the line number, 100, in BCD (5 digits significant)

Next, the keyword 'IF' is encountered and recognized as a valid statement type.

It is translated to a single byte internal token and the syntax table entry for 'IF' is accessed. The syntax for 'IF' requires that the keyword be followed by an arithmetic expression, so the appropriate syntax subroutine is called.

Without going into detail as to what is a valid arithmetic expression, and how the syntaxer determines that it has or has not encountered one, it is sufficient to note that certainly 'A2 = 3.7' would be considered valid. That being so, the variable 'A2' makes its first appearance and must be added to the variable table. It's entry would look something like this:

2 bytes -- the variable name ('A2')
1 byte -- the variable type, assume a 02, short floating point
4 bytes -- the actual value of the variable, in the length required for its type... set to zero until altered

The statement table would receive a byte signifying that a numeric variable had been found followed by two bytes which are the address of the entry for 'A2' within the variable table.
The equals sign ('=') would be recognized as an operator, and its internal code stored next in the statement table.
The value '3.7' would be recognized as a floating-point constant (integer constants don't have decimal points), and a routine would be called to translate it to internal format. The statement table would receive a byte signifying a particular type of constant (short-floating here) followed, and then the internal constant would be stored.
The 'IF' syntax requires that the expression be followed by a 'THEN' keyword. When it is found, its internal token is stored in the statement table.
'THEN' may be followed by either another whole new statement or (as in this case) by a line number. The processing of this line number is partly what qualifies the Cromemco Basic interpreter to be called an incremental compiler.
The lable tabel is scanned for a line number '300'. When it is not found, an entry is made for it similar to the one previously made for line '100'. However, for line number 300, the address of its corresponding line in the statement table is set to an illegal value to indicate that it doesn't exist! (But, later, if a line 300 were entered, the addressed would be changed to reflect the fact.) Also, the entry for line 100 is updated to reflect that its logical successor is line 300 (the next sequentially numbered line).
The statement table receives a token to indicate that a label table address

will follow, and then the address of the
label table entry for line '300' is
stored. Finally,a byte of zero (00)
is stored to indicate the end of line.

At this point, then, the statement
table will contain the following:

1 byte	--	token for the 'IF'
1 byte	--	token, a variable table address follows
2 bytes	--	the variable table address for 'A2'
1 byte	--	token for '='
1 byte	--	contains 02 hex to indicate that a short floating point constant follows
4 bytes	--	the internal form of the constant '3.7'
1 byte	--	token for 'THEN'
1 byte	--	token, a label table address follows
2 bytes	--	label table address for line number 300
1 byte	--	value 00 hex, indicates end of line

Run Time

Assume now that the user requested a
RUN of the program (and, incidentally, even
this simple statement must go through the
syntax process before it is executed!)

Basic searches out the first logical
line of the program from the label table
and then performs the following steps:

First, the address of the statement
within the statement table is found. Then
the first byte of the statement is used to
vector to the routine responsible for pro-
cessing IF statements.

The 'IF' processor knows that what
follows is an expression, so it callsthe
expression execution routine to evaluate
'A2 = 3.7'.

The expression evaluator notes that
the first item is a variable, so it uses
the variable address to extract the value
of the variable from the variable table
entry for 'A2'. It places this value in
an area of memory referred to as the argu-
ment stack.

The '=' operator (note: this is <u>not</u>
the same thing as the assignment operator
'=' used in LET statements) is then pushed
onto another Basic-maintained stack known
as the operator stack.

The constant token causes the short
floating point constant '3.7' to also
be placed on the argument stack.

Since there is no more to the expres-
sion, the expression evaluator calls on
the processing routine for the equal oper-
ator. The equality processing routine
extracts the two values (contents of

'A2' and the value '3.7') and returns either
a one (1) if they are equal or a zero (0)
if they are not. The expression evaluator
is finished and returns with this value to
the 'IF' processor.

At this point let us examine what
happens if the value returned is '1' (which
it can't be in our example). The 'IF'
processor ignores the 'THEN' token (it has
to be there, the syntaxer passed it!) and
encounters the label table token. This
implies a change in program flow to the
line specified in the table entry. So the
'IF' processor simply places the entry en-
countered in Basic's own 'next logical
statement' location. It then checks to
ensure that a line really exists to corres-
pond to the entry. Lo and behold, the
line does not exist and Basic issues the
error message, "GOTO UNDEFINED LINE NUMBER".

Now backtrack a bit and assume that
the expression evaluator had returned with
a '0' instead (which it would in our example).
In this case, the 'IF' processor simply
aborts processing of its line and returns
to allow Basic to execute the next logical
line. When Basic encounters the next log-
ical line ('300' in our example) it checks
to see if the line exists (it doesn't here)
and executes it if it does. If it does not,
Basic ignores the line and "falls through"
to the next logical line. Since line 300
does not exist, and since there is no
logical successor Basic would print out
"***END***".

Summary

The idea behind presenting this
example was that in so doing the reader
might be presented with a better under-
standing of the beneficial effects of
the incremental compiler.

In many Basics, several operations
described herein would have proceeded much
differently. As examples, the variable
'A2' might cause a sequential scan through
a variable table looking for a match on
the ASCII characters; the constant '3.7'
might have to be converted from ASCII to
internal format; and the processor might
have had to sequentially search lines in
order to locate line number 300.

In Cromemco Basic, there is no speed
penalty for using any and as many variables
wherever desired in the program, there is
no reason to put certain statements (i.e.,
subroutines) first in the program, and
constants in programs execute as fast or
faster than variables.

No attempt has been made here to
explore the various features of Cromemco
Basic. This paper was intended primarily
to show why the particular interpreter
design used was chosen.

GLOSSARY

There follows a glossary of some of the more complex terms as well as some of the terms with implied meanings as used in the body of this paper.

Assemblers, One and Two Pass. Assemblers are language processors of sorts that translate assembly language into machine code. One pass assemblers usually must either play "tricks" or put limitations on the user in order to function. When a one-pass compiler is implemented, it must actually perform the functions of interpreter and assembler, all in one pass. Suffice to say this implies several programming tricks. When a one-pass-plus-assembly compiler is implemented, in often uses a macro assembler for the assembly phase, thus allowing the compiler writer to generate his own intermediate language-sensitive code which the assembler is "taught" to recognize.

Source and Object Code. Generally, source code is the original typed (keypunched, handwritten, etc.) program as entered into the machine, stored in characters recognizable to the language processor (ASCII representation is almost universally used for microcomputers). As a special case, though, Basic often considers source code to be what the user entered after it has been stripped of blanks (spaces) since the original Dartmouth Basic was not sensitive to the presence of blanks.

Object code might mean the actual machine code output by an assembler or compiler, the intermediate code output by a one-pass-plus-assembly compiler, or even the internal token form output by the first phase of a Basic interpreter of the keyword-token or incremental compiler variety.

Optimization refers to a technique often found in compilers whereby like expressions are recognized and extracted from more complex expressions, statements, program loops, and even whole programs. For example, even most moderately good compilers could optimize the following:
$B(J+K-1) = B(J+K-1) + 1$
The sub-expression '(J+K-1)' would be calculated before any other part of the line is executed.

A somewhat better optimizer would be able to treat that expression as simply:
$INCREMENT\ B(J+K-1)\ BY\ 1$
Generally, Local Optimization refers to that done within a single expression (or statement, etc.) while Global Optimization refers to broader, multi-line or even multi-subprogram optimizing.

Syntax Checking refers to the process of validating the executability of a (set of) statement(s). In other words, the language processor must ensure that the statements given to it follow the rules of the 'grammar' of the language which it compiles/interprets. For example, the statement
$701\ FOR\ I = 5\ TO\ 100\ STEP\ 5$
may work great in Basic, but a Fortran compiler would get lost after the first three or four characters and should give the user a "syntax error" message to tell him something is amiss.

"Syntax" is used loosely within this paper as a noun and verb in various forms, and where appropriate should be thought of as "Syntax Checking".

Vector, as used herein, refers to the technique of using a code or coded value to access a table of similarly encoded items, the idea being to choose from several possibilities. For example, a one byte code could be used to "vector" into a table of 256 different addresses, each of which might represent a different statement type, function to be performed, etc.

= =

The author would like to acknowledge the permission of Cromemco, Inc., owner of the 16K BASIC described herein, to divulge the details of their product.

PLEASE NOTE that Cromemco is the owner of this language, although produced by Shepardson Microsystems, Inc., and any requests for copies of the interpreter, documentation, or simply more general information must be addressed to them.

Shepardson Microsystems is the owner of the 6800 Basic noted herein, but it is currently available only on an OEM basis.

Inquiries about the design philosophy, etc., of these and other software products are invited, and comments on any of the foregoing would be very much appreciated.

AN ARITHMETIC EVALUATOR FOR THE SAM–76
LANGUAGE

Karl Nicholas
Box 257, Route 1
Pennington NJ 08534

```
| |                                                               | |
| |            Arithmetic Evaluator - Karl Nicholas               | |
| |_____| |
```

This example demonstrates two basic algorithms; substitution and a bit of
systematic juggling. What this example does is enable the user to express
equations in a format approaching that of APL however still completely
compatible with SAM76 of course. The user invokes "EXP" (expression),
arithmetic expression = value. One can get the expression scanned in either
direction by using neutral or active fetches. A normal active fetch will
give a normal left to right scan. For an example of the left to right
scanner:

```
{}-------------------------
{}  %EXP,2+10\2/=6
{}
  -------------------------
```

or an example of the right to left scanner:

```
{}-------------------------
{}  &EXP,2+10\2/=7
{}
  -------------------------
```

or to get what you want from either scanner:

```
{}-------------------------------
{}  %EXP,2+{10\2}/=7
{}
  -------------------------------
```

As you can see, hierarchy is established by use of curly braces, they can be
infinitely nested, of course. The user defines all other functions by use of
the "define" function. This function creates two simultaneous lists where in
the first is the symbol to be used in the evaluator and in the second is the
SAM76 function or user defined function to be substituted in its place. The
user invokes "define", first list append, second list append. For example:

```
{}-------------------------
{}  %define,+,ad/=
{}  %define,\,di/=
{}  %vt,list1/=
{}  ,+,\
{}  %vt,list2/=
{}  ,(,ad,),(,di,)
{}
  -------------------------
```

This will cover the functions used in the previous example. A view text on define reveals:

```
{}--------------------------------------------------
{}  %vt,define/=
{}  %dt,list1,&ft,list1/(,([1]))/`
{}  %dt,list2,&ft,list2/(,(,[2],))/
{}
{}--------------------------------------------------
```

This one is simple; the extra commas in list2 are there to separate numbers from functions for the evaluator.

The procedure that tests for the neutral and does the substituting is "EXP". Looking at it we see:

```
{}--------------------------------------------------
{}  %vt,EXP/=
{}  %ni,`
{}  (%dt,arith,([1])/%pt,arith,{,}%ft,list1//`
{}  %EXP11,%arith,<%EXP11,>,</>%ft,list2///),`
{}  (%dt,arith,([1])/%pt,arith,{,}%ft,list1//`
{}  %EXP1,%arith,<%EXP1,>,</>%ft,list2///)/
{}
{}--------------------------------------------------
```

The first thing "EXP" does is test to see if the fetch to "EXP" was neutral or not. In both cases it defines text arith as what ever was after "EXP" then partitions out of arith the curly braces and all list1 symbols. After that it fetches which ever evaluator it's supposed to depending on the neutral implied, substituting arith with commas in between all numbers and list2 replacement functions. The procedure that does all the evaluating is either "EXP1" or "EXP11". "EXP11" evaluates from right to left and to see what it does we will do a view text on it:

```
{}--------------------------------------------------
{}  %vt,EXP11/=
{}  %ii,[2],,[1],(%[2],[1],%EXP11[#3]//)/
{}
{}--------------------------------------------------
```

This is very similar to the recursive algorithms except that the numbers and functions are user defined in each case. "EXP1" then evaluates from left to right. A view text on it gives:

```
{}--------------------------------------------------
{}  %vt,EXP1/=
{}  %ii,[2],,[1],(%EXP1,%[2],[1],[3]/[#4]/)/
{}
{}--------------------------------------------------
```

This time it takes the first set of numbers and function and evaluates it then loops back on itself until only a number is left. If you want to use an user defined functions make sure you follow the same format as always. (numberFUNCTIONnumber) If the function only requires one number it must be enclosed in curly braces because no other form of hierarchy is programmed in.

```
|¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯|
||                ||   In this section I would like to just go to the
||   Examples     ||   teleprinter and show some examples using this
||_____||   evaluator. Remember everything that has been defined
                       in this section is still here and valid. My main
example will be simulating APL sort of. Anyway on with the show:
```

```
{}------------------------------------------------------------------
{}   %dt,FAC,!%ii,q2,,1,(%mu,q2,%FAC,,%su,q2,1///)/////////=
{}   %pt,FAC,,q2////=
{}   %define,^,exp/=
{}   %define,(@!),FAC/=
{}   %define,(@#),mrn/=
{}   %define,*,mu/=
{}   %define,-,su/=
{}   %dt,APL,!%ca,%xc,0D//%ut,**/%apl///=
{}   %dt,**,!%os,
{}   EXIT/%ri///=
{}   %dt,apl,!%os,
{}        /%os,
{}   %EXP,&IS///%apl///=
{}   %APL/=
{}        3+3
{} 6
{}        5+3-2*3\9
{} 2
{}        3#27
{} 3
{}        2^8
{} 256
{}        !100
{} 9332621 54439441 52681699238856266700490715 96982
{} 6438162146859296389521759999322991 56089414639
{} 761 565182862 53697920827223758251185210916 8640
{} 0000000000000000000000000
{}        !200
{} EXIT
{}
{}------------------------------------------------------------------
```

To make clear how I exited, because it is a little tricky, I defined a user trap that means that when a delete code was hit during a multiply function the text "**" was executed. With a little imagination one can get closer and closer to real APL but never all the way there because functions don't come before multiply and divide and so on. However let me see you simulate APL with a different kind of language, especially one that gets a factorial of one hundred with ALL significant figures.

The SAM76 language deals mostly with the manipulation of text. It is designed for use through a reactive machine such as a personal computer such as a "home reckoner" set.

The language design has the structure to allow interaction of functions resident in the machine with expressions, scripts or procedures written by the user; in this manner the language gives the user an unusual amount of flexibility and freedom for invention and extension.

The syntax consists first of a "warning character" followed by the expression itself then terminated by a different second "syntax marker"; in the following discussion the "warning characters used will be one of the following three: % - percent sign, & - ampersand or ! - exclamation mark; the "syntax marker" will be the / - slant sign for example:

%........./ or &........./ or else !........./

The foregoing three examples represent respectively the three types of expressions used in the SAM76 language and are known respectively as "active", "neutral" or "protected" expressions; the significance of the three types will be explained later.

The expression itself is made up of arguments which are separated by commas. The first argument designates the action to be taken. If this first argument consists of two or three alphabetic characters, the action to be taken may well be one defined as a function built in to the language or otherwise a language primitive function. Each argument following contains text or data to be dealt with by the action taken within the execution of the expression.

For instance we wish to add two and four; consequently we type everything in the following example up to and including the "=" equal sign which tells the computer to do its thing:

```
{}----------------
{}   %ad,2,4/=6
{}
   --------------------
```

The two letter code "ad" signifies the primitive of addition. Upon execution, which was initiated by the equal sign after the slant sign, the value of the second argument or 2 was added to the value of the third or 4. Then the value computed is outputted. The system then returns to a waiting condition known as the idling program which identifies itself by moving the "cursor" or printer to the beginning of the next line.

The idling program is actually the following expression:

%os,%is//

When starting, the innermost expression is located which contains an "is" primitive; "is" - or "input string" accepts input from the keyboard up to the reception of the current "activator" namely (in our case) the equal sign. The computer replaces the %is/ expression with this typed in text. Now the system goes back one level of nesting to the expression whose command was "os" or "output string"; this expression outputs the contents of the second argument which is now the text accepted from the keyboard, thus repeating what was typed in. For example:

```
{}------------------------------
{}   %os,%is//=ABC=ABC
{}
   ------------------------------
```

In actual fact the expression "%os,%is//=" is executed everytime the idling
program is loaded; it is not printed out and lives in what is known as the
working area of the memory so actually the printed example should be:

```
{}-----------------
{}  ABC=ABC
{}
  ------------------
```

It is important to be able to store text, script or procedures in memory. To
this end the "dt" which is the mnemonic for the "define text" function is
used. If we wish to define a text to be named "A" containing the words "AN
APPLE" we type:

```
{}---------------------------
{}  %dt,A,AN APPLE/=
{}
  ---------------------------
```

Now stored in memory is a "text" named "A" containing the words "AN APPLE".
To retrieve this information we "fetch" the "text" named "A", and in this
processs the second argument of the idling program will contain the words
stored and the "os" will output the value returned in the fetching of "A"
thusly:

```
{}---------------------------
{}  %ft,A/=AN APPLE
{}
  ---------------------------
```

When we defined the text "A" nothing was returned since "dt" does not return
any value on execution.

To Continue - "pt" or "partition text" removes one or more characters from a
string and in its place sets markers which represent the value of the
partition.

```
{}-------------------
{}  %pt,A,AN/=
{}
  -------------------
```

The second argument holds the name of the text to be dealt with; the third
argument is the string of characters which if found in the "text" will be
removed and replaced by partitions. Now to examine "A":

```
{}-------------------
{}  %ft,A/= APPLE
{}
  -------------------
```

Note that "AN" is missing and nothing shows its presence because the
expression that fetched "A" above did not require any partitions to be
"plugged" in a manner to be shown later.

We will now define another text to be named "B":

```
{}---------------------------------------
{}  %dt,B,THE SHACK ON THE HILL/=
{}
  ---------------------------------------
```

We partition that text on space:

```
{}-------------------
{}  %pt,B, /=
{}
  -------------------
```

We fetch "B" and get:

```
{}----------------------------------
{}   %ft,B/=THESHACKONTHEHILL
{}
   ----------------------------------
```

Notice the spaces are omitted.

"fe" or "fetch element" returns the contents of the text designated by the second argument; but on finding a partition it stops output. On the execution of the next "fe" on that text the next element of the text between partitions are returned:

The first:

```
{}-------------------
{}   %fe,B/=THE
{}
   -------------------
```

The second:

```
{}-------------------
{}   %fe,B/=SHACK
{}
   -------------------
```

The third:

```
{}----------------
{}   %fe,B/=ON
{}
   ----------------
```

It is very simple to find out where the partitions and the divider happen to be at any time by using the "vt" or "view text" primitive thus:

```
{}-----------------------------------------------------
{}   %vt,B/=THE[1]SHACK[1]ON[1][|]THE[1]HILL
{}
   -----------------------------------------------------
```

In this view of text "B" the partitions, all of value "1" are shown as [1], and the location of the text divider is shown by [|].

At the end of the "text" there is nothing left to return:

```
{}----------------
{}   %fe,B/=
{}
   -------------------
```

The gadget which remembers where one left off in the "text" is known as the "text divider"; each text has one of its own. This divider may be moved around by the execution of a a number of different primitives or may be ordered around through the use of the "md" - "move divider" function thus:

```
{}----------------
{}   %md,B/=
{}
   ----------------
```

will return the divider to the beginning or left end of the "text" named "B".

Now to explain how to replace the partitions in a text with characters; to do this we add arguments to the expression that is used to fetch the text. For instance if we wish to fetch "B" replacing the partitions with an asterisk:

```
{}----------------------------------------------
{}   %ft,B,*/=THE*SHACK*ON*THE*HILL
{}
    --------------------------------------------
```

Now we can redefine "B" as this value retuned and in other words return it to the original:

```
{}--------------------------
{}   %dt,B,%ft,B, //=
{}
    --------------------------
```

If we now fetch "B" it would seem that the original has never been changed:

```
{}------------------------------------------
{}   %ft,B/=THE SHACK ON THE HILL
{}
    ------------------------------------------
```

Since the SAM76 language works from the inside to the outside of the expression, it first fetched "B" replacing the partitions with spaces; then on doing the next expression a "text" named "B" was defined (erasing the original and partitioned out version). This usage of interactive functions, primitives within primitives is called nesting. In theory this can be done to any depth - in other words it is limited only by the amount of memory available.

Another example of nesting:

```
{}------------------------------------
{}   %dt,C,H/=
{}   %pt,B,%ft,C//=
{}   %ft,B/=TE SACK ON TE ILL
{}
    ------------------------------------
```

In the above example the text named "B" was partitioned on the basis of the characters received on fetching the text named "C". To return it to the original form we type:

```
{}----------------------------
{}   %dt,B,%ft,B,%ft,C///=
{}
    ----------------------------
```

In the latter reconstitution, text "C" was first fetched then this in the act of fetching "B" was used to replace partitions found therein; the result was then the argument of the define text expression.

At this time we will introduce a short cut in the act of "fetching". If the name of the text to be fetched is not the same as any of the primitives or built in functions then the first argument "ft" may be left out and the name of the text is used as the first argument of the expression. As we are using one character names for all our examples we can do this quite safely from now on.

In order to find out what the primitives in a system are you can do this by
executing the "@f" - "what function" command thus:

```
{}------------------------------------------------
{}  &@@f, /={function list will be here}
{}_____
```

Observe the use of the & - ampersand instead of the % sign as a warning
character to start the expression; also since @ is in its own right a
warning character, it is protected by preceding it with a second @, and the
space after the comma is used to tell the function what you wish to use to
separate the individual function mnemonics from each other.

The SAM76 language provides the ability of executing text strings and have
the functions or expressions in that string executed. This is done by
enclosing these executable expressions within the bounds of a "protected
expression" thus inhibiting execution at the time of definition. These
protected expressions are also called procedures or scripts.

```
{}--------------------------------------
{}  %dt,D,!%pt,B,%C////=
{}  %D/=
{}  %B/=TE SACK ON TE ILL
{}_____
```

The fetching of "D" caused the execution of the procedure stored therein
which in turn said - partition text "B" on the contents of "C".

Next we can define a text that will restore "B" to its original state:

```
{}--------------------------------------
{}  %dt,E,!%dt,B,%B,%C/////=
{}  %E/=
{}  %B/=THE SHACK ON THE HILL
{}_____
```

The part of the expression to be executed is enclosed between an ! -
exclamation mark and a / - slant sign showing an executable procedure. It is
easy to go from here and let the expression call another expression or
itself by simply fetching the text it is contained in. This ability of
recursion lets individual strings act as "subroutine" expressions.

There are two other ways of protecting procedures, besides using the !...../
form; these are by using (.....) or <.....> ; in this manner you can
incorporate ! and / in your text without having them act as if they were
warning characters.

Let us now say that we wish to be able to fetch one string and have it
partition out of "B" the contents of "C"; then output the contents of "B"
and then restore "B" to its original form. To do this we need to use the
"os" or "output string" primitive. This primitive outputs the contents of
the second argument of its expression:

```
{}------------------
{}  %os,ABC/=ABC
{}_____
```

Now if we nest the expression that fetches "B" within the "os" expression,
the contents of "B" will be displayed on execution:

```
{}----------------------------------------
{}  %os,%B//=THE SHACK ON THE HILL
{}_____
```

We will now use "os" in an executable expresssion to display the contents of
"B":

```
{}----------------------------------------
{}   %dt,F,!%D/%os,%B//%E///=
{}   %F/=TE SACK ON TE HILL
{}
     ----------------------------------------
```

First on execution of "F", "D" was fetched. Execution of "D" caused the
partitioning of "B" on the contents of "C". The execution of "os" displayed
"B" as it stood with its partitions empty or "null". Then "E" was fetched
and in its execution "E" caused the redefinition of "B" replacing the
partitions with the contents of "C" thus restoring it back to its original
condition.

Finally we would like to know just what we have created and stored in the
"text area" of memory. To do this we use the "lt" or "list text" primitive;
the second argument represents the character string we wish to use to
precede each name just so we can tell them apart from each other thus:

```
{}--------------------------------
{} %lt, /= A C D E F B
{}
     --------------------------------
```

In this example we used a space which precedes each name; note that "B" is
last in the list - that is because it was redefined for the last time when
we fetched "F" in the previous example.

	The editor of this beginner's description of the SAM76
Nota Bene	language wishes to credit Robert M. Evans, from whose
	first technical writing effort this was derived.

```
                        ALGOL-M
                 AN IMPLEMENTATION OF A
           HIGH-LEVEL BLOCK STRUCTURED LANGUAGE
         FOR A MICROPROCESSOR-BASED COMPUTER SYSTEM
```

LT Mark S. Moranville
Naval Postgraduate School
Code 52MI
Monterey, CA 93940
PH 408-646-2449

Abstract

The design and implementation of the ALGOL-M programming language for use on a microprocessor-based system is described. The implementation is comprised of two subsystems, a compiler which generates code for a hypothetical zero-address machine and a run-time monitor which executes this code. The system was implemented in PL/M to run on an 8080 microcomputer in a diskette-based environment with at least 20k bytes of user storage.

History of Algol

The definition of the algorithmic language (ALGOL-60) was the result of the work of a committee of distinguished computer scientists and was originally published in 1960 [2]. The purpose of the developers of ALGOL-60 was the establishment of a universal computer language specifically designed to allow for the logical and efficient program representation of algorithms. Additional versions and extensions of ALGOL-60 such as ALGOL-68 [5] and ALGOL-W [6] have been developed and have found acceptance in the academic communities and in Europe.

Microcomputer Software

The rapid development of microcomputer hardware since 1975 has generally resulted in a considerable lag in the corresponding development of compatible software, particularly that of high level languages. The Intel 8080 microprocessor is one of the few microprocessors which has endured long enough to permit software development to advance beyond the assembly language level. The majority of high level languages currently available for microcomputer based systems are extensions of the original Dartmouth BASIC and although they allow for a reasonable level of programming sophistication, they are encumbered by the inherent limitations of the BASIC language constructs.

Objectives of ALGOL-M

The major objective of this project was to develop a dynamic, block-structured, recursive high level language which would provide adequate programming power and flexibility for applications programming using microcomputer based systems. ALGOL constructs were chosen because of their simplicity and power and because it was possible to write the grammar in LALR(1) form for use with available compiler-compiler generated parse tables [4]. ALGOL-M was developed to run on 8080 based microcomputer systems because of the availability of a high level system development language (PL/M) [1] which produces 8080 object code and which could be run on the Naval Postgraduate School's IBM 360. The availability of an 8080 based disk operating system (CP/M) [3] simulator on the IBM 360 and the widespread use of CP/M were also strong factors in the choice of 8080 microprocessor and CP/M operating system.

Features of the ALGOL-M Language

Although ALGOL-M was modeled after ALGOL-60 no attempt was made to make it a formal subset of ALGOL-60. This was done intentionally in order to provide a language which would be best suited to the needs of applications programmers using microcomputer systems. However, the basic structure of ALGOL-M is similar enough to ALGOL-60 to allow simple conversion of programs from one language to the other. This was considered particularly important in view of the fact that the standard publication language is ALGOL-60. Therefore, there exists a large source of applications programs and library procedures which can be simply converted to execute under ALGOL-M.

Type Declarations. ALGOL-M supports three types of variables: integers, decimals, and strings. Integers may be any value between -32,767 and +32,767. Decimals may be declared ith up to 18 digits of precision and strings ᵣ be declared as long as 255 characters. The ᵢault precision for decimals is ten digits ᵢ d the default length for strings is ten characters. Decimal and string variable lengths may be integer variables which can be assigned actual values at run-time.

Another form of declaration in ALGOL-M is the array declaration. Arrays may have up to 255 dimensions with each dimension ranging from

-32,767 to +32,767. The maximum 8080 micro-processor address space of 64k bytes limits practical array sizes to something smaller than the maximum. Dimension bounds may be integer variables with the actual values assigned at run-time. Arrays may be of type integer, decimal or string.

Arithmetic Processing. Integer and binary coded decimal arithmetic are supported under ALGOL-M. Integers may be used in decimal expressions and will be converted to decimals by the compiler. The integer and decimal comparisions of less-than, greater-than, equal-to, not-equal-to, less-than-or-equal-to, and greater-than-or-equal-to are provided. Additionally, the logical operators AND, OR and NOT are available.

Control Structures. ALGOL-M control structures consist of BEGIN, END, FOR, IF THEN, IF THEN ELSE, WHILE, CASE, and GOTO constructs. Function and procedure calls are also used as control structures. ALGOL-M is a block structured language with a block normally bracketed by a BEGIN and an END. Blocks may be nested within other blocks to nine levels. Variables which are declared within a block can only be referenced within that block or a block nested within that block. Once program control proceeds outside of a block in which a variable has been declared, the variable may not be referenced and, in fact, run-time storage space for that variable no longer exists.

Functions, when called, return an integer, decimal or string value depending on the type of the function. Procedures do not return a value when called. Both functions and procedures may have zero or more parameters and may be called recursively.

Input/Output. The ALGOL-M WRITE statement causes output to the console on a new line. The desired output is specified in a write list which is enclosed in parentheses. String constants may be used in a write list and are characterized by being enclosed in quotation marks. Any combination of integer, decimal and string variables or expressions may also be used in a write list. A WRITEON statement is also available which is essentially the same as the WRITE statement except that output continues on the same line as the output from a previous WRITE or WRITEON statement. When a total of 80 characters have been written to the console a new line is started automatically.

Console input is accomplished by the READ statement followed by a read list of any combination of integer, decimal and string variables enclosed in parentheses. If embedded blanks are desired in the input for a string variable, the console input must be enclosed in quotation marks. A READ statement will result

in a halt in program execution at run-time until the imput values are typed at the console and a carriage return is sent. If the values typed at the console match the read list in number and type, program execution continues. If an error as to number or type of variables from the console occurs, program execution is again halted until values are re-entered on the console.

Implementation

The implementation of ALGOL-M consists of of two subsystems, a compiler and an interpreter

Compiler Implementation. The compiler was designed to read source language statements from a diskette and to produce an intermediate language file with optional source listing at the console. A two pass approach was used to facilitate the implementation of GOTO statements forward subroutines, and control statements. Pass one builds the symbol table and saves all branch locations for resolution during pass two. Pass one also computes the size of the program reference table(which is used at run-time for maintaining the actual memory locations of variables, arrays, and subroutines) and writes this information out to the intermediate file. Pass two resolves all forward references and emits code to the intermediate file on disk.

Interpreter Implementation. The ALGOL-M pseudo machine, as shown in Figure 1, is a software simulation of a stack-oriented CPU with an instruction set which is particularly well suited for execution of ALGOL-M programs. The ALGOL-M interpreter is loaded at address 100 hex (as are all executable programs under the CP/M operating system) and proceeds to read the ALGOL-M intermediate code from disk, constructing the pseudo machine begining at the first free memory location. The ALGOL-M intermediate code is read into a buffer in 128 byte segments. The first two bytes of the intermediate code represent an integer value equal to the number of bytes to be used for the program reference table (PRT).

The remaining intermediate code is manipulated in accordance with the algorithm shown in Figure 2 in order to construct the pseudo machine code area.

The ALGOL-M interpreter uses the pseudo machine code area as input data. Each pseudo machine operator is equated to an integer value which is evaluated in order to provide the correct entry into a large case statement in the interpreter. Each entry in the case statement contains the necessary code to cause proper run-time execution of the specific ALGOL-M pseudo instruction. The case statement is executed continually until the ALGOL-M program has been completed, at which time control is passed back to the operating system. A run-time

```
RA ──────┐
         │  ┌─────────┬─────────┐
         └──┤         │         │
RB ──────┐  ├─────────┼─────────┤      STACK
         └──┤         │         │
            ├─────────┴─────────┤
            │                   │
            │                   │
            │                   │
            │                   │      CODE  AREA
            │                   │
            │                   │
RC ──────┐  │                   │
         └──├─────────┬─────────┤
0004 ───────┤         │         │
            ├─────────┼─────────┤      PROGRAM REFERENCE
0002 ───────┤         │         │      TABLE  (PRT)
            ├─────────┼─────────┤
0000 ───────┤         │         │
            ├─────────┴─────────┤
            │                   │      INTERPRETER
            │                   │      PROGRAM
100H ───────┤                   │
            └───────────────────┘
```
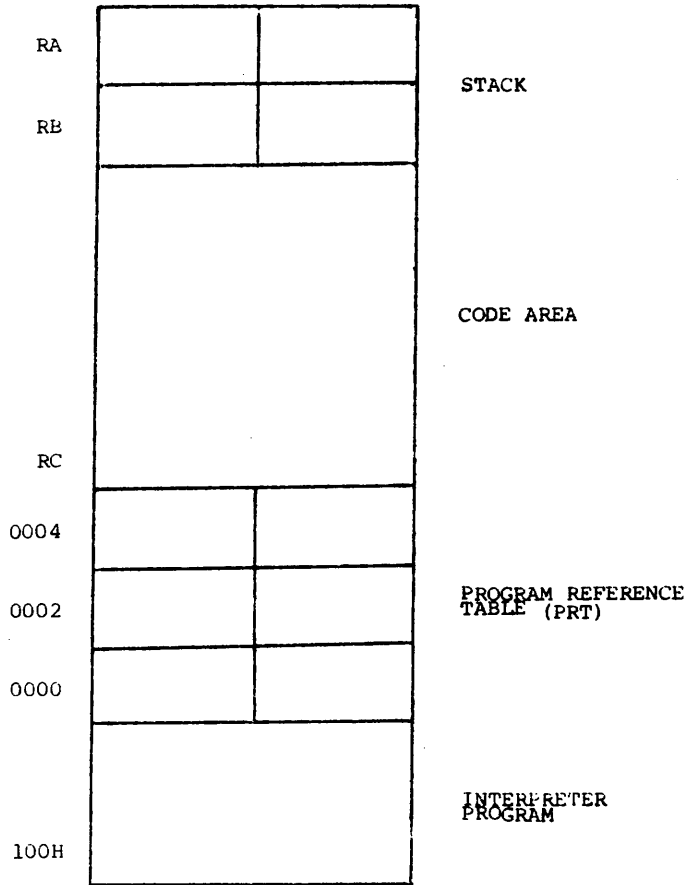
ALGOL-M MACHINE
MEMORY ORGANIZATION

FIGURE 1
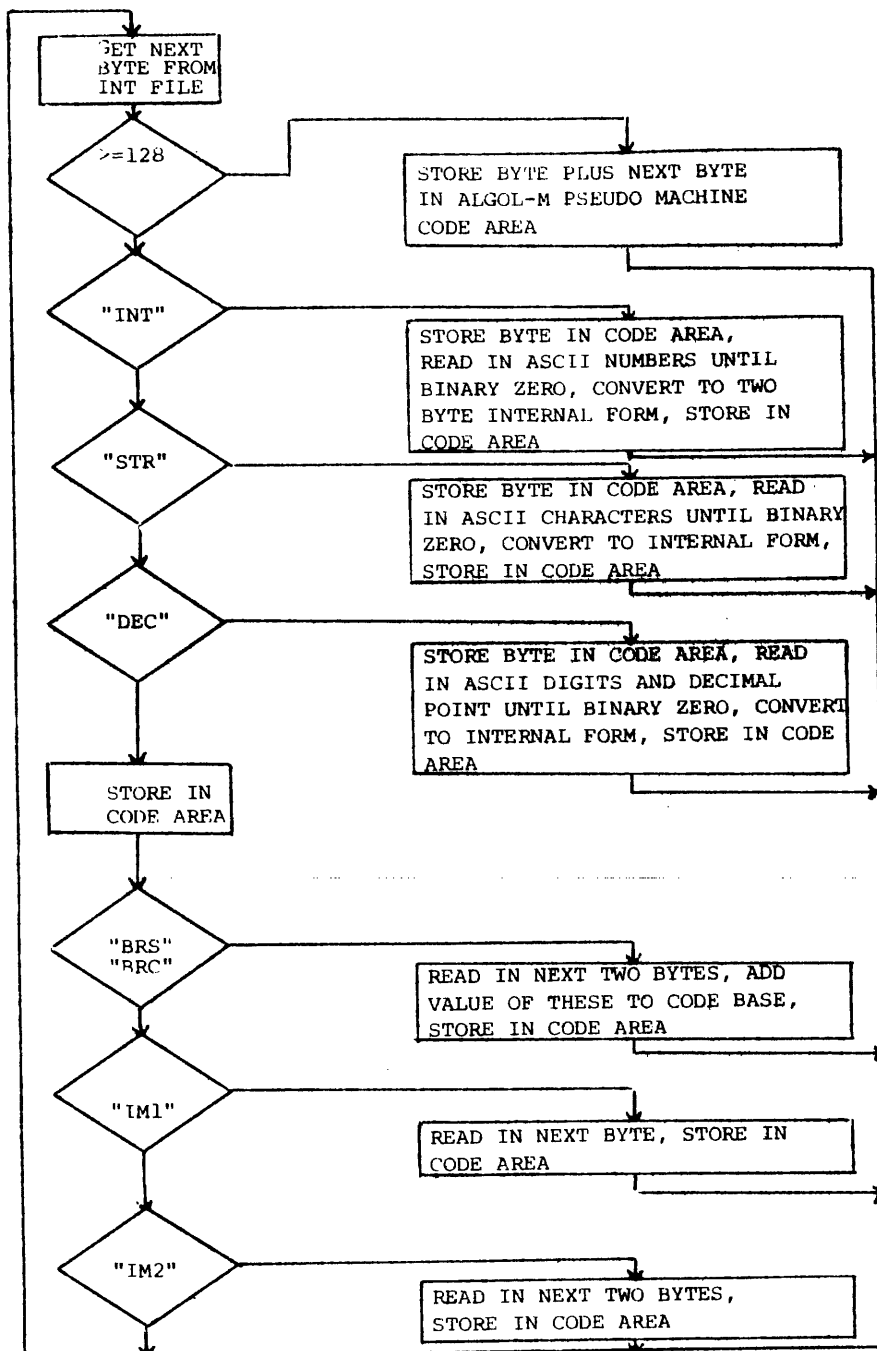
FIGURE 2

stack is used to facilitate the execution of ALGOL-M pseudo instructions and for run-time storage allocation. Figure 3 illustrates how the stack is used in allocating decimal and string variables(integers are stored directly in the PRT). Each time a new block is entered the block level pointer is set to the top of the stack. When a block is departed the top of stack pointer(RA) is decremented to the address of the previous block level. Therefore the storage allocated for variables within a given block is automatically de-allocated when the block is departed.

Conclusions

This project has resulted in the construction of a high-level, block-structured, applications oriented compiler for micro computers with 20k bytes of memory or more. When compared to a fully dynamic scheme, the stack storage allocation and retrieval scheme presented here appears to enhance program execution speed, reduce memory requirements, and simplify compiler implementation. Included in appendix A are benchmark programs 4, 5, 6, and 7 obtained from REF [7]. These benchmark programs illustrate the structure of ALGOL-M as compared to BASIC.

Acknowledgements

ALGOL-M is an outgrowth of a thesis completed at the Naval Postgraduate School by the author and LCDR John P. Flynn. LCDR Flynn's contribution to the completion of this project cannot be over emphasized.
Professor Gary Kildall was our thesis advisor and I wish to express my appreciation for his assistance and direction throughout this project.
Thanks also goes to the W.R. Church Computer Center Staff for their technical support.

List of References

1. Intel Corporation, 8008 and 8080 PL/M programming Manual, 1975, 3065 Bowers Ave, Santa Clara, CA., 95051.

2. Naur, P. (ed.), "Report on the Algorithmic Language ALGOL-60", Comm. ACM, Vol. 3, No. 5, May 1960, pp.299-314.

3. Strukynski, Kathryn B. Information on the CP/M Interface Simulator, internally distributed technical note.

4. University of Toronto, Computer Systems Research Group Technical Report CSRG-2, " an Efficient LALR Parser Generator," by W.R. Lalonge, April 1971.

5. van Wijngaarden, A., B. J. Mallioux, J. E. L.,

Peck, C.H.A. Koster, "Report on the Algorithmic Language ALGOL-68", Numer. Math. Vol 14, 1969.

6. Wirth, N., C. A. R. Hoare, "A contribution to the Developments of ALGOL", CASM, Vol.9, No. 5, June 1966, pp. 413-432.

7. Feldman, P. and Rugg, T., "BASIC Timing Comparisons", Kilobaud, Issue #6, p. 66-69, June 1977.

```
BLOCK LEVEL 0        BEGIN
                         DECIMAL A,B,C;
BLOCK LEVEL 1            BEGIN
                            STRING X,Y,Z;
BLOCK LEVEL 2               BEGIN
                               DECIMAL C,D,E;
END BLOCK LEVEL 2           END;
END BLOCK LEVEL 1        END;
END BLOCK LEVEL 0    END;
```

```
2
1
0
LEVEL  ADDRESS
```

STACK

RA

RB

STORAGE FOR
BLOCK LEVEL 2

STORAGE FOR
BLOCK LEVEL 1

STORAGE FOR
BLOCK LEVEL 0
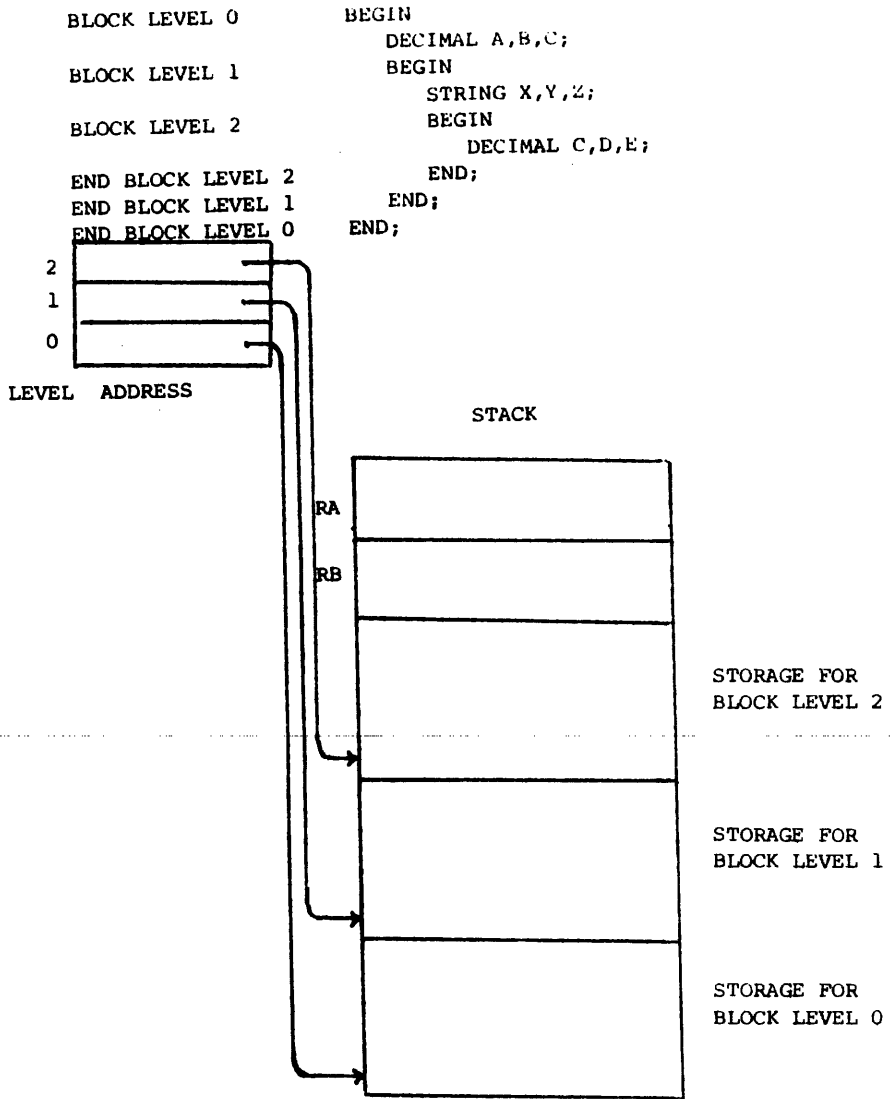
BLOCK LEVELS AND STACK STORAGE

FIGURE 3

Benchmark Program 4

```
300 PRINT "START"                    BEGIN
400 K=0                              INTEGER A,K;
500 K=K+1                            WRITE("START");
510 A=K/2*3+4-5                      K:=0;
600 IF K<1000 THEN 500               WHILE K<1000 DO
700 PRINT "END"                        BEGIN
800 END                                  K:=K+1;
                                         A:=K/2*3+4-5;
                                       END;
                                     WRITE("END");
                                     END
```

Benchmark Program 5

```
300 PRINT"START"                     BEGIN
400 K=0                              INTEGER A,K;
500 K=K+1                            PROCEDURE DONOTHING;
510 A=K/2*3+4-5                        A:=0;
520 GOSUB 820                        WRITE("START");
600 IF K<1000 THEN 500               K:=0;
700 PRINT "END"                      WHILE K<1000 DO
800 END                                BEGIN
                                         K:=K+1;
                                         A:=K/2*3+4-5;
                                         DONOTHING;
                                       END;
                                     WRITE("END");
                                     END
```

Benchmark Program 6

```
300 PRINT"START"            BEGIN
400 K=0                     INTEGER A,K,L;
430 DIM M(5)                INTEGER ARRAY M[1:5];
500 K=K+1;                  PROCEDURE DONOTHING;
510 A=K/2*3+4-5              BEGIN
520 GOSUB 820                A:=0;
530 FOR L-1 TO 5             END;
540 NEXT L                  WRITE("START");
600 IF K<1000 THEN 500          K:=0;
700 PRINT"END"              WHILE K<1000 DO
800 END                      BEGIN
                              K:=K+1;
                              A:=K/2*3+4-5;
                              DONOTHING;
                              FOR L:=1 STEP 1 UNTIL 5 DO
                               A:=0;
                             END;
                            WRITE("END");
                            END
```

Benchmark Program 7

```
300 PRINT "START"           BEGIN
400 K=0                     INTEGER A,K,L;
430 DIM M(5)                INTEGER ARRAY M[1:5];
500 K=K+1                   PROCEDURE DONOTHING;
510 A=K/2*3+4-5              BEGIN
520 GOSUB 820                A:=0;
530 FOR L=1 TO 5             END;
535 M(L)=A                  WRITE("START");
540 NEXT L                  K:=0;
600 IF K<1000 THEN 500      WHILE K<1000 DO
700 PRINT"END"              BEGIN
800 END                      K:=K+1;
820 RETURN                   A:=K/2*3+4-5;
                             DONOTHING;
                             FOR L:=1 STEP 1 UNTIL 5 DO
                               M[L]:=A;
                             END;
                            WRITE("END");
                            END
```

SPL/M - A CASSETTE-BASED COMPILER

Thomas W. Crosley
1675 New Brunswick Ave.
Sunnyvale, CA 94087

Abstract

SPL/M is a subset of the PL/M language suitable for systems programming on small computers. The subset was necessary in order to fit the one-pass compiler into the author's system, which has 20K of RAM and two cassette decks.

The paper describes how the subset was selected, and includes the BNF for SPL/M. The implementation is also described, with an emphasis on the code generation and optimization.

Introduction

My hobbiest software interests lie mostly in the systems programming area. Therefore shortly after I had my 6800-based system up and running, and had written an assembler in machine language, I was already looking around for a suitable high-level language to use instead of assembly code. In particular, I wanted a procedure-oriented systems programming language that had:

1. Arbitrary length identifiers
2. Structured programming constructs
3. Block structure (local variables)
4. Arithmetic & logical operators
5. Pointer variables
6. String and character manipulation
7. Direct access to user memory
8. Recursion capabilities
9. Easy linkage to assembly language

The language also had to be compilable in one-pass, since the only secondary storage on my system consists of two cassette decks.

Rather than try to design yet another high-level language, I chose one I already knew, namely PL/M. PL/M was specifically designed for implementation on 8-bit micro-processors, and also fully met all of the conditions stated above except (6) and (8). PL/M has no built-in string functions but one can write user functions instead. Recursive procedure calls are possible; however automatic stacking of parameters is not provided.

PL/M was originally designed as a cross-compiler for the Intel 8008 microprocessor and was later made available for the 8080 [1]. However I am using as my primary language reference the implementation by Intermetrics of PL/M6800 [2], also a cross-compiler. Intel has since come out with a resident version of PL/M-80 for the 8080, but it is not compatible with their cross-compiler version. (It also requires 64K and dual floppys.)

Description of SPL/M

I had decided to write the compiler in a subset of PL/M, and then hand-translate it to assembly language. Once the subset was up and running the compiler would be able to compile itself. I would then be able to incrementally add features to the compiler until the full language was implemented.

The initial subset included only those features necessary to write the compiler. For example, the only arithmetic operators available were addition and subtraction.

After I had the initial version of the compiler running, it was large enough that it was obvious a full PL/M version would not fit on my system. Therefore I decided to continue to expand the original compiler (by adding additional PL/M features in assembly code) until I had a version of the language that met my needs.

The resulting subset is called SPL/M (for Small PL/M). The features of SPL/M (Version 1.0) are listed in Figure A-1 at the end of the paper, while the full grammar of the subset is shown in Figure A-2.

A detailed list of the the PL/M features that are not in SPL/M is given in Figure A-3, along with the reason for leaving them out. In general for each feature that was

removed there is an alternate construction available in SPL/M that will do the same thing.

Some of the features that were removed are expected to be added back in for the next version of the compiler, as indicated in the comments column of Figure A-3. However I do not plan on putting GOTO's back in; think of this as promoting structured programming. I did however add a BREAK statement to SPL/M (borrowed from the language C) for abnormal loop termination.

Another extension to PL/M are the MEM and MEMA arrays. They are predeclared to start at memory address 0, and allow direct access to memory like the POKE and PEEK functions in some BASIC's. MEM and MEMA are also used to simulate based variables which are not included in V1.0 of SPL/M.

MEM is type byte, while MEMA is type address. The normal doubling of subscripts for address variables is not done for MEMA; for example,

$$MEMA(38H) = 0F050H;$$

sets memory locations 38 and 39 to the hexadecimal value F050.

The last extension added was to allow the construction CALL <number> in addition to CALL <identifier>. This makes it easier to call assembly language procedures. Typically the <number> is substituted at compile time via a DECLARE LITERALLY, so that the source code is still symbolic.

Except for the above three extensions, SPL/M is syntatically a proper subset of PL/M. The major semantic difference is in the treatment of mixed mode (byte/address) statements. In PL/M, the result of an arithmetic operation is based only on its immediate operand(s). Therefore, assuming X is an address variable, the PL/M statement:

$$X = -1;$$

would set X equal to OFFH, rather than the expected OFFFFH. This is because the unary operation −1 is the same as 0−1, which involves two byte operands.

In SPL/M, I have chosen to make the result of an arithmetic operation equal to the highest precision encountered so far in the statement; therefore in the above example X would be set to OFFFFH in SPL/M since the destination is type address.

Implementation

Because all variables and procedures must be defined before they are referenced in SPL/M, the language can be compiled in a single pass over the source code. One-pass compilers have a few special problems, namely in code generation (discussed later). However the overall organization of the compiler (scanner, parser, code generator, and code optimizer) is the same as most other compilers. There is just more happening in parallel.

Scanner. The lexical scanner reads the source characters out of an internal buffer, and from these constructs the source program "tokens" (such as identifiers, reserved words, integers, and one or two character symbols). Each token is represented internally by a token index value.

The scanner is implemented as a subroutine which is called by the parser. When called, the scanner recognizes the next source program symbol and passes the token index up to the parser.

The scanner knows very little about SPL/M, leaving that to the parser. However it does resolve some ambiguities for the parser, such as the difference between a variable at the beginning of an assignment statement and a label in front of a procedure declaration. The scanner also removes comments (delimited by a /* */ pair) which may appear anywhere a blank is allowed.

Symbol Table Routines. Since SPL/M is a block structured language, each block can have its own symbol table. Thus the overall symbol table is organized as a stack.

The head of each symbol table block consists of a fixed portion 55 bytes long. The first two bytes are a backwards link to the previous block. The next byte is the nesting level for the current block. The last 52 bytes are used as pointers into a dynamic area for the block. The pointers are indexed by the first letter of the symbol. Each entry in the dynamic area contains the following fields:

1. Two byte link to the next entry in the chain.
2. Number of characters in the symbol.
3. 2nd through nth character of the symbol.

4. Token index.
5. Address (only if a variable or procedure).

The token index indicates whether the symbol is a reserved word, byte or address variable, or a procedure.

Each time the scanner scans out an identifier, it first searches the first symbol table block to see if it is a reserved word. If not, then the most current block is searched. If the symbol is still not found, the backwards link field is used to search the next most current block. This is repeated as necessary until all the blocks have been searched.

For DECLARE statements, the above procedure is performed except that the search is stopped after only searching the reserved word block and the most current block.

Every time a DO or procedure block is entered, the current nesting level is incremented. However a new symbol table block is not allocated until a variable (if any) is declared within the block. At the end of the DO group or procedure, the current nesting level is compared with the nesting level for the current symbol table to see if it should be deallocated.

Parsing. The parsing technique used by the compiler is called recursive descent. It was chosen because the resulting parser closely follows the BNF of the grammar, and is therefore easier to debug and modify than bottom-up parsers.

For instance, the SPL/M procedure GROUP (Figure A-4) corresponds to the production <group> in the grammar. Since DO groups can be nested, it first stacks two of its variables which allows the procedure to be called recursively. It then calls the procedure GRP$HEAD which will parse out either a plain DO group or else a DO-WHILE. GRP$HEAD also calls a procedure to increment the nesting level.

On return from GRP$HEAD, if there have been no syntax errors so far (E=0) then zero or more statements are parsed out (procedure STMT) until an END is encountered. At the end of GROUP, a call is made to EXIT$BLK to decrement the nesting level and delete the symbol table block if one was allocated.

Error Handling. Errors are caught by both the parser and the scanner. When an error is detected, the source line is printed followed by a line containing one or more single letter flags indicating the error(s). For example, 'S' is used to indicate a syntax error, 'U' means an undefined identifier, and 'D' stands for a duplicate definition.

The flags are positioned under the token where the error was discovered. For example in the printout below,

```
0210    TBL(I) = CTR1 ++ CTR2 ;
****    U                S U
```

TBL and CTR2 are undefined, and there is a syntax error because of the second '+'. When a syntax error is discovered, the remainder of the statement is ignored (up to the next ';'), except that the scanner continues to flag undefined symbols. Also, when undefined symbols are encountered code is still generated (assuming an address of 0) to allow patching.

Generation of error messages was another factor in choosing recursive descent for parsing. Because top-down parsing is goal oriented, at any point the parser knows exactly what symbol (or symbols) should come next. This makes it easier to locate the 'S' flag under the offending token.

Code generation. Since the 6800 has a limited number of registers, code generation is not complicated by having to perform elaborate register allocation as it would be with the 8080 microprocessor.

Accumulator A is used for all byte expressions, with accumulator B used only for address operations. (In the latter case BA is treated as a 16 bit accumulator.) If there is a value in either accumulator that must be saved temporarily, it is pushed onto the run-time stack. Later the temporary is accessed via the index register. For example,

```
PSHA       (save temp value)
     .
(code which reloads ACCA)
     .
TSX
ANDA 0,X (X points to temp)
INS
```

The index register is primarily used however to address arrays. Subscripted addresses are built up in BA and then transfered to the X register via a subroutine. (The lack of a instruction to do this is probably the single greatest weakness in the 6800 instruction set.) The only other subroutine calls generated by the compiler are for multiplication and division. The compiler does not have to generate the the above three subroutines since they are all available in my system ROM.

Most compilers translate the entire source program into some internal form, such as a tree or a list of quadruples. Additional passes translate the internal form into the object code in addition to performing optimizations.

Since SPL/M is a one-pass compiler, this technique is not used. Instead code is generated as the parsing takes place. For example (again referring to Figure A-4), at address 1577 a jump (7EH) is generated back to the beginning of the the DO WHILE following the 'END'. (The address to jump to had been previously saved in GH$ADDR by procedure GRP$HEAD).

Code generation in a one-pass compiler is complicated by not knowing what is coming next in the source. In an IF or DO WHILE statement, a jump is required to go to the end of the statement if the <expr> evaluates false, but the end of the statement isn't known yet. So the compiler generates a jump to 0 instead (see address 1567), and later a "fixup" is performed to patch up the jump to the proper address. Code generated for fixups is listed in parentheses on the printout (e.g. see code following address 156D).

Initially I tried to handle code generation for expressions in the same manner; i.e. in the same routines that were parsing out the expression. However because of the hierarchy of operators I found it difficult to generate the proper code at the right time. I therefore separated the code generation for expressions into individual routines for each operator. Communication between the parser and the code generator is via two stacks: one for operands and the other for operators.

Entries on the operator stack are all one byte long, and consist of an integer representing the particular

operator. The operand stack has variable length entries from one to three bytes in length. Each entry describes the operand type and where it is currently located (at run-time):

Code	Length	Meaning
0	2	Byte constant
1	3	Address constant
2	1	Byte value in A
3	1	Address value in BA
4	1	Byte value on TOS
5	1	Address value on TOS
6	3	Byte variable
7	3	Address variable
8	1	Byte variable, addr in BA
9	1	Addr variable, addr in BA
A	1	Byte variable, addr on TOS
B	1	Addr variable, addr on TOS
C	1	Byte variable, addr in X
D	1	Addr variable, addr in X

A — Accumulator A
BA — Accumulators BA
TOS — Top of stack
X — Index register

As an example, the statement:

$$X = 500 + Y * 31H ;$$

is represented internally as:

Operand Stack	Operator Stack
0 (byte const)	
31	
7 (addr var)	
HIGH(.Y)	
LOW(.Y)	
0 (addr const)	
01	
F4	5 (code for *)
7 (addr var)	2 (code for +)
HIGH(.X)	B (code for assign =)
LOW(.X)	0 (beg of stmt)

where HIGH(.Y) and LOW(.Y) refer to the high and low bytes of the address of variable Y.

As each operand in an expression is parsed out, it is pushed onto the operand stack. However as each operator is parsed out, a pair of vectors (called F and G functions in Gries [3]) are used to see if the code for the operator already at the top of the operator stack should be generated first.

This is done by comparing the F value corresponding to the operator at

the top of the stack with the G value for the new operator just parsed. If the F value is greater, then a code generation routine is called for the operator at the top of the stack. It will remove one or two operands from the operand stack, delete the operator from the top of the operator stack, and push a descriptor of the result onto the operand stack.

If instead the F value is less then the G value, the new operator is just pushed onto the operator stack.

The values for the F and G vectors reflect the operator precedence for the language. I wrote a BASIC program to compute the vectors, based on an algorithm on page 116 of Gries. The input to the program consisted of strings representing the BNF for SPL/M expressions.

Code Optimization. Because of the limitations on the size of the compiler, optimizations are performed only on expressions within a single statement. In particular, no attempt is made to remember the value of any of the registers across statement boundaries.

Most of the optimizing is done be examining the operand and operator stacks for particular patterns after parsing out an entire expression. For example the statement

        I = 0 ;

is placed on the two stacks as:

Operand	Operator
0	
0	
6	
HIGH(.var)	B (assign=)
LOW(.var)	O

When the compiler recognizes this pattern, it generates a CLR memory instruction rather than a CLRA, STAA memory sequence.

The same technique is used to optimize statements such as

        I = I + 1 ;

into an INC memory instruction. However if the statement was written

        I = 1 + I ;

the pattern would not be recognized and no optimization would be performed. However the second form is much less likely to be written.

Further optimizations done in this way include using the index register for double precision loads and stores where appropriate.

Another type of optimization is performed for IF and DO WHILE statements. In SPL/M (as in PL/M), an expression is true if the rightmost bit of the result is a one. So the general code for an IF <expr> THEN statement is:

        (code for <expr>)
        LSRA
        BCS 3+*
        JMP (to end of IF)
        (code for true part)

However if the expression is a <logical primary> of the form:

        <arith expr> <rel> <arith expr>

the code generated would be rather inefficient:

        (code preceding REL test)
        Bxx 3+*   (conditional branch)
        CLRA   (false)
        BRA 2+*
        LDAA =0FFH   (true)
    * (end of code for <expr>)
        LSRA
        BCS 3+*
        JMP (to end of IF)
        (code for true part)

since the test ends up being performed twice. Therefore the sequence is optimized to just

        (code preceding REL test)
        Bxx 3+*
        JMP (to end of IF)
        (code for true part)

Refer to Figure A-4 for examples of the various optimizations discussed above.

Cassette I/O Routines. The compiler is designed to operate with two cassette decks — one in "play" mode, reading in source, and the other in "record", for writing out the object code.

To keep the RAM requirements to a minimum, the source program is read in as a series of blocked records into a 2K buffer, each block consisting of

(typically) 100 source lines. The scanner, on detecting the end of a buffer, calls the cassette read routine to read in the next block. This is continued until the last line of the program (containing "EOF") is read.

The object program is also divided into blocks; however a double buffer system is used, each buffer being 512 bytes long. This insures that the fixup routine used to patch up forward references always has at least 512 bytes to look back into. This is considered adequate since fixups are always done within a single procedure.

Each object block consists of a header, which includes the block name, length, type, and start address. An object block is written out whenever the current buffer overflows, or if a new origin is found. The compiler generates only absolute code.

System Considerations

The compiler is presently designed to run only on the Sphere 6800 system, Model 330. In particular, it assumes the existence of:

    CRT/1
    KBD/1 or 2
    Cass I and II
    Printer
    MEM/1 (20K RAM total)
    PDS V3N ROM set

The printer is the only device that will vary from one system to another. Therefore two user routines must be provided: one called at the beginning to initialize the printer ACIA, and a second to print a single character contained in accumulator A.

The compiler object is just over 8K in length. It should be possible to run the compiler in a 16K system, and still have around 2.5K available for the symbol table. That would be adequate for approximately 200 six-character symbols. The total number of symbols in a program could be much higher than that, since symbol table blocks are dynamic.

Compile Time Options. The compiler has several options relating to the input/output devices.

Source input normally comes from Cassette I, but as an option it can be instead input from the keyboard. This

is useful for debugging and demonstrations. Since SPL/M is a one-pass compiler, the code will be output following every statement.

By default, a full listing is output to the CRT only, with just an error printout going to the printer. However as an option the full listing (including generated object code) can be printed as shown in Figure A-4.

An additional option allows the symbol table to be printed out. Each symbol table block is dumped out (just before it is deallocated) in the same order it is stored in memory; therefore symbols are alphabetized on the first letter only. Along with each symbol is listed the type (BYTE, ADDR, PROC, or LIT), and its value.

Summary and Conclusions

My original goal was to write a compiler for PL/M that would fit in my system. I did not fully succeed in doing that, but I have implemented a compiler for a useful subset of the language. Even so that took over one and one-half years of part time effort.

I feel it was worth it, since writing systems software in SPL/M is much easier than using assembly language. However I find that the compiler, on the average, generates about twice as much code as I do when coding in assembly level.

The text formatter I am using to print this paper was the first major SPL/M program I wrote; it consists of approximately 800 lines of SPL/M and is just over 5K bytes long. While debugging the formatter I was able to trace only one bug to an error in the compiler.

My next major goal is to complete Version 2.0 of SPL/M, which I hope will be much closer to being a full PL/M.

References

1. 8008 and 8080 PL/M Programming Manual, Revision A. Intel Corporation, Santa Clara, CA, 1975.

2. PL/M6800 Language Specification. Intermetrics, Inc., Cambridge, MA, 1975.

3. Gries, D. Compiler Construction for Digital Computers. John Wiley & Sons, Inc., New York, NY, 1971.

Statement Types:

    DECLARE
    Assignment (e.g. COUNT = 1;)
    IF-THEN (with optional ELSE)
    DO-WHILE (loop control)
    Grouping (DO; statement list END;)
    PROCEDURE definition

Declarations:

    Define variable types, either BYTE (8 bits), or
        ADDRESS (16 bits)
    Define arrays (one dimension only), either
        variable or constant (DATA)
    Define compile time numeric substitutions (LITERALLY)

Operators:

    Arithmetic:
        + - * / MOD (modulo)
    Logical:
        NOT AND OR XOR
    Relational:
        < > = <= >= <> (not equal)

Procedures:

    Defined within a PROCEDURE-END pair and called via a
        CALL statement

Built-in functions:

    Type conversion (LOW, HIGH, DOUBLE)
    Direct memory access (MEM, MEMA)

Miscellaneous:

    Identifiers may be any length
    Decimal, hexadecimal, and string constants
    Integer arithmetic only

Figure A-1.   SPL/M Language Features (V1.0)

```
<program> ::= <stmt list> EOF

<stmt list> ::= <stmt> ¦ <stmt list> <stmt> ¦ NIL

<stmt> ::= <basic stmt> ¦ <if stmt>

<basic stmt> ::= <assignment> ;
              ¦ <group> ;
              ¦ <proc def> ;
              ¦ <call stmt> ;
              ¦ RETURN ;
              ¦ BREAK ;
              ¦ <decl stmt> ;

<if stmt> ::= <if clause> <stmt>
           ¦ <if clause> <basic stmt> ELSE <stmt>

<if clause> ::= IF <expr> THEN

<group> ::= <group head> <stmt list> END

<group head> ::= DO ;
              ¦ DO WHILE <expr> ;

<proc def> ::= <proc head> <stmt list> END

<proc head> ::= <identifier>: PROCEDURE ;
             ¦ <identifier>: PROC ;
             ¦ <origin> <proc head>

<call stmt> ::= CALL <identifier> ¦ CALL <number>

<decl stmt> ::= DECLARE <decl element>
             ¦ DCL <decl element>
             ¦ <decl stmt> , <decl element>
             ¦ <origin> <decl stmt>

<decl element> ::= <identifier> <type>
                ¦ <identifier> ( <number> ) <type>
                ¦ <identifier> <data list>
                ¦ <identifier> LITERALLY '<number>'
                < <identifier> LIT '<number>'

<type> ::= BYTE ¦ ADDRESS ¦ ADDR

<data list> ::= <data head> <constant> )

<data head> ::= DATA (
             ¦ <data head> <constant> ,

<origin> ::= <number>:

<assignment> ::= <variable> = <expr>
```

Figure A-2. SPL/M Grammar

```
<expr> ::= <logical factor>
         | <expr> OR <logical factor>
         | <expr> XOR <logical factor>

<logical factor> ::= <logical secondary>
         | <logical factor> AND <logical secondary>

<logical secondary> ::= <logical primary>
         | NOT <logical primary>

<logical primary> ::= <arith expr>
         | <arith expr> <relation> <arith expr>

<relation> ::= = | < | > | <> | <= | >=

<arith expr> ::= <term>
         | <arith expr> + <term>
         | <arith expr> - <term>

<term> ::= <secondary>
         | <term> * <secondary>
         | <term> / <secondary>
         | <term> MOD <secondary>

<secondary> ::= <primary>
         | - <primary>

<primary> ::= <constant>
         | <variable>
         | .<identifier>
         | ( <expr> )

<variable> ::= <identifier>
         | <identifier> ( <expr> )

<constant> ::= <number> | '<string>'

<identifier> ::= <letter>
         | <identifier> <dec digit>
         | <identifier> <letter>
         | <identifier> $

<letter> ::= A | B | C ... | Z

<number> ::= <dec number> | <hex number> H

<dec number> ::= <dec digit>
         | <dec num> <dec digit>
         | <dec num> $

<hex number> ::= <dec digit>
         | <hex num> <hex digit>
         | <hex num> $

<dec digit> ::= 0 | 1 | 2 ... | 9

<hex digit> ::= <dec digit> | A | B | C | D | E | F

<string> ::= <str element> | <string> <str element>

<str element> ::= <ASCII char> | ''
```

Figure A-2 Continued.   SPL/M Grammar

Feature Removed	Rationale	Comment
GOTO and labels	not needed	BREAK added
Imbedded & mult-iple assignments	use multiple statements	
DO CASE	use nested IF's	
Based variables	use MEM,MEMA	planned for V2.0
Iterative DO	use DO WHILE	planned for V2.0
Proc parameters	use globals	planned for V2.0
User defined functions	use globals	planned for V2.0
Shift functions	use * or /	planned for V2.0
LENGTH & LAST functions	use DCL LIT \<number\>	planned for V2.0
Factored DCL's	use multiple DCL'S	planned for V2.0
INITIAL attribute	use DATA	
DCL LIT \<string\>	use DCL LIT \<number\>	PROC, ADDR, DCL & LIT built-in
Binary & octal constants	use hexa-decimal	
Interrupt access and 6800 flags	write in assembly	

Figure A-3. PL/M Features Not in SPL/M V1.0

```
                              0012    152CH:
                              0013    GROUP: PROCEDURE;
                              0014
                              0015    DCL GPI BYTE;
                              0016
                              0017    400H: DCL GP$STK (20H) ADDR,
                              0018             GH$STK (20H) ADDR;
                              0019
                              0020    GP$STK(GPI) = GP$ADDR;
        152C:  96 7A
        152E:  5F
        152F:  48
        1530:  59
        1531:  8B 00
        1533:  C9 04
        1535:  BD FE 29
        1538:  96 66
        153A:  D6 65
        153C:  E7 00
        153E:  A7 01
                              0021    GH$STK(GPI) = GH$ADDR;
        1540:  96 7A
        1542:  5F
        1543:  48
        1544:  59
        1545:  8B 40
        1547:  C9 04
        1549:  BD FE 29
        154C:  96 68
        154E:  D6 67
        1550:  E7 00
        1552:  A7 01
                              0022    GPI = GPI + 1;
        1554:  7C 00 7A
                              0023    /* PARSE OUT DO OR DO-WHILE */
                              0024    CALL GRP$HEAD;
        1557:  BD 16 FA
                              0025    IF E = 0 THEN DO;
        155A:  96 5B
        155C:  27 03
        155E:  7E 00 00
                              0026       DO WHILE N <> END$TOKEN;
        1561:  96 5A
        1563:  80 86
        1565:  26 03
                              0027          CALL STMT;
        1567:  7E 00 00
        156A:  BD 18 1C
                              0028       END;
        156D:  7E 15 61
      (1567:  7E 15 70)
                              0029       IF GH$ADDR <> 0 THEN DO;
        1570:  DE 67
        1572:  26 03
        1574:  7E 00 00
```

Figure A-4. Example Compiler Printout

```
                              0030              /* GEN JUMP TO BEG OF DO */
                              0031                 CA(0) = 7EH;
1577:  86 7E
1579:  97 70
                              0032                 CA(1) = HIGH(GH$ADDR);
157B:  96 68
157D:  D6 67
157F:  17
1580:  97 71
                              0033                 CA(2) = LOW(GH$ADDR);
1582:  96 68
1584:  D6 67
1586:  97 72
                              0034                 CALL GEN3;
1588:  BD 0C 43
                              0035              /* FIXUP JUMP TO END */
                              0036                 PAT$ADDR = GP$ADDR;
158B:  DE 65
158D:  DF 69
                              0037                 CALL FIXUP;
158F:  BD 1A BB
                              0038              END;
                              0039           END;
(1574:  7E 15 92)
                              0040           GPI = GPI - 1;
(155E:  7E 15 92)
1592:  7A 00 7A
                              0041           GP$ADDR = GP$STK(GPI);
1595:  96 7A
1597:  5F
1598:  48
1599:  59
159A:  8B 00
159C:  C9 04
159E:  BD FE 29
15A1:  E6 00
15A3:  A6 01
15A5:  97 66
15A7:  D7 65
                              0042           GH$ADDR = GH$STK(GPI);
15A9:  96 7A
15AB:  5F
15AC:  48
15AD:  59
15AE:  8B 40
15B0:  C9 04
15B2:  BD FE 29
15B5:  E6 00
15B7:  A6 01
15B9:  97 68
15BB:  D7 67
                              0043           CALL EXIT$BLK;
15BD:  BD 15 AD
                              0044           END; /* GROUP */
15C0:  39
```

Figure A-4 _Continued._ Example Compiler Output

AN EXPERIMENTAL PASCAL-LIKE LANGUAGE FOR MICROPROCESSORS

H. Marc Lewis, Information Systems Manager
Regional Information Systems, Eugene, OR 97401

Introduction

This paper describes an experimental PASCAL-like high level language oriented to microprocessor implementation and use. The design criteria include modest memory requirements, self-compilation, simplicity, reasonable access to hardware features, and ease of extensibility. Program structure, data declarations, and control structures are described and examples given. Novel features of the language are discussed. An appendix gives a formal description of the language via syntax graphs.

Background

As a computer scientist I have always been disappointed with the plethora of articles about the BASIC programming language which have appeared in the various computer hobbyist publications. That is not to say I don't like BASIC itself, in fact I think it's an excellent language for introducing someone to the programming game. But it has its limitations, as do all languages, and is not well suited to the design and implementation of systems software. I am not a hardware type, and I don't particularly get off on tracking down electronic problems with my microcomputer. I am very interested however in systems software design, i.e. device drivers, operating systems, monitors, compilers, and even some end-user applications like real-time and process control stuff.

You may think I should be coding in Assembler, but I disagree. The debate on the merits and disadvantages of writing systems software in Assembler versus some higher level language has been going on for many many years among computer scientists, and the scales, though still not stabilized, have tipped towards high level languages. Burroughs Corp., always in the forefront of computer architecture in my mind, has built several machines designed to be programmed at the lowest level by a high level language. And even companies producing more conventional hardware architectures are now specifying that all systems software will be written in a high level language. Texas Instruments, for example, has chosen PASCAL for all development on their computers, from the micros to their super-computer, the ASC.

But enough rhetoric, the purpose of this talk is to explain why I felt justified is designing yet another language, and in particular why it looks the way it does. There are certain-ly plenty of languages and variants of languages now, so why another?

Well, to start with, there currently does not seem to be any suitable Systems Implementation Language (I'll abbreviate with SIL from now on) available for microprocessors. Sure I know that there is at least one good PASCAL implementation available for the Z-80, and there are 8080 and 6800 versions also in the works or on the street. But PASCAL is not really a candidate for a SIL, Dr. Wirth even admits that the language wasn't designed for that (however, his newest language, MODULA, is) even though it contains an excellent assortment of data and control structures which allow straightforward and well-structured program design. And I hope we all agree that BASIC, FORTRAN, PILOT, etc. are not even worth considering as SIL's.

So, where does that leave us? How about in a place where some new language, incorporating some of the better features of more general and ambitious high level languages, and which was very modest in its requirements for memory, could fit? The language I've designed, which incorporates features of PASCAL, MODULA, and C, is just such a language. Incidently, I haven't yet named the language so please forgive the awkward references to simply "The/This Language".

Basic Design Criteria

The following criteria were considered in the design of this SIL:
1. Small memory requirements (less than 16K if possible).
2. Self-compiling: The language would first be developed for one particular machine and then the compiler would be written in its own language. Thus by simply modifying the code-generating portions of the compiler it could be adapted for any target machine.
3. Simplicity: The language, and its compiler implementation, should be as simple and straightforward as possible to allow novices in compiler construction to modify it to their particular requirements.
4. Versatility: The language should allow access to as much of the hardware as possible, including the stack, I/O registers, the accumulator(s), etc.
5. Extensibility: This is an experimental language, therefore it should be easy for someone to experiment with it, for example includ-

ing some desirable feature or extending the
language to make it more suitable for a par-
ticular application.

The Language Itself

A program consists of zero or more data
definitions, followed by zero or more PROCEDURE
definitions, followed by one or more executable
statements, in the form of a BLOCK. The lan-
guage is "block-structured" in the manner of
ALGOL, PL/1, and PASCAL.

Data Declarations. The simple data types
supported are INTEGERs, CHARs, and STRINGs. A
DECIMAL type may be added later if it seems
necessary. Constants of either type may be
defined.

Singly dimensioned arrays may be defined
for INTEGERs and CHARs by appending the integer
array size enclosed in brackets to the defini-
tion statement. An array of size "n" will have
"n+1" elements, the first element being the zero-
th element, which in the case of character arrays
contains the current length of the string.
Arrays of strings are implemented as a vector
of pointers to the individual character arrays.
The maximum length of a character array is 255
characters due to the 8-bit character size of
most machines.

Strings are actually pre-defined character
arrays. The size of the strings for any given
program is specified once, in the data declara-
tion section, by appending the integer string
size (between 0 and 256) enclosed in brackets
to the STRING keyword. Examples of data dec-
larations:

```
INTEGER A,B,C;
CHAR LINE[80], NEXTCHAR, EOFBYTE;
STRING[64] PAGE[32], HEADER;
```

There is no static initialization of var-
iables as in PL/1. Initialization must be per-
formed explicitly (and dynamically) via an
assignment statement a la PASCAL and MODULA.
This requirement arises out of the necessity
for recursion in PROCEDURE calls.

To simplify the compiling process CONSTants,
which may not be dimensioned, must be declared
first, then any INTEGERs, CHARacters, or STRINGs.
The syntax charts in appendix A-1 depict this
graphically.

Comments. Comments may appear anywhere in
a program and are enclosed by a familiar "slash-
star" and "star-slash" of PL/1 and PL/M. PASCAL
type comments, delimited by (* and *) are also
valid. Comments within comments are not recog-
nized.

Control Sturctures. For control structures,
the language contains the standard sequencing
structure of sequential execution of statements
unless modified by selection or repetition

structures. Statements are separated by semi-
colons as in PASCAL, rather than using semi-
colons to terminate all statements as in PL/M.

In this language assignment is denoted by
the familiar two-character symbol ":=" of
PASCAL, C, and MODULA. This symbol (which I
read as "gets") has the advantage of having a
single meaning, unlike the "=" operator of
PL/M which indicates either assignment or a
test for equality, depending upon the context.
Examples of assignments are:

```
A:=B
A:=SIZE MOD 2
CELL[I]:=SIZE*8
```

A novel feature of this language is that
expressions are restricted to the form of a
single (possibly signed) constant or variable,
or a pair of constants and/or variables separ-
ated by an operator. Operators are arithmetic
(+,-,/,*,MOD), boolean (AND, OR, XOR), rela-
tional (>,<,>=. <>), or shifting (<<,>>).

An unary "address of" operator (the argyle)
and an increment/decrement operator (++,--)
are also provided. For example, A:=@B; puts
the address of variable "B" into variable "A",
++A; increments variable "A" by one, and
A:=A<<3; shifts variable "A" left 3 bits.

The justification for limiting the com-
plexity of expressions comes, again, from the
design goal of keeping the compiler small, and
from the desire to see if such a restriction
places undue constraints on the programmer.
In a study of a large number of student-written
FORTRAN programs by Knuth, it was found that 9
out of 10 assignment statements had only one
or two terms on the right of the equals sign.

The IF statement and the CASE statement
comprise the selection control structures.
The IF statement has the forms:

```
IF condition THEN statements END
IF condition THEN statements
    ELSE statements END
IF condition THEN statements
    ELSIF condition THEN...END
```

and the CASE statement has the form:

```
CASE expression OF
    caselabel:  BEGIN statements END;
                  .
                .
                  .
    caselabel:  BEGIN statements END
END
```

These are fairly common forms for these con-
structs and are borrowed directly from MODULA.

Repetition is handled by the LOOP construct
of MODULA. This single type of statement is
sufficient to express all repetitions. Thus,
in striving to keep the language and its com-
piler small, no WHILE, REPEAT, FOR, or DO
statements are included.

The general form is:

```
LOOP statements WHEN condition DO
     statements EXIT
     statements WHEN condition DO
     statements EXIT
              .
              .
              .
     statements WHEN condition DO
     statements EXIT
     statements
END
```

However, more simple forms of the LOOP statement are also valid:

Example of an infinite LOOP

```
LOOP GETCH(CELL[I]; ++I END
```

Example of a 10-Iteration LOOP

```
I:=0; (* initialize loop counter *)
LOOP GETCH (A);
     ++I; CELL[I]:=A
     WHEN I=10 EXIT
END
```

Example of a "WHILE" LOOP

```
LOOP WHEN CH=';' EXIT
     GETCH(CH)
END
```

Example of a "REPEAT" LOOP

```
LOOP GETCH(CH)
     WHEN CH=';' EXIT
END
```

Notice that there is no LABEL definition as in PASCAL. This is because there is currently no GOTO statement, thus labels (excluding CASE labels, which are constants) are unnecessary. As with MODULA, the omission of the GOTO statement is an experiment only, and if it fails the GOTO (and labels) will be included in the language.

Procedures

PROCEDURE calls are subroutine calls in the PL/M sense. No function calls are defined (except predefined or external PROCEDURES like GETCH, ORD, etc.). All PROCEDURES are declared immediately after the data declarations and before the first executable statement of the main program. Parameters are passed by reference, not by value. This allows parameters to be modified by the called PROCEDURE. Examples of PROCEDURE calls are:

```
DOIT(A,B,100*8)
SWITCH(CELL[I],CELL[J])
READLN
```

PROCEDUREs may be called recursively, therefore the allocation of storage for local variables must take place at execution time rather than at compile time. Each PROCEDURE has asso-

ciated with it a data segment called an "activation record" which contains 1) the return address of the caller, 2) a dynamic link pointer to the activation record of the calling PROCEDURE, and a static link pointer to the activation record of the PROCEDURE in which the called PROCEDURE is declared, and 4) the storage for locally declared variables.

Built-In Functions

Built-in functions include:

CHR(n) -returns a character whose numerical value in the collating sequence is given by "n".

ORD('a') - returns the integer value of the ASCII character 'a'.

GETCH(A) -gets the next character from the input stream (probably the terminal) and places it in character variable "A".

PUTCH(A) -prints or displays upon the standard output device the character contained in variable "A".

READ(A,B,...) -performs formatted input. Character variables are filled to their current length with the next 'n' characters from the input stream, and integers are converted from ASCII to binary and placed into their corresponding INTEGER variables.

WRITE(a,b,...) -performs formatted output. The inverse of READ.

READLN(...) -like READ but will position the input stream to the first character of the next line before returning.

WRITELN(...) -like WRITE but terminates the current line with appropriate new line character(s).

PUSH(A) -pushes variable "A" onto the stack

POP(A) -removes the top element of the stack and places it into "A". If no variable is specified it simply decrements the stack pointer by one.

STACK(n) -places the value 'n' in the stack pointer register.

Formal Language Definition

I have chosen to describe the language via syntax graphs, rather than the more common BNF form. I did this for two reasons, first because syntax graphs are easier for humans to read and understand, and second because the compiler will be written with a technique called recursive descent (a top-down or goal-

directed approach) which can be directly derived from the syntax graphs. Most PASCAL compilers use this technique. There are superior compiler construction schemes, my own favorite is called SLR(k), a table driven technique, but given the design criteria the recursive descent method was the best choice.

The graphs are read from left to right, and generally, from top to bottom. The arrows show the flow of the parser. Symbols enclosed in circles or oblongs correspond to terminal symbols, i.e. variables, reserved words, operators, constants. Symbols enclosed in boxes or rectangles correspond to a recognizer for that particular non-terminal, i.e. a CASE statement in its entirety, a BLOCK, etc. Such recognizers are implemented as PROCEDUREs to recognize the desired construct. Where the arrows branch either path may be taken, thus in the example which follows, an "Integer Declaration" consists of the reserved word "INTEGER" followed by an identifier name, optionally followed by a left bracket, optionally followed by a comma, another identifier name, etc.

Integer Declaration
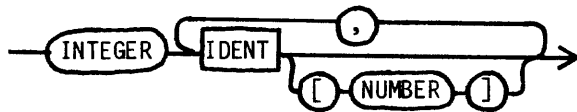
Example Program

```
/* find the smallest & largest number in
   a given list    */
  CONST N=10;
  INTEGER  I,X,Z,MIN,MAX,A[10];

     /* assume that array A has been
        initialized with the values
        12, -6, 98, 7, 47, -1, -50,
        0, -72, 33      */
BEGIN
 MIN:=A[1];  MAX:=MIN;  I:=2;
 LOOP
  X:=A[I];   Z:=A[++I];
  IF X>Z THEN IF X>MAX THEN MAX:=X END;
              IF Z<MIN THEN MIN:=Z END
  ELSE IF Z>MAX THEN MAX:=Z END;
       IF X<MIN THEN MIN:=X END
  END;
  ++I;
  WHEN I>N EXIT
 END
 WRITELN(' Max. value is ',MAX);
 WRITELN(' Min. value is ',MIN)
END.
```
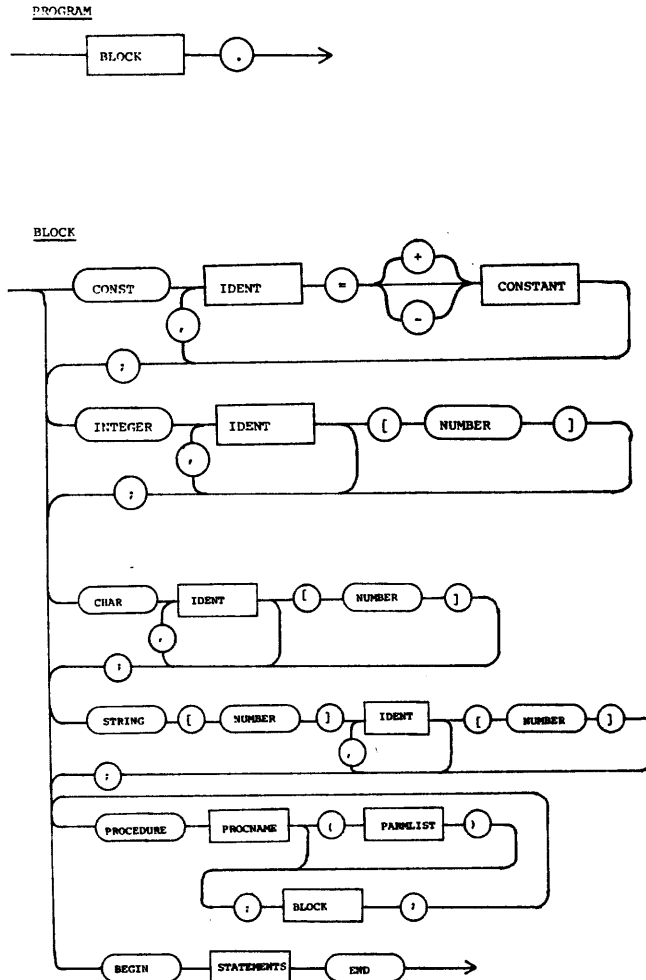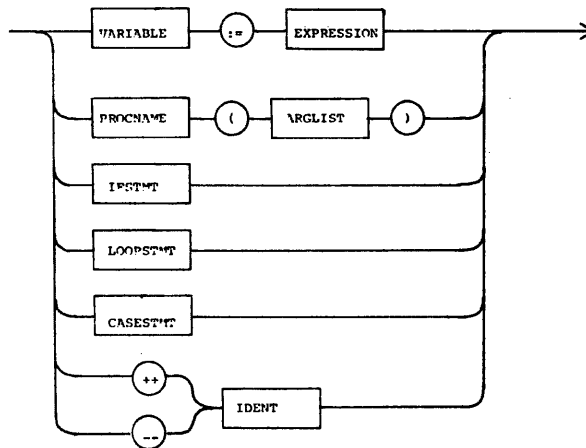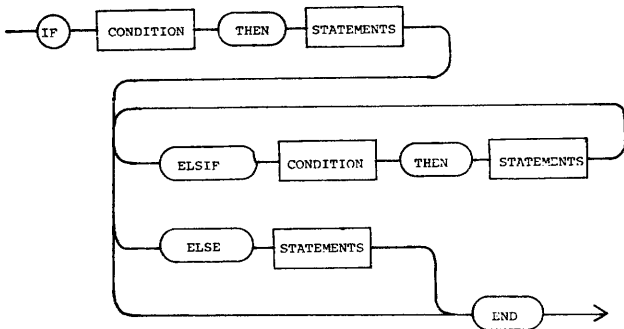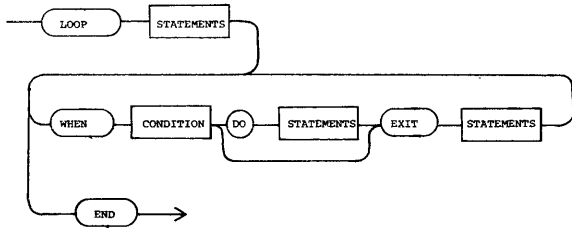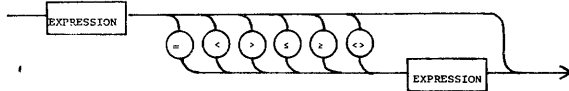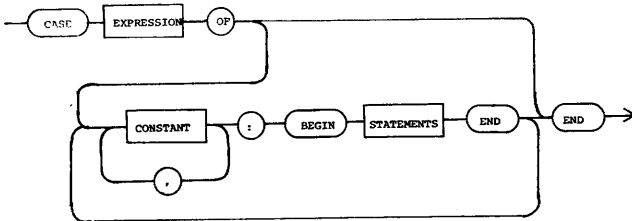
PROGRAM

BLOCK
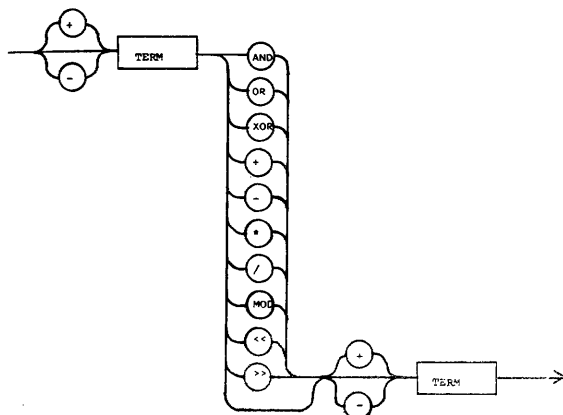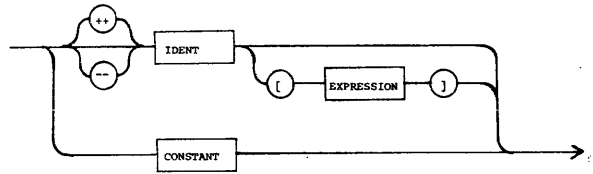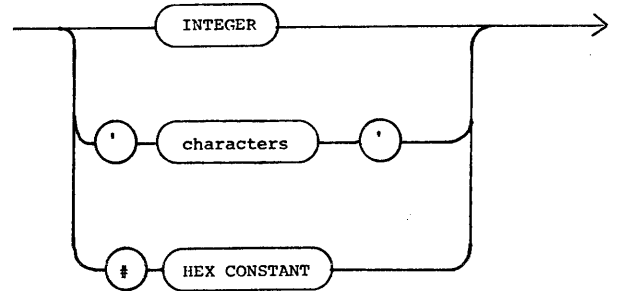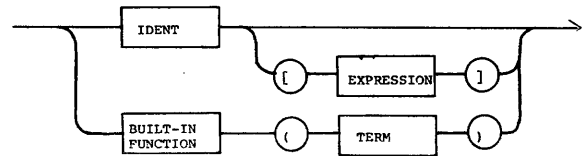
STATEMENT
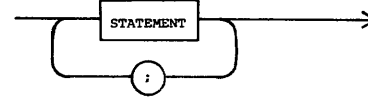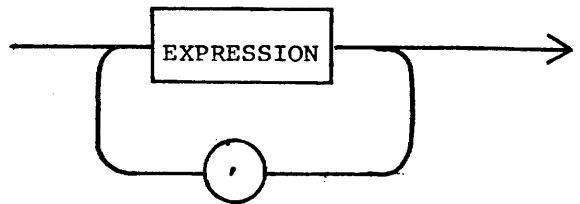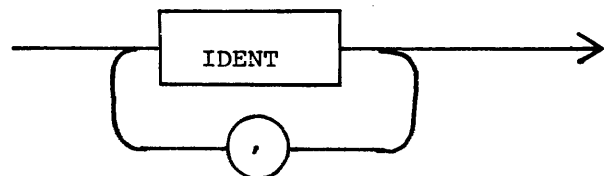
AN INTRODUCTION TO PROGRAMMING IN PASCAL

Chip Weems
Graduate Teaching Assistant
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Abstract:

This paper will concentrate heavily on the use of the Pascal language at the beginner's level. A minimal knowledge of some other programming language such as FORTRAN, BASIC or ALGOL is assumed.

The areas which will be covered are simple and structured statements in Pascal, simple and structured data types, plus procedures and functions. Emphasis will be placed on using Pascal statements, although some discussion of the power of user defined data types will also be included.

A list of machine models for which implementations of Pascal are known to exist, is provided as an appendix.

Part One: What is Pascal?

Historical Introduction:

Pascal is not an acronym, unlike many languages the letters which make up its name do not stand for anything. This is perhaps a first indication that Pascal is something different and a little special.

Pascal was named after the famous mathematician Blaise Pascal (1623 - 1662) who, among other things, invented an eight digit calculating machine which could perform addition and subtraction. Multiplication and division were performed by repeated addition or subtraction, respectively. He completed the first operating model at the age of 19, and built 50 more during the next 10 years.

The Pascal language was originally specified in 1968 by Niklaus Wirth at the Institut für Informatik, Zurich. This makes it a relative newcomer to world of programming languages. The first Pascal compiler became operational in 1970 and was published in 1971.

The following table shows just how new Pascal really is. Remember that most compilers are not introduced until three to five years after their initial specification. (For example, APL was initially spacified in 1962.)

Language	Introduction Date
FORTRAN	1957
COBOL	1960
ALGOL	1960
LISP	1961
SNOBOL	1962
BASIC	1965
PL/1	1965
APL	1967
Pascal	1971

After two years of experience, the language was revised and re-released in 1973. This version of the language is now generally referred to as standard Pascal. The important thing to note here is that Pascal was the first major new language to be developed after the concept of structured programming was introduced.

Structured Programming and Pascal:

There exists no exact definition of structured programming, although it has been termed "A collection of all good and wonderful programming practices." One fact becomes obvious in discussing it with groups of programmers: Some people love it, and some people hate it. However, those who hate structured programming are now finding themselves more often in the minority.

Some features to be found in a structured program are that it is generally more readable and more easily shown to be correct. The design of a structured program usually involves stepwise refinement, or top-down programming. Languages designed with

structured programming in mind will usually include a large group of program-flow control structures, which are entered at only one point and from which there is only one exit. Another notable point about such languages is that they often require explicit definition of all variables and data structures in the code. What does all of this mean? How does it relate to Pascal?

Readability:

One of the outstanding features of Pascal is that well written Pascal code is very readable; more so than most other programming languages. Probably the greatest single factor which makes this language so easy to follow, is the construction of data names. In Pascal there is no limit to the acceptable length of names. Generally, the compiler only uses the first eight characters of a name to distinguish it from all others, with the remainder of the name simply being ignored. This lack of constraints usually leads to very meaningful names in Pascal. Note that I have specifically avoided writing 'variable names'. Pascal permits not only variables to be named, but also constants, files, records, complex data structures, procedures and functions; all with the same naming conventions in effect. Compare this to ather languages such as BASIC or FORTRAN!

Pascal's readability is also enhanced by the wording of its statements. When meaningful names are used, almost always the coded statements will make sense as english phrases. This would almost seem to take the place of program comments, but even so, Pascal provides one of the most flexible commenting schemes possible. Comments may appear anywhere in a Pascal program except in the middle of words!

Stepwise Refinement:

In writing a Pascal program it becomes very easy to use top-down programming style. This is mainly due to the flexibility and ease of writing procedures and functions. It is not unusual to see incredibly complex Pascal programs, several hundred lines long, in which the main program accounts for less than one hundred lines. Such a main program will usually consist of the overall program flow-logic with dozens of calls to well-named procedures and functions.

Procedures and functions correspond roughly to subroutines and functions in FORTRAN, but are actually part of the Pascal program. This means that procedures and functions inherit all variables defined in the main program, similar to subroutines in BASIC, but they can also include declarations of variables and constants which are only valid within themselves.

It should also be noted that procedures and functions are fully recursive in Pascal, that is they may in turn call themselves.

Simply using the name of a procedure or function will invoke it; thus it becomes very easy to write code with procedure names and worry about all of the messy details at a later date. This is, of course, the basis of top-down programming.

Explicit Definitions:

Another level of stepwise refinement is careful pre-planning of a program. Usually, Pascal programs are most easily planned-out by using a form of loose, english-like pidgin ALGOL.

One thing should be noted here: Pascal is probably best classified as a descendant of ALGOL. People who know ALGOL seldom have any difficulty in learning Pascal. In fact, ALGOL-60 is generally considered to be a subset of Pascal.

Careful pre-planning is encouraged by the fact that Pascal has very rigid rules requiring virtually all data structures to be defined at the start of a program. Unlike many languages, you can't just throw in an extra variable, in the code, when you discover that you need it. Because Pascal also requires such things to be defined, careless pre-planning often becomes quite self-evident just by looking at the declarations. This feature is something which BASIC programmers typically have a hard time getting used to, but it often makes Assembly Language hackers feel right at home.

Probably the greatest single new idea to come out of Pascal is the user defineable data type. This construct, which appears in the declarations, permits the programmer to specify new types of data beyond the standard Real, Integer, Character and Boolean types. Data types of arbitrary complexity may be constructed; in fact adding complex numbers to a Pascal program is generally considered to be a trivial case!

Users may define data types as outrageously complex as say, a five dimensional array of records of arrays, scalars, records with variant parts, pointers and complex numbers. The programming power added by this concept is almost difficult to imagine; it provides us with the ability to create structured data as well as structured processes.

Single-Entry / Single-Exit Control

Structures:

One of the requisites for being able to show that a program will work correctly is that it must be possible to trace out all of the possible execution paths, through the program, for given sets of inputs. Usually, this is done by first breaking the program down into small units, showing that each unit works correctly, and then showing that combinations of units work correctly and so on.

This all sounds very simple, except for one item -- the GOTO statement throws a monkey wrench into the whole thing. The problem is that it doesn't take too many GOTO's combined with conditional branches, before an almost infinite number of possible execution paths appear in a program. How can you prove that a block of code will perform correctly, when you can't even be sure where it will be entered from, or where control will exit to, once it has completed?

As an example, consider a section of a BASIC program, possibly a scoring routine for a game, which is invoked by GOTO's from 20 different locations. In addition, these GOTO statements jump into the scoring routine code at six different points, depending on flags set by previous passes through the routine, and upon other outside events. Depending on the data present and the entry point, the routine may branch to several places in itself, loop in two places, or fall straight through. Also, when it completes, depending on outside conditions and also upon previous passes through it, the routine may branch to any one of eight other program sections. Stop and think about how much effort it would take to trace all possible paths through such a mess! This code might be clever and efficient, but is it worth all of the headaches which it will cause in the long run?

Not only is such convoluted logic difficult to follow and understand, but it is also a major chore to get all of the bugs out of it; and you can never be sure that all of them ARE out. As if that isn't enough, just try to make a major change to such a piece of code -- it would probably be easier to discard the whole thing, rather than try to patch it.

Now that we've raked the GOTO statement over the coals, what is there which will take its place? The answer is: single-entry / single-exit control structures. Flow of control, in a program, always enters the top of such a structure, and will only exit out through the bottom. This means that, if the program unit inside of the structure is correct, we can trace an effective straight line through the whole thing. A familiar example of a single-entry / single-exit structure is the FOR-NEXT loop in BASIC, but without any GOTO's which enter or or leave the middle: Flow will enter at the top, looping will occur, but eventually flow will continue through the bottom of the FOR-NEXT.

As it turns out, there are only three structures required to replace the GOTO statement. They are: The WHILE statement, the IF-THEN statement, and Compound statements. In Pascal, anyplace a statement can go, may be placed a Compound statement. Compound statements consist of the word BEGIN, followed by any group of statements (Which may include more Compound statements.), followed by the word END. Pascal also includes WHILE and IF-THEN statements, plus several other single-entry / single-exit structures which add to the convenience of GOTO-less programming.

The following is a list of all of the structured statements in Pascal along with flowchart segments to indicate how they function:

This is, of course, the well known IF-THEN statement:
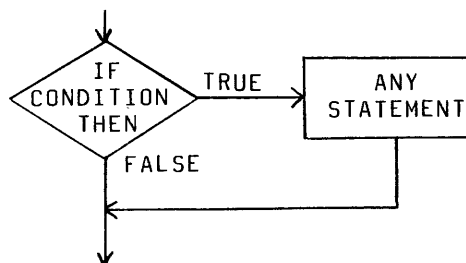
A convenient form of the IF-THEN statement is the IF-THEN-ELSE:

```
                    IF
       FALSE    CONDITION    TRUE
                   THEN
    ANY                        ANY
  STATEMENT                  STATEMENT
```

The WHILE statement has the form:

```
           WHILE
         CONDITION    FALSE
            DO
           TRUE
           ANY
        STATEMENT
```
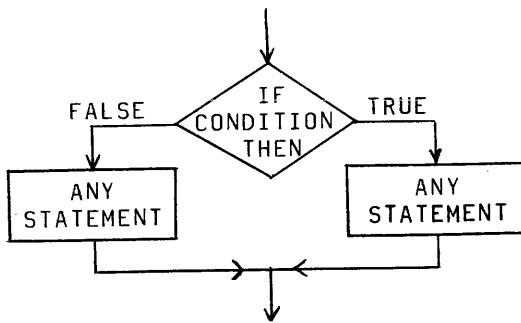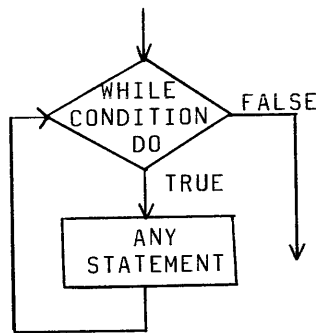
This next one is the REPEAT-UNTIL statement. There is an important difference between this and the WHILE statement which should be noted: If the condition is false, when a WHILE statement is entered, no action takes place -- control skips around the ANY STATEMENT part. In a REPEAT-UNTIL however, the ANY STATEMENT part always gets executed at least once, regardless of the conditional part.

```
         REPEAT ANY
         STATEMENTS

            UNTIL
   FALSE   CONDITION
             TRUE
```
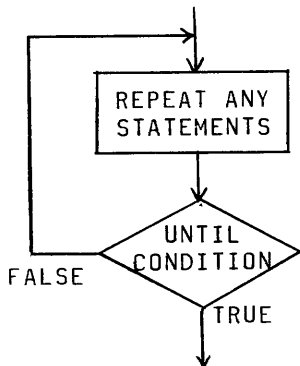
The FOR-UPTO statement is very similar to the FOR-NEXT statement in BASIC, except that it is restricted to an increment of 1. This is intended to add to the reliability of the construct,

since most digital computers can not exactly represent fractional numbers. If other increments were permitted, it might be possible for the increment to not exactly match the terminator when it reached the desired value, and so perhaps the loop would continue for an extra pass. This is a very frustrating problem, because it is usually highly machine dependent, and will typically only show up in a very few specific instances. All of this is eliminated by Pascal's restriction of the increment value to 1. One positive side effect which results from this is that the speed of the statement is often greatly increased, since many machines have single instructions for incrementing and testing memory locations, or registers.

The FOR-DOWNTO statement is identical to this, except that the index is decremented by one, each time through the loop:

```
        INDEX:=
      INITIAL VALUE

        INDEX >
      FINAL VALUE    TRUE
          ?
        FALSE
          ANY
        STATEMENT
      INDEX:=INDEX+1
```
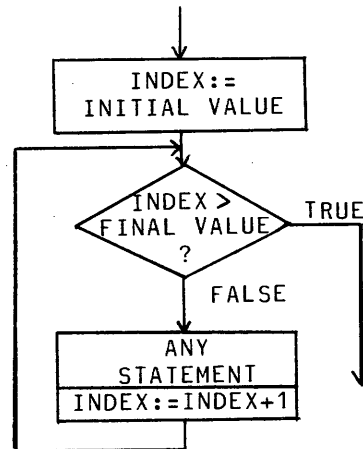
This last one is the CASE statement, which is somewhat like the ON-GOTO statement in BASIC:

```
            LABEL
          EXPRESSION

LABEL LIST 1         ANY
                   STATEMENT

LABEL LIST 2         ANY
                   STATEMENT

LABEL LIST n         ANY
                   STATEMENT
```
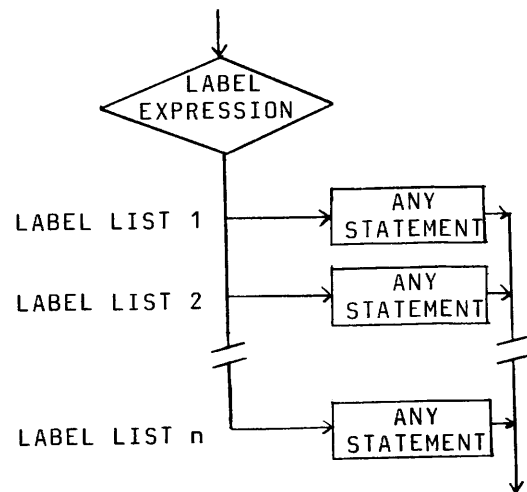
All of this should not be taken to imply that Pascal is a GOTO-less language; it does have labels and GOTO's. The important point is that the experienced Pascal programmer will almost never use them, since they are never needed and only rarely of any value.

Part Two: A summary of Pascal statements, with examples.

Character Set:
The standard Pascal character set includes: Letters A - Z (and depending on the implementation, a - z), numbers 0 - 9, special characters + - * / = < > ( ) [ ] . , ; : ' ↑ (and the space or blank character).

Names:
• Names in Pascal consist of letters and/or digits, and may be any number of characters in length. The first character must be a letter, and the first 8 characters must be different than the first 8 characters of any other name.

Examples:

ENDOFDATA TYPES AVERAGE SN7473A
TOTAL SCORES PAYRATE CARDCOUNT

Numbers:
Numbers in Pascal are either real or integer. They may be signed or unsigned.
Integers are a string of digits.

Examples:

+7 43 365 -18 8388607 4092 0

Reals have three forms:

digits.digits
digits.digitsEscale factor
digitsEscale factor

The E notation indicates multiplication by 10 raised to the scale factor power.

Examples:

3.1415 6.02E23 9.11E-31 -1E9

Note that the scale factor is always an integer.

Comments:
Anything typed between the symbols (* and *) will be ignored by the compiler as comments. On systems which have them, curly brackets{ } are used instead.

Operations:
Integer operations

*	Multiplication
DIV	Division (Integer part only, remainder discarded.)
+	Addition
-	Subtraction
MOD	Modulo (A MOD B = A-((A DIV B)*B))

Real operations

*	Multiplication
/	Division
+	Addition
-	Subtraction

Boolean operations

AND	Logical AND
OR	Logical OR
NOT	Logical NOT

Relational Operations (give boolean results)

<	Less than
>	Greater than
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
IN	Used with data type SET, to determine membership of an element

Examples:

A * B	A times B
X DIV Y	X divided by Y
TOP <= BOTTOM	Numerical comparison
ABOVE AND BEYOND	True if both (ABOVE and BEYOND) are true boolean variables.

Functions:

Name	Action
ABS	Absolute value
SQR	Square
TRUNC	Truncate to integer part
ROUND	Rounded-up integer form
SUCC	Next highest (Integer or Char)
PRED	Next lowest (Integer or Char)
SIN	Trigonometric sine
COS	Trigonometric cosine
ARCTAN	Trigonometric arctangent
LN	Natural (Base e) logarithm
EXP	e raised to the power
SQRT	Square root
ORD	Numeric value associated with the character

Name	Action
CHR	Character associated with the numeric value
ODD	True if the integer argument is odd
EOLN	True when end-of-line is reached
EOF	True when end-of-file is reached

	Result type for argument of type		
Name	Integer	Real	Character
ABS	Integer	Real	
SQR	Integer	Real	
TRUNC		Integer	
ROUND		Integer	
SUCC	Integer		Character
PRED	Integer		Character
SIN	Real	Real	
COS	Real	Real	
ARCTAN	Real	Real	
LN	Real	Real	
EXP	Real	Real	
SQRT	Real	Real	
ORD			Integer
CHR	Character		
ODD	Boolean		
EOLN	Argument is always a file name, result is always boolean.		
EOF	Argument is always a file name, result is always boolean.		

Statements:

Program Heading:

PROGRAM programname (filename, filename,...);

Exmaple:

PROGRAM TESTSCORES (INPUT,OUTPUT);

Constant Definition:

CONST constname = value; constname = value;...

Example:

CONST ENDOFDATA = -1.0; PI=3.141592;
      MAXSCORE = 100; MINSCORE = 0;

Note that the constant definitions can continue onto more than one card, but the CONST is only typed once.

There are some predefined values which do not need to be declared as constants in Pascal programs. These are:

TRUE	Boolean true value
FALSE	Boolean false value
MAXINT	Largest integer the computer can work with
NIL	Null pointer

Variable Definition:

VAR varname, varname,...:type;
    varname, varname,...:type;...

Example:

VAR SCORE,MAX,MIN,TOTAL: INTEGER;
    RADIUS,DIAMETER,CIRCUMFERENCE:
    REAL;
    FOUND,DONE,FLAG,OK:BOOLEAN;

Note that the declarations may continue on several lines, but only one VAR is required.

Procedure Definition:

PROCEDURE procname (value
    parameters; VAR variable
    parameters);
body of procedure

Example:

PROCEDURE INCREMENTBY (INCREMENT:REAL;
    VAR VARIABLETOBEINCREMENTED:REAL);
BEGIN
    VARIABLETOBEINCREMENTED :=
        VARIABLETOBEINCREMENTED +
            INCREMENT
END;

Function Definition:

FUNCTION functnam, (value
    parameters): result-type;
body of function

Example:

FUNCTION RADIUS (CIRCUMFERENCE:REAL):
    REAL;
CONST TWOPI = 6.2831;
BEGIN
    RADIUS:=CIRCUMFERENCE/TWOPI
END;

Assignment Statements:

varname := expression

Examples:

WEEKSPAY := PAYRATE * HOURSWORKED;
VOLTS:=AMPS*OHMS;
CONEVOLUME:= (PI*SQR(RADIUS)*HEIGHT)
    /3.0;
ARRAYLOCATION :=ARRAYLOCATION + 1;

Note that the assignment statement is very free-form: Spaces may be inserted as needed, the assignment may continue onto more than one line, etc. The only restriction is that words can not be broken in the middle.

The Compound Statement:
     In Pascal, any place where a state-
ment can be used, a compound statement
may also be used. A compound statement
is formed by the word BEGIN, a group of
any statements, followed by the word
END.

Examples:

```
BEGIN
     SCORESUM:=SCORESUM+SCORE;
     SCORECOUNT:=SCORECOUNT+1
END

BEGIN
     X:=(Y+Z)/100;
     BEGIN
          T:=(Q/75)+15;
          F:=N-18
     END
END
```

Placement of Semicolons:
     The simplest rule for the place-
ment of semicolons, in a Pascal pro-
gram, is: Place a semicolon between
any two Pascal statements.
     Note: BEGIN and END are not Pascal
statements, they are simply delimiters.
A compound statement is a statement, and
must be separated from other statements.
Also note one exception in the rule --
The ELSE in the IF-THEN-ELSE takes the
place of a semicolon in separating the
two statements.

Conditional Statements:

The IF-THEN Statement:
     IF expression THEN statement

Example:

```
IF MAXSCORE < SCORE THEN MAXSCORE:=
     SCORE
```

The IF-THEN-ELSE Statement:
     IF expression THEN statement ELSE
statement

Example:

```
IF TIME < 0 THEN TIME:=0 ELSE TIME:=1
```

The CASE statement:
     CASE expression OF
          case-label-list:statement;
          case-label-list:statement;
                    •
                    •
                    •
          case-label-list:statement
     END

Example: (* Determine command group
from a command number *)

```
CASE COMMANDNUMBER OF
    0,1,3  : GROUP:=1;
      2,4  : GROUP:=2;
    5,9,11 : GROUP:=3;
    6,7,8  : GROUP:=4;
        10 : GROUP:=5
END
```

Repetitive Statements:

The WHILE-DO Statement:
     WHILE expression DO statement

Example:

```
WHILE NOT EOF(INPUT) DO
BEGIN
     READ(SCORE);
     SCORESUM:=SCORESUM+SCORE;
     SCORECOUNT:=SCORECOUNT+1
END
```

The REPEAT-UNTIL Statement:
     REPEAT group-of-statements UNTIL
expression

Example:

```
REPEAT
     X:=X-1;
     Y:=Y+1
UNTIL (X < 0) OR (Y > 0)
```

The FOR Statement: (Two forms.)
     FOR control-variable := initial-
value TO final-value DO statement
     FOR control-variable := initial-
value DOWNTO final-value DO statement

Examples:

```
FOR INDEX := 1 TO ARRAYTOP DO
     ARRAY[INDEX] := 0
FOR INDEX := 100 DOWNTO ARRAYBOTTOM
     DO IF ARRAY[INDEX] < 0 THEN
          ARRAY[INDEX]:= 0
```

Transfer of Control Statements:

     The conditional and repetitive
statements previously described are
sufficient control structures to
perform any required computation.
Remember that although labels and
GOTO's are provided in Pascal, they
are unnecessary and will often only
create confusion in program logic.
Therefore it is recommended that the
be avoided except in those rare ex-
treme cases where they actually have
some value.

Label Definition:
      The label definition is placed
after the CONST declarations in the
program.
      LABEL integer, integer, ...;

Example:

LABEL 10, 20, 25, 100, 9999;

GOTO Statement:
      GOTO label

Example:

GOTO 9999

Input and Output in Pascal:

      Pascal I/O statements are not
really statements, but are actually
calls to predefined procedures. None
the less, they are often referred to as
statements.

Input Procedures:
      READ(variable-list)
      READLN(variable-list)
      READ(filename, variable-list)
      READLN(filename, variable-list)

Examples:

READ(X,Y,Z,MAXVAL)
READLN(HIGHSCORE,LOWSCORE,AVGSCORE)
READ(WEATHERFILE,TEMP,HUMIDITY,PRESSURE)
READLN(CUSTOMERFILE,NAME,NUMBER,BALANCE)

      The filename must have been de-
clared in the program heading.
      The difference between READ and
READLN is that successive READ state-
ments will continue to input successive
values from the same record, only
going to a new record when all values
on the current one have been exhausted.
A READLN, on the other hand, will skip
any additional values on the current
record, and go to the next record to
begin reading values.

Example:
      Two records:

      ┌─────────────────┐
      │ 0.0 1.0 2.0   │
      └─────────────────┘
      ┌─────────────────┐
      │ 3.0 4.0 5.0   │
      └─────────────────┘

READ(A,B);
READ(C,D)

The result of this would be A=0.0,
B=1.0, C=2.0, D=3.0.

READLN(A,B);
READLN(C,D)

Would result in A=0.0, B=1.0, C=3.0,
D=4.0.

Output Procedures:
      WRITE(expression-list)
      WRITELN(expression-list)
      WRITE(filename, expression-list)
      WRITELN(filename, expression-list)

Examples:

WRITE(A,B,C)
WRITELN(X*Y/Z,MAX,SQRT(Q),'*****')
WRITE(NEWFILE,NAME,ADDRESS,PHONE,
      AMT+1.0)
WRITELN(PLOTFILE,XCOORD,YCOORD,PENPOS,
      MARK)

      Successive WRITE statements cause
the values to be written, all as one
record. Each time a WRITELN is exe-
cuted, however, a new record is output.

Examples:

WRITE('A','B');
WRITE('C','D')
                  Would output ABCD.
WRITELN('A','B');
WRITELN('C','D')
                  Would output AB
                               CD.

      Formatting numeric output is very
easy in Pascal. Each expression in a
WRITE or WRITELN can actually have one
of the following three forms:

expression
expression:width-expression
expression:width-expression:fraction-
      width-expression

      The expression gives the value
which is to be output. The width-
expression gives the minimum number of
character positions to be included in
the output. If the expression value
isn't require all of the positions,
extras will be filled with blanks.
If the number is too big to fit in the
area, the area size is expanded to
accommodate the number.
      The fraction-width-expression spec-
ifies how many digits will be printed
to the right of the decimal point for
a real number.

Examples:

A=100, B=1.5, C=137875.3217,
D=128.34152

WRITE(A:5,B:5,C:5,D:9:3)   would output

   100   1.5137875.3217   128.341

WRITE(A:3,B:5:2,C:9:1)   would output

100 1.50 137875.3

Carriage Control:
    Although this is machine and im-
plementation dependent, most Pascal
systems will destroy the first char-
acter of each record output to a
printing device. Thus, an extra char-
acter must be provided at the start of
each output line, usually a space.
    In reality, this character acts as
a carriage control command, which is
either directly implemented in the
hardware of the printer, or which is
simulated by the monitor or operating
system, in software.
    The following are the standard
carriage control command characters
used in Pascal:

Character	Action
Space	Normal, single spacing
0 (Zero)	Double space, skip 1 line
1	Page eject

    Depending upon how the carriage
control is implemented, using other
characters may have different effects,
which may, or may not be desireable.

Data Types:

    All data type definitions are
placed between the CONST and VAR
declarations at the start of the
program.

Scalar Types:
    TYPE typename = (identifier,
identifier,...);

Example:

TYPE MONTH = (JAN,FEB,MAR,APR,MAY,JUN,
            JUL,AUG,SEP,OCT,NOV,DEC);

Subrange Types:
    TYPE typename = constant..constant;
    VAR varname-list : constant..con-
stant;

Examples:

TYPE LETTER = A..Z;
TYPE WINTERTERM = JAN..MAR;
VAR SCORE:0..100;

Array Types:
    TYPE typename = ARRAY[index-type]
OF element-type;
    VAR varname-list : ARRAY[index-
type] OF element-type;

Examples:

TYPE COEFFICIENTS = ARRAY[0..4] OF REAL;
VAR SAMPLELIST = ARRAY[0..100] OF REAL;

    Note: INTEGER and REAL are not
permitted as index types.
    Multidimensional arrays are defined
by specifying multiple index-types.
    TYPE typename = ARRAY[index-type,
index-type,...] OF element-type;
    VAR varname-list : ARRAY[index-
type,index-type,...] OF element-type;

Examples:

TYPE SIMLINEQS = ARRAY[0..5,0..6] OF
               REAL;
VAR FOURSPACE : ARRAY[0..10,0..10,
    0..10,0..10] OF INTEGER;
VAR NAMELIST : ARRAY[1..100,1..30] OF
    CHAR;

    Packed arrays are almost identical
to normal arrays, except that by dec-
laring an array to be packed, it may
be possible to reduce the size of the
memory space used by it. The amount
of reduction depends upon the machine
and the implementation, and may in
fact be nil. This may also reduce the
running speed of the program.
    TYPE typename = PACKED ARRAY
[index-type-list] OF element-type;
    VAR varname-list = PACKED ARRAY
[index-type-list] OF element-type;

Example:

VAR FOURSPACE : PACKED ARRAY[0..10,
    0..10,0..10,0..10] OF INTEGER;

    Elements in arrays are referenced
by placing the index expression(s)
between square brackets associated
with the array name.
    array-name[index-expression-list]

Examples:

A[1,5]   FOURSPACE[X,Y,Z,T]   LIST[N+1]

Record Types:
```
     TYPE typename = RECORD field-list
END;
     VAR varname-list= RECORD field-
list END;
```

Examples

```
TYPE COMPLEX = RECORD REAL,IMAGINARY:
        REAL END;
TYPE CUSTOMER =¯RECORD
      NAME,STREET:ARRAY[1..30] OF
          CHAR:
      CITY:ARRAY[1..20] OF CHAR;
      STATE:ARRAY[1..2] OF CHAR;
      ZIP: 0..99999
   END;
```

In addition to fixed format rec-
ords, a case construct can be added to
the record description to permit var-
iable structure.

Example:

```
TYPE
  THIRTYCHARS=PACKED ARRAY[1..30] OF
     CHAR;
  EMPLOYER=(GOVT,PRIV,SELF,OTHE);
  TWENTYCHARS=PACKED ARRAY[1..20] OF
     CHAR;
  TWOCHARS=PACKED ARRAY[1..2] OF CHAR;
  EMPLOYMENT=RECORD
     NAME:THIRTYCHARS;
     ADDRESS:RECORD
        STREET:THIRTYCHARS;
        CITY  :TWENTYCHARS;
        STATE :TWOCHARS;
        ZIP   :0..99999
     END;
     SEX:(MALE,FEMALE);
     SOCSEC:INTEGER;
     CASE TIMEEMPLOYED : EMPLOYER OF
        GOVT:(YEARS :INTEGER;
              BRANCH:TWENTYCHARS);
        PRIV:(YEARS :INTEGER);
        SELF:( );
        OTHE:(DESCRIPTION:THIRTYCHARS)
  END; (*EMPLOYMENT*)
```

There are two different ways to refer-
ence fields in records. The first is a
format for writing the variable name,
the other is a statement which selects
a particular group of records and
allows reference to fields directly,
within the confines of the statement.

    recordname.fieldname

Examples:

EMPL.NAME   CUST.ZIP   EMPL.ADDRESS.STATE

The WITH statement restricts the
program to only those records which
it specifies. This permits the pro-
grammer to directly reference fields
within those records. Typically, this
statement is used in conjunction with
a compound statement which performs
all of the desired functions on the
restricted record set.
     WITH recordname-list DO statement

Example:

```
WITH EMPL,COMP DO BEGIN
   A:= NAME;
   B:= ADDRESS;
   C:= REAL;
   D:= IMAGINARY
END
```

Set Types:
```
     TYPE typename = SET OF base-type;
     VAR varname-list = SET OF base-type;
```

Examples:

```
TYPE LETTER = SET OF 'A'..'Z';
VAR DIGITS : SET OF 0..9;
VAR SIZE : SET OF (SMALL,MEDIUM,LARGE);
```

Note that the base-type must be a
scalar or subrange type.

Set Operations:
```
   +    Union
   *    Intersection
   -    Difference
   =    Set equality
   < >  Set inequality
   <= >= Set inclusion
   IN   Left operand is a scalar, right
        operand is a set.  Evaluates to
        TRUE if the scalar is an element
        of the set.  In other words, if
        the scalar is IN the set.
```

File Types:
```
     TYPE filename = FILE OF type;
     VAR varname-list = FILE OF type;
```

Examples:

```
TYPE DATA = FILE OF INTEGER;
VAR CUSTFILE : FILE OF CUSTOMER;
```

References to files are made through a
set of predefined procedures which are
listed below. When a file is declared,
(All I/O files must also be declared
in the program heading, as was noted
earlier.) a buffer with the same name,
followed by an ↑ symbol is created.
This buffer variable is like a window

on the current position of the file.

Examples:

B:= CUSTFILE↑
CUSTFILE↑ := XYZ

    The standard I/O procedures for
use with files are:

RESET(filename)   Returns the file win-
                     dow to the beginning
                     of the file.
REWRITE(filename)   The file is replaced
                     by an empty file, the
                     window is set to the
                     beginning of the file,
                     and the file becomes
                     writeable.
GET(filename)   Advances the window
                     to the next position
                     in the file.
PUT(filename)   Appends the current
                     value of filename↑ to
                     the file. Will only
                     work if the window is
                     at end-of-file.
                     filename↑ becomes
                     undefined after a PUT.
EOF(filename)   Evaluates to TRUE if
                     the window is at end-
                     of-file.

    See the section on input and output
in Pascal for more information on the
following:

READ(filename, varname-list)
WRITE(filename, varname-list)
READLN(filename, varname-list)
WRITELN(filename, varname-list)

Pointer Types:
    TYPE typename = ↑type;
    VAR varname-list : ↑type;

Examples: (A linked list)

TYPE LINK = ↑PART
    .
    .
    .
    PART = RECORD
        .
        .
        .
            NEXT:LINK;
        .
        .
        END;

    Pointer variables are only proto-
typically defined by their definition.

Actual storage must be allocated for
them, at run time, by the standard pro-
cedure NEW(pointer-variable). For
record types with variable field lists,
NEW(pointer-variable, case-tag, case-
tag,...) is used, where each case-tag
specified must be listed in the same
order as in the record description.

Conclusions:

    Pascal is a relatively new and
powerful general purpose programming
language. It is also one of the first
languages to employ many of the prin-
ciples of structured programming.
    As a result of this, programs
written in Pascal are usually more
straightforward and considerably more
readable than those written in most
other contemporary languages.
    Since its introduction, Pascal
has seen an amazing rise in popularity
throughout the world. This fact is
well evidenced by the number of col-
leges and universities whose computer
science departments have switched
their emphasis from FORTRAN or BASIC
to Pascal, in the past few years.
Educators are discovering that Pascal
is an excellent introductory language,
since it is not only easy to learn,
but also teaches good programming
habits right from the beginning.
    Pascal is certainly not the utopia
of programming languages -- it is far
from perfect. However, it provides
a significant improvement, in general
purpose computing, over most of those
older languages listed earlier, thus
it would seem to be the next logical
rung on an endless ladder reaching
towards a perfect language.

References:

Jensen, Kathleen and Wirth, Niklaus
Pascal User Manual and Report New
York: Springer-Verlag, 1974.

Schneider, G. Michael; Weingart, Steven
W. and Perlman, David M. An Intro-
duction to Programming and Problem
Solving with Pascal New York: John
Wiley & Sons, 1977.

Ralston, Anthony (ed.) and Meek,
Chester L. (asst. ed.) Encyclopedia of
Computer Science New York: Petrocelli
/Charter, 1976.

Mickel, Andy (ed.) <u>Pascal</u> <u>News</u> #9/10
Minneapolis, MN: Pascal User's Group,
1977.

Conway, Richard; Gries, David and
Zimmerman E.C. <u>A</u> <u>Primer</u> <u>on</u> <u>Pascal</u>
Cambridge, Mass.: Winthrop, 1976.

Bowles, Kenneth L. <u>Problem</u> <u>Solving</u>
<u>Using</u> <u>Pascal</u> New York: Springer-
Verlag, 1977.

Appendix 1:   List of Machines with
              Known Implementations
              of Pascal

     The greatest proliferation of
Pascal implementations appears to
have occurred on Burroughs B6700,
CDC-6000/Cyber series, IBM 360/370
series and DEC PDP-11 series
machines. Pascal is available in
forms which will run on most config-
urations of these machines.
     The following is a list of
machines for which an implementation
of Pascal is known to exist. (*)
indicates that the implementation
was still under development at the
time of this writing. For detailed
information, see <u>Pascal</u> <u>News</u> #9/10
September, 1977.

Amdahl 470
Burroughs B1700, B3700, B4700, B5700,
         B6700, B7700
Control Data Corp. Cyber 18 and 2550,
         3200(*), 3300, 3600, 6000
         series, Cyber 70 series,
         Cyber 170 series, Omega 480,
         Star 100
CII IRIS 50, IRIS 80, 10070
Computer Automation LSI-2
Cray-1
Data General Eclipse, Nova
Digital Equipment Corp. PDP-8 series,
         PDP-10 series, PDP-11 series
Dietz MINCAL 621
Foxboro Fox-1
Fujitsu FACOM 230
Harris/4
Hewlett-Packard HP-21MX, HP-2100,
         HP-3000(*)
Hitachi Hitac 8800/8700
Honeywell H316, 6000
IBM Series 1(*), 360/370 series, 1130
ICL 1900 MK2, 2970, 2980
Intel Intellec 8080A
Interdata 7/16, 7/32, 8/32
Itel AS/4, AS/5
Kardios Duo 70

Mitsubishi MELCOM 7700
MITS Altair 680b
MOS Technology 6502 (*)
Motorola 6800 (*)
Nanodata QM-1
NCR Century 200 (*)
Norsk Data NORD-10
Prime P-400
Sems T1600
Siemens 330, 4004, 7000
Solar 16/05/40/65
Telefunken TR-440
Terak 8510A
Texas Instruments ASC, 9900/4
Univac 90/30(*), 90/70, 1100
Varian V-70
Xerox Sigma 6, Sigma 7, Sigma 9
Zilog Z-80 (*)

Appendix 2:   Pascal User's Group

     The Pascal User's Group is in its
third year, and already boasts a world
wide membership with branch offices
in Europe and Australia. The Group
is based at the University of
Minnesota, under the direction of Andy
Mickel. The main function of the
User's Group is to promote the use of
Pascal, by providing an open forum for
members, in the form of the quarterly
published <u>Pascal</u> <u>News</u>. The content of
Pascal News is determined by the motto
"All the news that fits, we print."
     Membership/subscription dues are
$4.00 per academic year. To join, or
get more information (Your letters may
or may not be answered -- these people
are extremely busy. If you just want
information, it's best to just join,
and then send in a letter for public-
ation.) write to:

Pascal User's Group, c/o Andy Mickel
University Computer Center: 227 EX
208 SE Union Street
University of Minnesota
Minneapolis, MN 55455 USA

     If you are joining, you should
send along $4.00, your name, address,
phone number, type(s) of computers
you are using (Especially if you have
a Pascal implementation on any of
them), and be sure to date your letter.
     Better yet, if you know someone
who already gets Pascal News, just copy
the All Purpose Coupon from one of the
issues, and send that in.

HEX - OCTAL - DECIMAL - BINARY

CONVERSION CHARTS

4 - COLOR WALL SIZE CHARTS (2' X 4')

4 BASES ON EACH CHART
2 ORIENTATIONS TO CHOOSE FROM

OCTAL

ENTER WITH OCTAL
CONVERT TO HEX
DECIMAL OR BINARY

HEX

ENTER WITH HEX
CONVERT TO OCTAL
DECIMAL OR BINARY

Scan either chart for decimal to hex/octal/binary
each chart covers ∅ - 255 (one byte)

Regular $6.00 each

Special faire price   both for $8.00

To order send check or money order to:

ZETA SYSTEMS CANADA LTD.
2547 Heather Street
Vancouver, B.C.   V5Z 3J2

Charts will be sent first class, in a four (4) ply
mailing tube.

The Computer Magazine
You Can Read!

Sol-20. First it was THE SMALL COMPUTER. Now, it's THE SMALL COMPUTER SYSTEM.

A year ago, we introduced the Sol-20. It wasn't the first small computer. It was the first complete small computer with everything needed to get it up and on the air as it came from the factory. The keyboard, interfaces, extra memory, factory backup, and service notes were all there.

The results are in: Sol-20 is now the number one small computer in the world. Sols aren't the cheapest, just the most valuable.

We originally designed the Sol-20 as the heart of a complete computer system. So now to solve the problems of science, engineering, education, business management and control and manufacturing, we offer fixed price Sol systems in either kit or fully tested and assembled form. We offer language flexibility, Extended BASIC, Assembler, PILOT* and FORTRAN.* We

offer Helios II/PTDOS, an extraordinarily capable disk operating system. And remember, though we call these small or personal computer systems, they have more power per dollar than anything ever offered. They provide performance fully comparable and often superior to mini-computer systems costing tens of thousands of dollars more.

What you get. What it costs.

Typical systems include Sol System I priced at $1600 in kit form, $2095 fully assembled and tested. Included are a Sol-20/8 with SOLOS personality module storing essential system software, an 8192 word memory, a 12″ TV/video monitor, and a cassette recorder with BASIC tape.

Sol System II has the same equipment with a larger capacity 16,384 word memory. It sells for $1825 in kit form; $2250 fully assembled.

For even more demanding tasks, Sol System III features Sol-20/16 with SOLOS, 32,768 words of memory, the video monitor and the dual drive Helios II Disk Memory System with the PTDOS disk operating system and Extended DISK BASIC Diskette. Price, $5795 fully assembled and tested.

More information.

For the most recent literature and a demonstration, see your dealer listed below. Or if more convenient, contact us directly. Please address Processor Technology Corporation, Box O, 7100 Johnson Industrial Drive, Pleasanton, CA 94566. Phone (415) 829-2600.

*Available soon.

Processor Technology

Since 1972 we've been a place where people and computers get together.

Do you know anyone who wants to learn how computers work? Or how to program? Without masses of technical jargon? Then send them to *People's Computers*! We're a bimonthly magazine devoted to demystifying computers. Our pages aren't padded with ads—they're packed with articles, tutorials, interviews, listings, games, reviews and letters. *People's Computers* covers:

Computerese - basic principles about how computers work; tutorials on how to program.

Education - ready-to-use programs; tips on how to write computer-assisted educational materials; reports on educational uses of computers.

Blue Sky - Tiny BASIC and Tiny PILOT were born in our pages; there's now talk of robots and . . .

People & Computers - examination of the growing impact of computers on our lives and society.

Fun & Fantasy - computer games; whimsical odds and ends; and our swashbuckling hero—FORTRAN Man!

Some Comments about *People's Computers:*

"One of the first, if not *the* first, of the people-oriented computer publications . . . At $8 a year, it is a good bargain. You'll like it." *Kilobaud*

"The editorial flavor is . . . intended to be readable and enjoyable for the neophyte. Our resident non-computer people at *Byte* grabbed the first issue so quickly that it became difficult to find a copy . . . Should be sampled to be believed." *Byte*

"*People's Computers* has been plugging away for years as a force for good in the community". *Personal Computing*

"For the novice as well as the experienced computer user . . . an informal style, many useful annotations . . . a sounding board for novel ideas." *Computer Notes*

"Here's a first rate magazine for people who want to know about minicomputers and microcomputers . . . Powerful, low-cost technology is taking off — keep up with it by reading *People's Computers*". *The Workbook*

people's computers

Please start my one-year subscription (six issues) to *People's Computers* and bill me for just $8.

NAME _____

ADDRESS _____

CITY/STATE _____ ZIP _____

Outside the U.S., add $4 for surface postage. Airmail rates on request.

Unconditional Guarantee: If you ever wish to discontinue your subscription *for any reason*, we will refund the complete amount for the remainder of your subscription.

☐ Visa   Card Number _____

☐ Mastercharge   Expiration Date _____

Send this form or a facsimile to: *People's Computers*, Dept 57, 1263 El Camino Real, Box E, Menlo Park CA 94025