Computer Using Educators—Position Paper On Commercial Software Pricing

The 6502 Resource Magazine
PET • Apple • Atari • OSI • KIM • SYM • AIM

Mapping And Modifying Unknown Machine Language

COMPUTE!

$2.50
June, 1981
Issue 13
Vol. 3, No. 6
63379

The Journal For Progressive Computing™

Apple II High Resolution Character Generator

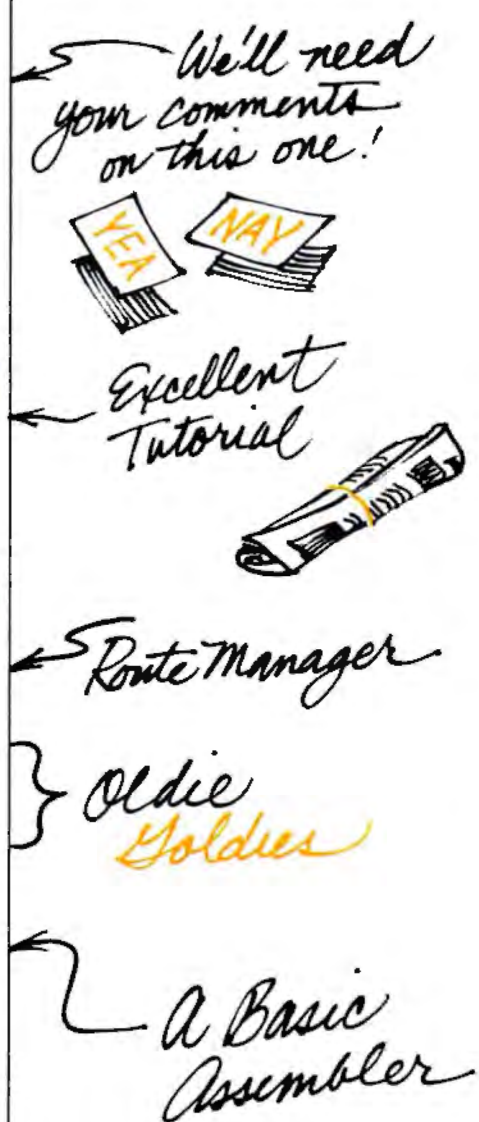
Computerized (Talking) Greeting Cards On Your Atari

Relocation Of BASIC Programs On Your PET/CBM

RAM/ROMs A New Style Of Memory?

Table of Contents

June, 1981, Vol. 3, No. 6



COMPUTE! The Journal for Progressive Computing (USPS: 537250) is published 12 times each year by Small System Services, Inc., P.O. Box 5406, Greensboro, NC 27403 USA. Phone: (919) 275-9809. Editorial Offices are located at 200 East Bessemer Ave., Greensboro, NC 27401.

Domestic Subscriptions: 12 issues, $20.00. Send subscription orders or change of address (P.O. Form 3579) to Circulation Dept., **COMPUTE!** Magazine, P.O. Box 5406, Greensboro, NC 27403. Controlled circulation postage paid at Greensboro, NC 27403. Application to mail at controlled circulation rates pending at Hickory, NC 28601. ISSN 0194-357X.

A Small System Services, Inc. Publication

The Editor's notes...

Robert Lock,
Editor/Publisher

A **COMPUTE!** *first: I'm turning most of my space over to Jim this time for his following article. He didn't write it with the expectation it would show up here, but I think it's import is substantial. We've been discussinng software protection/copying for several issues, and these new products will have a direct impact on existing protection methodology.*

A few pages away, you'll find a position paper from CUE — Computer Using Educators — on software protection. We mentioned possibilities of educational licensing arrangements several issues back, and have had interesting input from educators (we'd welcome more ...). We have yet to hear formally from any software vendors; if you're a seller of software, please read the CUE position paper and RAM/ROMs. Send us some feedback. We'd like to maintain an open forum on the subject for several issues and see if we can converge on some common standards/principles.

One last note ... If you recall Richard Mansfield's excellent article on Machine Language for beginners (March, '81), you'll be pleased to know he is joining our full time staff as Assistant Editor. We're very pleased to have him. RCL

RAM/ROMs— A New Style Of Memory?

Jim Butterfield
Toronto, Canada

Two new commercial products may change the way we use ROM sockets. Both Instant ROM from Greenwich Instruments, England and Soft ROM from Canadian Micro Distributors, Canada are based upon the same principles: RAM that plugs into a ROM socket.

A Trip Down Memory Lane

ROM is read-only memory: its information is permanently burned in, often at the factory. It's marvellous stuff, since it gives your computer its style and intelligence by means of the programs embedded in ROM. When you power up, the programs are in place instantly; they are fixed in ROM forever.

RAM stands for **R**andom **A**ccess **M**emory, but it would be better named Read and Write memory. This is where you store information that is created after power on: programs and the data that they use. RAM memory can be easily changed, but when you turn the computer off, RAM loses its information. This type of amnesia is a characteristic of "volatile" memory; information evaporates when the power goes.

Some picky people like to say that ROM isn't memory at all. The way they see it, if you can't store information there, it can't be memory. Most of us like to think of it as a memory whose information was pre-stored by the manufacturer or supplier. It can't learn anything new, but it won't forget anything.

Empty Sockets In The Old Corral...

Many computers such as the PET/CBM have empty sockets intended for future ROM expansion. You may plug in a wide variety of ROMs, many of which are commercial products: Toolkit, Command-O, Visicalc, WordPro and many others. Some of these ROM chips contain the complete system; others contain supporting subroutines and the rest of the system will reside in RAM. A few chips are purely cosmetic: the main program in RAM checks to ensure the ROM is in place but doesn't use it in any functional way.

In addition to commercial products, users are generating their own ROM systems. A special type of ROM called an EPROM can be written by the user using inexpensive equipment; if the user changes his mind or finds a bug in his program, he can erase the chip and re-write it. User clubs have generated their own club ROM systems in this manner.

And now the new products, which I'll dub RAM/ROM, allow ROM systems to be emulated with no special equipment. A user can plug this device into a ROM socket, store information into it, and proceed as if he had a ROM in place.

Program Protection: ROM Rations?

These developments have generated concern on the part of software houses. In the past, they have relied on ROM chips to protect their programs against theft. Now this protection is lessened: EPROMs are not difficult to write, and RAM/ROMs make it a simple job to create a simulated ROM.

Although one must sympathize with these concerns, it's probably high time that ROM protection systems were retired. There are few enough sockets available on a processor board, and the user is often faced with having to permanently give up a socket to one of several software packages; he must choose one and reject all the others ... it's

Subscription Information

(12 Issue Year):
COMPUTE! Circulation Dept.
P.O. Box 5406
Greensboro, NC 27403 USA
U.S. $20.00
Canada $25.00 (U.S. funds)
Europe: Surface Subscription, $25.00 (U.S. funds) if ordered direct, or available in local currency from the following distributors:

United Kingdom
Contact L. P. Enterprises,
8-11 Cambridge House
Cambridge Road
Barking, Essex
England IG1 18NT

Germany,
Switzerland,
Austria,
Contact Ing. W. Hofacker GMBH
8 Munchen 75
Postfach 437
West Germany

Canadian Retail Dealers should contact:
Micron Distributing
409 Queen Street West
Toronto, Ontario M5V 2A5
(416) 361-0609

Advertising Sales

If you're in **Oklahoma, Texas or the Western States, we're now represented by Jules E. Thompson, Inc.** Give them a call for space reservations, contract/insertion information, or questions. You can reach them through the following offices:

Southern California, Arizona, New Mexico
Jules E. Thompson, Inc.
2560 Via Tejon
Palos Verdes Estates, CA 90274
213 378-8361
Jo Ann Sullivan

Northern California, Pacific Northwest, Rocky Mountain States, Texas, Oklahoma
Jules E. Thompson, Inc.
1290 Howard Avenue, #303
Burlingame, CA 94010
In Texas or Oklahoma call our Houston number: 713 731-2605
Elsewhere: 408 354-5553 or 415-348-8222
Phoebe Thompson

If you're in the East, We're now reprsented by **The Gittelman Company.** You can reach them through the following offices:

New England, New York State
The Gittelman Company
Statler Office Building
Suite 582
20 Providence Street
Boston, MA 02110
617 451-0822
Joan Donahue

New York City Metro Area, Mid-Atlantic and Southeastern States:
Local Numbers:
New York 212-567-6717
Atlanta 404 523-1252
The Gittleman Company
Summit Office Centre
7266 Summit Avenue
Fort Washington, PA 19034
215 646-5700
Doug Johnson

If you're in the **Midwest, we're now represented by GB & Associates.** You can reach them through the office of:

GB & Associates
P.O. Box 335
Libertyville, IL 60048
312 362-1821
Gordon Benson

Address all advertising materials to:
COMPUTE
200 East Bessemer Avenue
Greensboro, NC 27401 USA

Mailing address: COMPUTE!
Post Office Box 5406
Greensboro, NC 27403 USA

Telephone: (919) 275-9809

impractical to keep switching ROMs in and out of the processor board. Even ROM switches which allow the user to select one of several ROMs eventually reach their limits. Several dozen proprietary programs fighting each other for occupancy of a very few sockets won't lead to a healthy and prosperous software industry. There must be a better way: one that doesn't limit the user.

So while the software houses may groan at the loss of one of their favored protection mechanisms, they may find that in the long run they are being encouraged to use a sounder mechanism. My own biased opinion is that there's no better protection than sound documentation, a good warranty system and continuing customer support.

The user, in turn, is likely to welcome the opportunity to acquire much more software without congesting his machine. He will also be enabled to roll his own software, and to adapt commercial ROMs to his own specific needs.

When would you use "regular" memory for your programs, and when would you put them into ROM space? In general, RAM/ROM systems will tend to hold semi-permanent machine language programs; programs that support or supplement Basic or that aid a separate process such as communications input/output. Basic isn't likely to go up there. Temporary machine language programs will use Basic memory so that they will be replaced when the next program is loaded.

The Commercial Boards

It's worth mentioning that fitting RAM into ROM space isn't an entirely new concept. MTU's Visible Memory, for example, fits into PET's ROM memory space, and can be used as RAM/ROM. The two products discussed here are novel in that they plug directly into the ROM socket.

The units described fit the 2532 type socket which provides 4K of memory on the PET. Other units or strappings are available to match various socket configurations.

ROM sockets don't have the proper pin connections to allow memory to be written ... after all, a ROM is intended only for reading. To overcome this, both products are furnished with mechanisms that allow the computer's "write line" to be accessed.

Instant ROM

Instant ROM is produced by Greenwich Instruments Ltd., 22 Bardsley Lane, Greenwich, London, England SE10 9RF. Price in the UK is 57 pounds.

The device is nicely packaged in an enclosure and very compact. It's not hard to imagine carrying one of these around in your pocket. In fact, there's a carrier provided to allow you to do this.

The striking thing about Instant ROM is that it's non-volatile: the memory contents are not destroyed when you turn the power off! How is this posible? The unit contains its own tiny batteries. When external power is lost the batteries take over, and they can keep the ultra low current CMOS memory going for up to three months. When you turn on your computer again, the batteries will be recharged; as little as one hour's use per week will keep the batteries fully charged.

The non-volatility makes Instant ROM a useful device in a wide variety of applications. You don't need to reload memory every time you power up your computer. It's even possible to rewrite your computer's operating system to run to your own specifications. Would you like to see your computer start up with a message like THE JOE DOAKES PERSONAL BASIC SYSTEM? You can do it by replacing the system ROM with an Instant ROM.

The computer's "write line" is picked up on a wire-wrap connector mounted on one side of the package. A special connector is furnished for the PET. If memory is "permanent" and not subject to change, this connector need not be fitted.

Soft ROM

Soft ROM is distributed by Canadian Micro Distributors Ltd., 365 Main Street, Milton, Ontario L9T 1P7, Canada. Price is $129.

The device is a circuit board with a set of ROM pins attached to one edge. The board plugs in vertically. Soft ROM contains A ROM socket on the board; the user can plug in a ROM here and then select whether he wants to use the on-board ROM or the RAM/ROM.

Memory is volatile, so that the user will need to reload the contents of RAM/ROM after power has been off. Soft ROM is suitable for systems which load each package as it's needed.

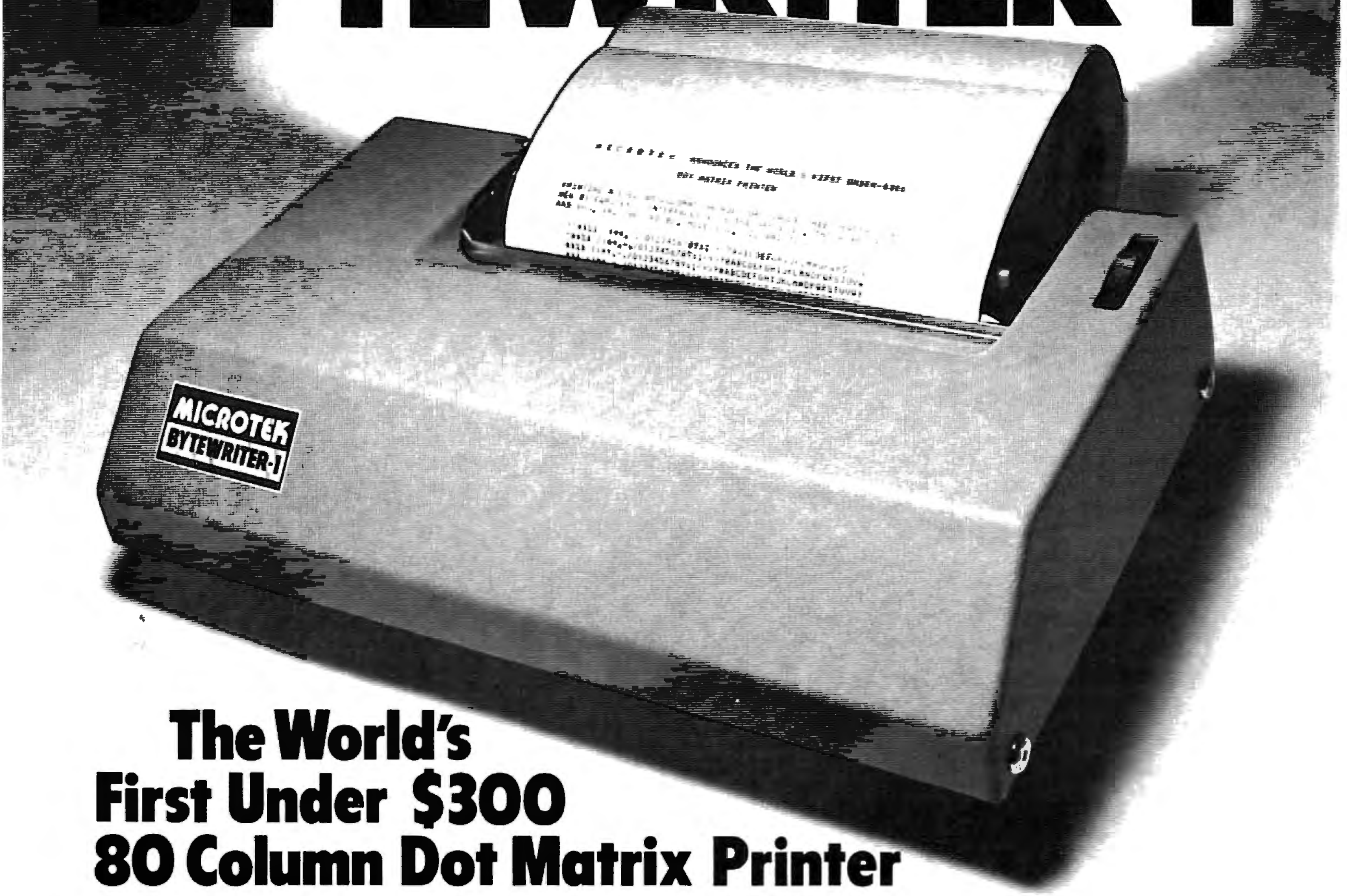
A connector is provided to pick up the computer's write line. Additionally, a three position switch is provided which extends outside the computer and can be attached to the case. The switch positions are marked: Read/Write; Read Only; Write Only. Read/Write is normal RAM configuration; you can read it or write it. Read Only makes the unit look like true ROM; it cannot be written. Write Only seems puzzling, though: what on earth can you use a Write Only memory for? Answer: it can be used to write into the board even though the computer's write line is not connected; this may be useful in some circumstances.

Summary

It's a new game. Some users will never need RAM/ROMs ... they will plug in the ROMs that they need, if any, and that will be it. Others will appreciate the versatility of the new devices. Each one has its own characteristics and price.

Whether you decide to use one or not, the existence of these new devices will change the way we perceive our computers. We are no longer socket slaves ... these devices will give our programs extra elbow ROM ... ©

BYTEWRITER-1



The World's First Under $300 80 Column Dot Matrix Printer

PRICE BREAKTHROUGH FROM MICROTEK

Bytewriter-1, the $299 printer that has taken the industry by storm! A product made possible through cost-effective design, efficient manufacturing operations, a proven low-cost print mechanism and the simplest possible control electronics. We have done extensive testing with all three models of the TRS-80, the Apple II, and the Atari 400 and 800. That is why we can guarantee you won't have any interface problems with the Bytewriter-1.

TRS-80 is a trademark of Radio Shack, Div. of Tandy Corp.
Apple II is a trademark of Apple Computer, Inc.
Atari 400 & 800 are trademarks of Atari, Inc.
Bytewriter-1 is a trademark of Microtek, Inc.

Microtek, Inc.
9514 Chesapeake Drive
San Diego, CA 92123
Tel. 714-278-0633
Outside Calif. call
toll free: 800-854-1081
TWX. 910-335-1269

MICROTEK inc.

BYTEWRITER-1 SPECIFICATIONS

Printing Technology:	7-wire bi-directional impact wire matrix
Printing Speed:	60 lines per minute (80 characters per second) continuous
Character Set:	96 character ASCII (upper and lower case)
Character Size:	10 characters per inch (80 columns per line) plus expanded printing
Paper:	Friction feed (synchronous), accepts single sheet and roll paper up to 9½ inches maximum width. Prints original plus 3 copies.

By the way, our replacement print-head costs less than $30 too!
Call or write today for more information.

Computers And Society

David D. Thornburg
Los Altos, CA

Every year our neighborhood throws a Fourth of July party, with a pot luck dinner and plenty of entertainment for the kids. Over the years this party has become the *one* annual event at which most of the people in our neighborhood get to visit with each other.

Like many communities in the heart of Silicon Valley, we have a high proportion of residents who either work with or who design computers and computer-based systems. Working for companies like Xerox, Atari, Hewlett Packard, or Lockheed, these people have a solid grasp of the utility, power, *and* limitations of computer technology.

Other residents in our block cover a wide range of professions — contractors, landscape architects, medical doctors, engineers, teachers, *etc.* These people generally have little understanding of the effort needed to make computers do truly useful things.

And it so happens, every Fourth of July, that sometime after dinner, while the kids are getting the fireworks ready, that a doctor (or contractor, or other member of this second group) will say:

"Boy, I don't know about these personal computers. I tried to use one in my business and it was *worthless*. All I use it for now is games — I'm better off keeping my books by hand."

"Just what happened?", one of the "computer" people will ask.

"Well", says the doctor, "I had been reading all this stuff about the microcomputer revolution, and those magazine ads really made an appealing pitch. For a thousand bucks I could buy one of these micros, and I was supposed to be able to write programs which would simplify my business."

"I have a small practice — simple billing procedures, and just a couple of employees — so I figured that what the heck, for a grand or so I could bring my office into the twentieth century. Now I didn't have a whole lot of money to spend, so I went down to the local computer store and picked up a brand new __________ (fill in with the name of your favorite micro). While I was planning on spending only a thousand bucks, the dealer said that my system was going to be pretty useless without a disk drive and a printer. By the time I got home, I had dropped $3,000 in that place."

"Next, I spent a couple of weeks trying to write a simple billing program in BASIC. After I saw how much time I was pumping into this project, went out and got a consultant to help me. So far I've paid him as much as I paid for the whole system, and I'm still not happy."

During the recitation of this tale of woe, those of us in the computer industry nod reassuringly, and then try to show our disgruntled friend that his problem is quite common, and that it was avoidable.

The universality of this sad tale has taught me that a business person who buys a computer system without first knowing what he or she is going to do with it could save a lot of grief by flushing the money down the drain instead. And yet, computers are being sold to many thousands of people who have no idea what they are getting into.

... a business person who buys a computer system without first knowing what he or she is going to do with it could save a lot of grief by flushing the money down the toilet instead.

When confronted by a professional who has had this problem, my advice has run along one of two channels. First, subscribe to a local time-sharing service which provides all the needed accounting and bookkeeping programs. Even if it means changing one's bookkeeping practices slightly, there is a lot of merit in using a program which is being used by hundreds of other similar users every week. Since low-cost ($5.00/hour) time sharing services are available in our area, once the "mispurchased" computer is converted to a terminal emulator, we are all set.

The second approach is a bit more time consuming, since it involves looking at a whole bunch of "canned" business programs which are available through local computer dealers. Once again, local changes in record keeping might be needed, but this is a small price to pay for the benefit of having a working computer system.

The third alternative is to recognize that the computer was purchased prematurely and that, perhaps, the business just doesn't warrant a high-tech sledgehammer to drive a thumb tack. In this case, the user should stock up on some good games and enjoy the purchase at home.

I have often wished I had a book to give to my friends who are contemplating the purchase of a micro for small business applications. Until now, I haven't seen anything worth recommending. The books I had all were too technical, or too folksy and full of errors. And still, it is obvious that a non-technical business-oriented book on this topic is genuinely needed.

Fortunately my search is over. It was a most pleasant surprise for me to read the recently published **Business System Buyer's Guide** by Adam Osborne and Steven Cook (Osborne/McGraw Hill, $7.95, paper). This well written book is recommended reading for all small business owners who are contemplating the purchase of a computer system.

The authors make the point that such people are not buying a computer, they are buying a solution to the data keeping and processing problems of their business. Accordingly, the first task Osborne and Cook set out for their readers is to analyze their business practices from the perspective of information flow:

How big are the files to be maintained? What are the file requirements? 30,000 characters? More? Less? What are the input requirements? How many hours a day will the keyboard be used? Will you enter much numeric data? What are the output requirements? Is a printer needed? Is letter quality printing required, or is dot matrix OK? How many pages of printout will be generated per day?

By answering these basic questions, the reader is well on the way to defining his or her system requirements. By performing this task before looking at computer systems, the reader is less likely to be swayed by the kinds of hype which got our doctor friend in so much trouble.

Proceeding through an accurate but simple view of the components which make up a computer system, the authors proceed to show how the answers to the original questions can be used to help select a system. By this time, the reader is in contact with a vendor or two, and has a clear idea of what is available.

The authors encourage readers to look at mini- as well as microcomputers if the business applications are complex enough. The high cost of custom software is mentioned several times, and the reader is encouraged to rely primarily on off-the-shelf packages.

The book is rounded out with a surpirsingly accurate (if somewhat incomplete) list of major software packages and hardware systems to aid the reader in making a selection. My only complaint with their list of computers is that some of the smaller companies with excellent products (such as Exidy) were left out. Beyond that, I think that Osborne products received a little more attention than they might have if the book had been written by someone else. I may be too critical here, however, since the authors' treatment of other computers is quite fair, and their description of the Osborne I computer was quite succinct. Nonetheless, all the other computers mentioned had established themselves in the marketplace well before the book was written. To describe the features of the Osborne computer in a book published before any of these computers had been shipped is, at best, risky business.

All in all, **Business System Buyer's Guide** is an excellent book, and one which I plan on getting to my friends who are contemplating the purchase of a small computer for their business.

Who knows? Maybe *this* Fourth of July we can compare Star Raiders™ scores instead! ©

Brief Review:

The Osborne Microprocessor Handbooks

Jim Butterfield
Toronto, Canada

The **Osborne 4 & 8 Bit Microprocessor Handbook**, together with the **Osborne 16 Bit Microprocessor Handbook**, constitutes a formidable source of information on current microprocessors and support devices.

The books are reference in nature, rather than tutorial. They are handy for quickly looking up some characteristic of the Z80 or the 6809, for example: but they wouldn't be particularly well suited for learning how to program a chip. They contain more than just reference sheets, however. Each chip is described in a narrative style giving its major characteristics.

One of the things I enjoy about Adam Osborne's writing is his willingness to offer opinions along with hard facts. This might seem out of place in a book that is primarily reference in nature; but in fact the editorial comments help place the various chips in better perspective. Users may find their understanding is helped by sentences such as: "When it first appeared, the F8 was discussed as an off-beat product with a strange set of chips and a ridiculous instruction set" ... "All other 8080A manufacturers (other than Intel, Siemens and AMD) are unauthorized ... some differences exist." ... "The TMS 9900 product line has for some time been one of the enigmas of the microprocessor industry." You don't have to agree with these comments, but the fact that they are there will often help the reader to understand how a given chip is accepted in the industry.

These two massive volumes (The 4 & 8 Handbook alone has more than 1200 pages) are offshoots of the popular Osborne series, **An Introduction to Microcomputers**, and are intended to replace Volume 2 — **Some Real Microprocessors**. They haven't completely broken away from their former series membership. There are numerous references to Volume 1 and Volume 3 which might puzzle the reader who has bought the Handbooks as separate volumes. In particular, there are references to a hypothetical microcomputer; this turns out to be from Chapter 7 of Volume 1 of the Introduction series, entitled, "An Instruction Set".

The problem with the original Volume 2 was its packaging: it was a loose-leaf volume, intended to be kept up to date with periodic update inserts. The logistics of this proved difficult. As a result, we now have two "fixed" volumes — to update them the user will buy newer editions as they appear. I suspect that most of us will find this more familiar, convenient, and portable.

The titles of the two volumes are slightly misleading. 4 & 8 Bit should have been named, "under 16 bits", since the twelve-bit IM6100 is covered. The 16-bit volume also covers the 2900 chip slice (or bit slice) family, which allows you to have however many bits you would like.

A sample program called a "benchmark" is given for all microprocessors. It's quite useful for gaining an understanding of the style of a micro. Although it's not strictly fair to use a single program as a measure of a processor's goodness, users will at least get a feel for the chip. Oddly enough, the authors themselves rather harshly criticize the value of such a program: "Benchmark programs are misleading, irrelevant and worthless ... we will demonstrate the capriciousness of benchmark programs via the following ... ". Gee ... I found it really useful to be able to look at a piece of sample programming.

COMPUTE! readers have a special interest in the 6502, of course, and are likely to use the 6502 material as their own "benchmark" of how good the book is. They may be disappointed. Much of the 6500 introduction describes the chip comparatively: how it relates to the 6800 and Z80. The dazzling speed of the 6502 — one of its main advantages — is ignored; no instruction timings are given. And worst of all, the instruction set is incomplete: the ROR (Rotate Right) instruction is completely missing. The first chips didn't have this instruction; but all 6502s manufactured since 1976 have ROR.

Summary

The Handbooks are a comprehensive pair of books covering a lot of information on a lot of microprocessors. They don't cover all micros: I missed the trusty old 8008, for example. Chip detail is extensive but not exhaustive, as can be seen from the omitted ROR and missing timing information for the 6502. The books contain opinion as well as fact, but I welcome that: it adds perspective.

I can't completely agree with the statement on the back covers: "This is the one source for complete, objective, and accurate information on 4 and 8-bit/16 bit microprocessors." Even so, they are a good set of books. If you are interested in information on a broad range of microprocessors, you'll do well to have these available.

[**Osborne 4 & 8 Bit Microprocessor Handbook/Osborne 16-Bit Microprocessor Handbook**, by Adam Osborne and Gerry Kane; Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley, California] ©

SOFT ROM

- 4096 BYTES OF SOFT ROM
- STORE MACHINE CODE SOFTWARE BEYOND THE BOUNDARIES OF BASIC
- WRITE PROTECT RAM WITH A FLIP OF A SWITCH
- STORE VARIABLES OR INDICES OUTSIDE OF BASIC
- SOLVE THE CONFLICTING ROM PROBLEM BY SOFT-LOADING THE APPROPRIATE ROM IMAGES

The SOFT ROM is compatible with any large keyboard PET/CBM or similar 2532 EPROM systems. It may be placed in any ROM socket to give the user room for machine code. If the SOFT ROM is placed in an occupied ROM socket, the user can transfer the PET/CBM ROM into the on-board ROM socket and select between ROM and RAM to manipulate the Commodore operating system.

Since the SOFT ROM places write protectable RAM into any of the computer's ROM sockets, it is ideally suited to use as a development tool to test ROM or EPROM based software systems before they are burned in.

Examples of software presently available for the SOFT ROM includes BASIC AID, UNIVERSAL WEDGE, SUPERMON, EXTRAMON, USER PORT PRINTER (Centronics parallel) and a buffered BACKGROUND PRINTER routine.

Installation is a simple plug-in into any available ROM socket.

$129.00

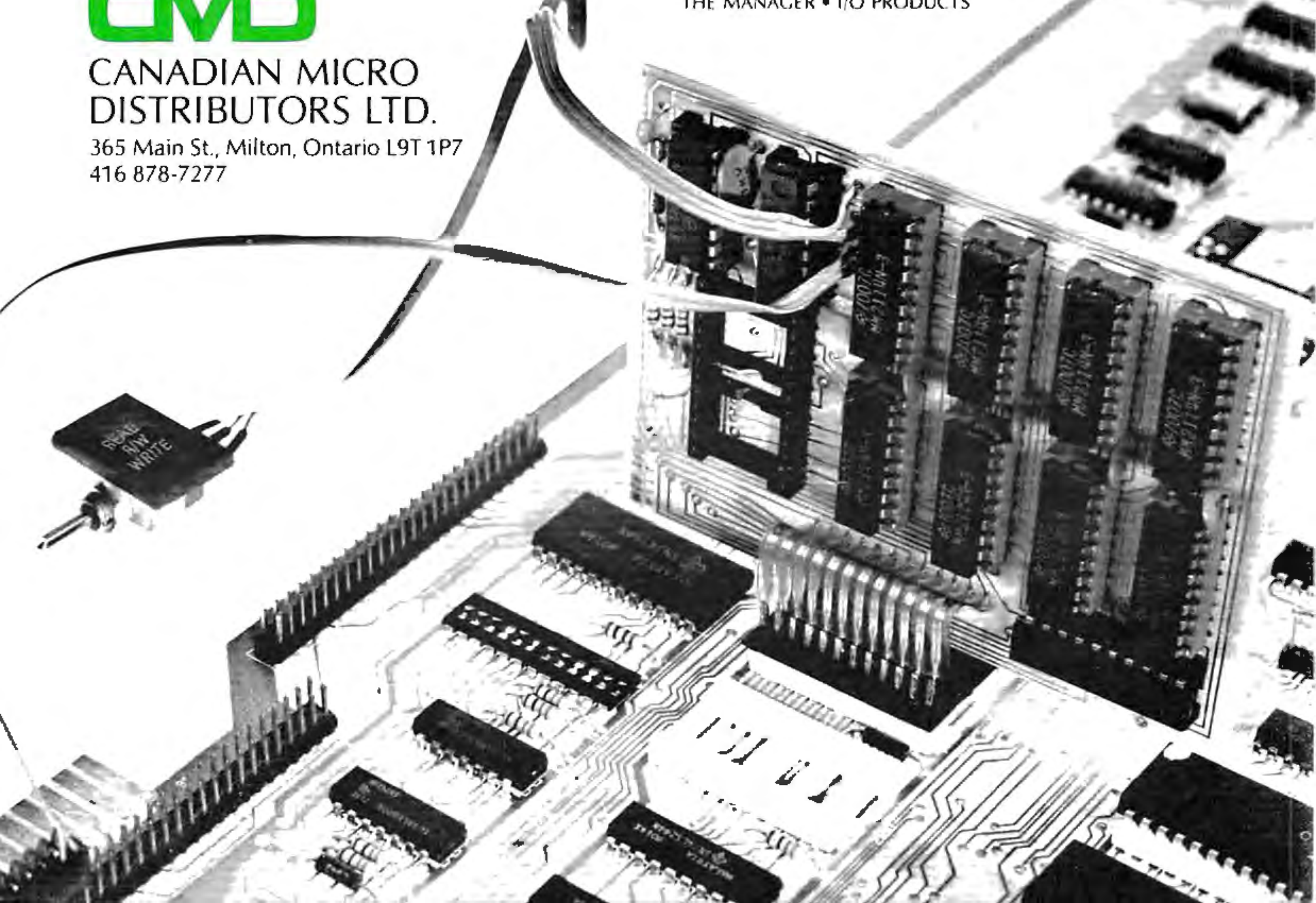
FROM

CMD

CANADIAN MICRO DISTRIBUTORS LTD.

365 Main St., Milton, Ontario L9T 1P7
416 878-7277

DISTRIBUTORS OF:
MUPET • DOUBLE-MUPET • SPOOLER
THE MANAGER • I/O PRODUCTS



Computer Using Educators

Position Paper On Commercial Software Pricing Policies

LeRoy Finkel
Menlo Park, CA

I. Problem:

Data on magnetic disks and cassettes are subject to damage from student mishandling and because of machine malfunction. It is essential that teachers have backup copies of disks and cassettes at their school sites so that classroom activity can continue uninterrupted in the event of such data loss.

Recent entrants to the computer software business are charging substantial prices for their programs, but many do not allow copies to be made for backup, or for use by other teachers in the same schools. Some firms allow additional copies to be purchased at a discounted price, while others issue a "license" to make X number of copies, or to use in Y classrooms, or on Z number of computers or terminals. Still other companies allow unlimited copies for classroom use at "the school" or will issue a license to an entire school district. There is no consistency in policies. If the restrictive policies become the accepted standard, they will produce insurmountable difficulties for schools and will inhibit the increased and productive use of computers in education.

These restrictions can lead to either very expensive educational programs, to underutilized computers, to unsold software, or to casually ignored license agreements. Our desire to minimize the occurence of each of these events leads to this position paper. We hope to encourage a dialog between the educational community and software producers.

II. Position:

At the current time, CUE (Computer Using Educators) encourages schools and educators NOT to purchase computer software from commercial sources who prohibit the purchaser from making free backup copies or who fail to provide reasonable arrangements for the use of the software on all computers operated at the school site.

III. Background:

The computer software industry is in its infancy.

Policies are in the making and the legality of software copyrights is being decided in the courts. Large firms, who are new to education or new to computers, are entering the software market. They have made substantial commitments to software development and it is quite natural for these firms to seek to protect their investment.

CUE understands the position of software distributors in this case. We realize that unless proper compensation can be derived from distributing software, this all important source of educational materials could dry up, to the detriment of all.

The early sellers of software warned users not to make copies for resale or profit. A further implied warning was not to impose on the sale of the materials (in other words, don't give away copies to others who might otherwise purchase the items).

CUE proposes a policy based on the school unit...

Then came Personal Software Inc., with their VisiCalc software for Apple computers. Apple VisiCalc contains locks that make it impossible to copy the program. For $150, the buyer gets one non-copyable disk that is guaranteed against defects, but not against machine malfunction, student abuse, etc. (a recent policy change allows original purchasers to buy one backup disk for $30). VisiCalc has tremendous potential as a classroomm management tool for teachers as well as providing a programming-type skill that we can teach to students. But not at $150 per student (Radio Shack VisiCalc does allow copies for personal use but the policy appears to be changing).

We have now spoken to four major companies who are new entrants to the computer software field. Each seems to be "field testing" non-copy policies similar to those used by Personal Software Inc. to see if they can, indeed, sell software to schools with these restrictions. We believe that resistance on the part of educators will inform these companies that while such a policy may be appropriate for industry or home use, it lacks sensitivity for the needs of education.

In a personal or small business environment, there will be only one or two computer stations being used, normally by competent personnel. In a school, however, with many stations per school, there may be hundreds of inexperienced students using the computers each day. The hazards that our computers and disks are exposed to daily far exceed the hazards that an industry installation might receive in a year. Must we be held financially accountable for the malfunction of our computers and our students? Apparently so, said the salesman from one large company new to this segment of the industry. His response to our query was, "If you crack up your brand new car, you can't take it back to the dealer, can you?". It seems that some of these firms have not given enough thought to the problems that educators have and that the car analogy is not relevant.

His response to our query was, "If you crack up your brand new car, you can't take it back to the dealer, can you?"

IV. Proposed Solution:

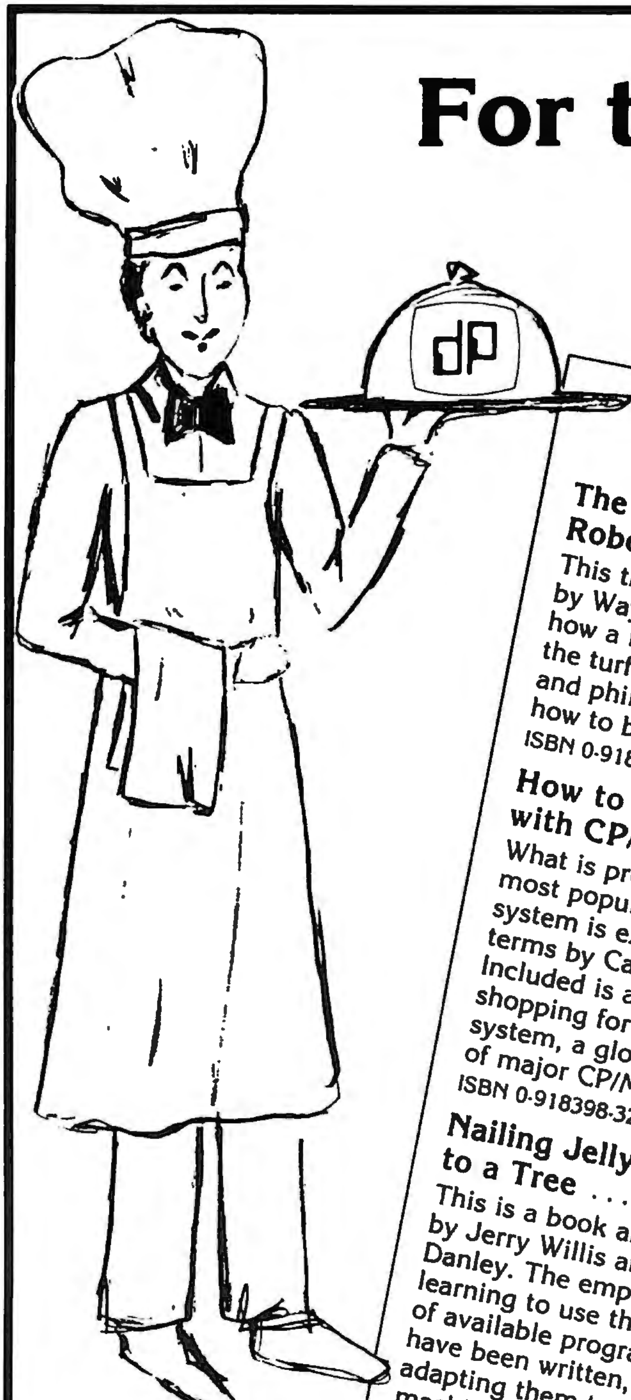
The major publishers and software vendors are seeking to gain a fair return on their investment. They are entitled to that. What they fear is that indiscriminate copying will erode this return.

What kind of policy will meet our requirements and at the same time protect their investment and return? What kind of policy will allow teachers to make enough policies to use on multiple computers in one room, OR multiple rooms in one school, OR multiple teachers in the same school, OR multiple schools in the same school district?

What kind of policy will be acceptable and be respected and honored by educators? The solution must be compatible with school financial policies. In most schools, budgets are allocated on a school-by-school basis. Likewise, Federal and State funding is allocated to districts and then reallocated to individual school sites. Therefore, a software licensing policy that allows unlimited copies at the school site-of-purchase may be a fair policy and compatible with existing school purchasing policies.

CUE proposes a policy based on the school unit, or as one software vendor put it, "for use in the same building or physical complex." Any software is "licensed" to that school, to be copied and used by any and all teachers in that one school regardless of the number of computer stations or type of installation. To be fair to smaller schools and to vendors, we further propose a sliding scale based on school enrollment.

As an organization, CUE will work to insure that educators are informed of the importance of copyrights and licensing, and we will strongly urge our colleagues to respect and abide by such agreements. ©


dP

NEECO

WHY BUY FROM THE BEST?

Service... Support... Software...

CBM™ 8000 SERIES BUSINESS COMPUTERS

The new Commodore 8000 series computers offer a wide screen display to show you up to 80-character lines of information. Text editing and report formatting are faster and easier with the new wide-screen display. The 8000 series also provides a resident Operating System with expanded functional capabilities. You can use BASIC on the 8000 computers in both interactive and program modes, with expanded commands and functions for arithmetic, editing, and disk file management. The CBM 8000 series computers are ideally suited for the computing needs of the business marketplace.

SCREEN
2000 character display, organized into twenty-five 80-column lines
64 ASCII, 64 graphic characters
3 x 8 dot matrix characters
Green phosphor screen
Brightness control
Line spacing: 1½ in Text Mode
1 in Graphics Mode

KEYBOARD
73-key typewriter style keyboard with graphic capabilities
Repeat key functional with all keys

MEMORY
CBM 8016: 16K (15359 net) random access memory (RAM)
CBM 8032: 32K (31743 net) random access memory (RAM)

POWER REQUIREMENTS
Volts: 110V
Cycles: 60 Hz
Watts: 100

SCREEN EDITING CAPABILITIES
Full cursor control (up, down, right, left)
Character insert and delete
Reverse character fields
Overstriking
Return key sends entire line to CPU regardless of cursor position

INPUT/OUTPUT
Parallel port
IEEE-488 bus
2 cassette ports
Memory and I/O expansion connectors

FIRMWARE
24K or ROM contains
BASIC (version 4.0) with direct (interactive) and indirect (program) modes
9-digit floating binary arithmetic
Tape and disk file handling software

CBM 8032 Computer $1795

CBM™ 8050 DUAL DRIVE FLOPPY DISK

The CBM 8050 Dual Drive Floppy Disk in an enhanced version of the intelligent CBM 2040 Disk Drive. The CBM 8050 has all of the features of the CBM 2040, and provides more powerful software capabilities, as well as nearly one megabyte of online storage capacity. The CBM 8050 supplies relative record files and automatic diskette initialization. It can copy all the files from one diskette to another without copying unused space. The CBM 8050 also offers improved error recovery and the ability to append to sequential files.

HARDWARE SPECIFICATIONS
Dual Drives
Two microprocessors
974K Bytes storage on two 5.25" diskettes (ss)
Tracks 70
Sectors 17-21
Soft sector format
IEEE-488 interface
Combination power (green) and error (red) indicator lights
Drive Activity indicator lights
Disk Operating System Firmware (12K ROM)
Disk Buffer (4K RAM)

FIRMWARE
DOS version 2.0
Sequential file manipulation
Sequential user files
Relative record files
Append to sequential files
Improved error recovery
Automatic diskette initialization
Automatic directory search
Command parser for syntax validation
Program load and save

CBM 8050 Dual Price $1795

CBM	PRODUCT DESCRIPTION	PRICE
4016	16K RAM-Graphics(N) or Business(B) Keyboard	$ 995.00
4032	32K RAM-Graphics(N) or Business(B) Keyboard	$1295.00
8032	32K RAM-80 Col. Screen-Business Keyboard	$1795.00
4022	Tractor Feed Printer	$ 795.00
4040	Dual Floppy-343K-DOS 2.0	$1295.00
8050	Dual Floppy-974K-DOS 2.0	$1795.00
4010	Voice Synthesizer	$ 395.00
8010	300 Baud IEEE Modem	$ 279.95
C2N Cassette	External Cassette Drive	$ 95.00
CBMto IEEE	CBM to 1st IEEE Peripheral	$ 39.95
IEEE to IEEE	IEEE to 2nd IEEE Peripheral	$ 49.95
2.1 DOS	DOS Upgrade for 2040	$ 100.00
4.0 DOS	O/S Upgrade for 40 Column Computer	$ 100.00
Word Pro 4+	Word Processing Software used w/8032	$ 450.00

*CBM is a registered trademark of Commodore. All prices and specifications are subject to change without notice.

commodore

NEECO

NEECO WILL MATCH ANY ADVERTISED PRICE ON CBM EQUIPMENT.

679 HIGHLAND AVE.
NEEDHAM, MA 02194

(617) 449-1760
Telex: 951021

MON-FRI 9:00 - 5:00
MasterCharge and VISA Accepted

Mapping And Modifying Unknown Machine Language

WordPro 4

Editor's Note: Although this article specifically deals with Professional Software's WordPro 4™, the material as presented is useful to any reader interested in learning more about machine language.

Richard Mansfield
Philipsburg, PA

At one time or another, all machine language programmers will face the challenge of modifying a large, undocumented program. If the program is your own, if you have documented the routines and major variables, the problems are relatively minor. But if you buy WordPro 4 Plus and want to run a Centronics printer with it — the challenge is severe. WordPro 4 Plus contains roughly 21,000 bytes of machine language code. And nothing but your wits and patience will help you unlock its secrets.

Experienced machine language programmers will forgive me if I seem to over-explain things. Without sufficient detail, however, articles such as this are useless to the majority of computer enthusiasts and contribute nothing to an understanding of machine language programming. For this reason, I will discuss techniques which I have found generally useful in analyzing unknown code and then explain some of the details of WordPro 4 Plus and how they can be used to customize the program for particular hardware and wordprocessing needs.

The Mountain Of Mystery Code

For several years I have been intrigued (and in awe) of the maps and lists provided by Jim Butterfield of BASIC machine language code. And, while I have written a good amount of machine language, the task of trying to untangle someone else's massive program seemed filled with pitfalls. I was happy to let Butterfield work his magic. Then things changed. I had to tackle a mountain of unknown WordPro 4 Plus code. I do consultant work for Maines Data Service in Pennsylvania and they wanted to put WordPro 4 Plus to work on a Centronics 737. The program is designed to work with NEC, Diablo, CBM, and other printers, but not Centronics.

For letter-quality printing, the Centronics uses a character set called "proportional" which looks very like typewriting. Each time the printer is powered up, it must receive two numbers from the computer — 27 and 17 — which alert it to use the proportional mode until further notice. The 27 is a special number (ESC, literally "escape" — meaning "take notice, something unusual follows") and it generally preceeds special printer codes. The 17 is notification that the printing style is to be proportional. WordPro 4 Plus will not send these numbers to the printer. The program was not written with any provision for Centronics.

It is possible, by defining special characters within a text, to send such numbers from within WordPro 4 to a printer. But this is unnecessarily complicated and needs to be remembered each session. Clearly, it would be desirable were the program to send the code automatically. Computers, as one of their primary advantages, automate. And, more important, the less complicated a program is to operate, the easier it is to demonstrate and to sell to businessmen. During demonstrations, you do not want to be typing in special symbols and odd codes just to get the printer set up. This has a chilling effect on prospective buyers.

techniques ... generally useful in analyzing unknown code ...

And WordPro 4's initialization sequence asks several questions such as the disk device number, printer number, ASCII? etc., all of which could be programmed in to save time and to customize the program for a specific hardware configuration. Finally, the underlining routine within WordPro 4 does not work correctly with Centronics. That should be fixed too. The machine language code will have to be mapped and studied. Unlike BASIC, which is essentially *lineal,* machine language does not proceed clearly from one job to the next. Rather, it is comprised of hundreds of interwoven, nested, and sometimes self-modifying routines. In BASIC, a main loop is fairly obvious (FOR ....NEXT or something similar), but in machine language, the main loop might be several dozen JSR statements of obscure purpose.

What to do? Like many machine language programs, WordPro 4 starts off with a little touch of BASIC on the theory that typing RUN is simpler than SYS 1037. If you load, then list WordPro 4, you will see 10 SYS 1037 on the screen. This means that the program proper begins at 1037. It starts there to make room for that BASIC line. Since the first thing we want to do is to send our code characters to the printer, we can just squeeze in the following without disturbing the body of WordPro 4:

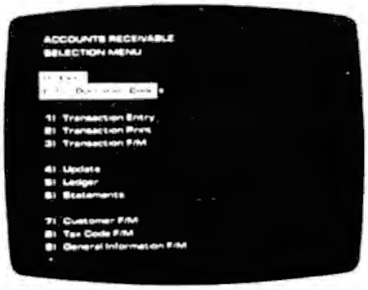
```
1OPEN4,4:SYS864
```

This allows us to easily open the correct I/0 (in/out) channel to the printer (and I/0 is the most

Professional Business Software

For The Commodore 32K Microcomputer System
With 2040 Dual Drive Disk & 2022 Tractor Feed Printer






General Ledger

- Holds Up To 300 Accounts
- Accepts Up To 3000 Transactions Per Month.
- Cash Disbursements Journal, Cash Receipts Journal, and Petty Cash Journal for simplified data entry.
- Maintains Account Balances For Present Month, Present Quarter, Present Year, Three Previous Quarters, And Previous Year.
- Complete Financial Reports Including Trial Balance, Balance Sheet, Profit & Loss Statement, Cash Receipts Journal, Cash Disbursements Journal, Petty Cash Journal and more.
- Accepts Postings From External Sources Such As Accounts Payable, Accounts Receivable, Payroll, Etc . ...

Accounts Payable

- Interactive Data Entry With Verified Input And Complete Operator Prompting.
- Automatic Application Of Credit And Debit Memos
- Maintains Complete Purchase Records For Up To 200 Vendors.
- Invoice File Accepts Up To 400 Invoices.
- Random Access File Organization Allows Fast Individual Record Updating
- Multiple Reports Provide A Complete Audit Trail.
- Check Printing With Full Invoice Detail.
- Full Invoice Aging
- Automatic Posting To General Ledger .. .

Accounts Receivable

- Maintains Invoice File For Up To 300 Invoices.
- Accomodates Full Or Partial Invoice Payments.
- Customer File Maintains Purchase Information For Up To 1000 Customers.
- Allows For Automatic Progress Billing.
- Provides For Credit And Debit Memos As Well As Invoices.
- Prints Individualized Customer Statements.
- Interactive Data Entry With FullOperator Prompting.
- Complete Data Input Verification And Formating.
- Automatic Posting To General Ledger ....

Payroll

- Maintains Monthly, Quarterly, And Yearly Cumulative Totals For Each Employee.
- Payroll Check Printing With Full Deduction And Pay Detail.
- Sixteen Different Reports Including W2 And 941.
- Interactive Data Entry With Easy Correction Of Entry Errors.
- Automatic Data Verification
- Complete Job Costing Option With Cumulative Totals And Overhead Calculations.
- Random Access File Organization For Fast Updating Of Individual Records.
- Automatic Posting To General Ledger ..

Structured around the time tested and reliability proven series of business software systems developed by Osbome and Associates, these programs have been designed to fill the need of a comprehensive accounting package for the new Commodore PET micro computer system. Each program can either stand alone, or be integrated with the others in a total software system.

Designed with the first time user in mind, these programs lead the operator through step by step, verified data entry It is impossible to 'crash' a program due to operator error or invalid data input. Design consistency has been maintained from program to program to greatly increase operator familiarity and confidence.

Documentation, normally a problem for small systems users, is provided by the comprehensive series of Osborne and Associates user manuals. These three manuals together total over 800 pages of detailed step by step instructions written at three levels for DP Department Managers, Data Entry Operators, and Programmers. You don't have to worry about getting 'promises' instead of documentation because the documentation was written before the programs were developed. A second set of manuals details any changes required during conversion. Each program provided on disk with complete documentation. Packaged in a handsome three ring binder with pockets and twelve monthly dividers for convenient storage of reports.

See your nearest Commodore dealer for a demonstration of this outstanding business software system.

CMS Software Systems

5115 MENEFEE DRIVE • DALLAS TX 75227 • 214-381-0690 tricky, most complex aspect of machine language programming). So, we don't bother doing it in machine language. We let BASIC do it for us. The very first thing that WordPro 4 does will be to jump to a vacant area of lower RAM (cells 860-1017) and perform what's in a little program that sends the proper numbers to the printer and then jumps back up to the start of WordPro 4 as if nothing had happened.

```
0360 LDX #$04
0362 JSR $FFC9
0365 LDA #$1B
0367 JSR $FFD2
036A LDA #$11
036C JSR $FFD2
036F JSR $FFE7
0372 JSR $FFCC
0375 JMP $17CC
```

This routine first puts the peripheral device number into register X. The printer is usually #4 and the next command (JSR FFC9) wants to have that device number in the X register. This JSR sets up the proper channels so that the computer will correctly talk to an external device. In this case, preparations will be made to send something to the printer.

Then we load the A register (the accumulator) with 1B which is the hex equivalent of 27. And we send that number to the printer along the already arranged channels. JSR FFD2 is an extremely common routine in machine language in Commodore computers. It sends a byte, usually to the screen, which has been placed in the accumulator. Next we put an 11 (17 decimal) in and again send it to the printer. Now our Centronics knows (it has a memory and some intelligence) that it will be printing in the proportional mode.

The next two JSRs (jump to subroutine) are into CBM BASIC also. We save a great deal of time by using routines already in ROM BASIC for these tasks. FFE7 aborts (cancels) all the channels set up previously. In this case, it cleans up the output to the printer so that no additional bytes can sneak in after the 27 and 17. If they did, they would be the first thing printed later on since they would remain first in line on the printer's buffer (holding area). Writing to a Mr. O'Connor, we might then see:

?,#Dear Mr. O'Connor:

The first three characters went into the printer without our realizing it. After this, we JSR to FFCC which restores the default (the conditions that the computer sets up when power is turned on) values for the output channels. With the default conditions reestablished, anything sent to FFD2 would print to the screen which is what the computer expects unless it deliberately changes the output status.

Finally, we jump directly into the WordPro 4 machine language code. This is where WordPro 4 initializes *its* default status and makes other necessary preparations. How did we know to jump to address 17cc (6092 decimal)? When we first examined WordPro 4, we noticed that its BASIC consisted of 10 SYS 1037. So, we looked at this address with our monitor and it was a jump to 17cc. Since we were merely adding some printer-specific code, we substituted a jump to 0360 *before* any WordPro 4 activities and then, at the conclusion of our tasks, we simply jumped up to where WordPro 4 wanted to go in the first place.

In this case, we did not need to worry about saving registers because nothing had happened yet in WordPro 4's world. Usually, however, care must be taken when jumping out of the middle of a machine language program. There are three registers (temporary data storage cells) in the 6502 CPU (central processing unit). Each is important and can contain necessary information. If you write your own routine (or access a part of BASIC) you will probably cause changes in one or more of these registers. Then, when you return to the host program, it will find the registers changed. This leads to unpredictable mixups. To be on the safe side, it is best to set aside a few bytes of your lower RAM area and when you enter your parasite routine, the first thing done is to save the X, Y, and A register values. In addition, it does not hurt to PHP which saves the status of processor flags (indicators that the last performed task resulted in a zero, or an overflow, or a negative, etc.) — they can be important to the host program as well.

These saves are then reloaded back into their registers (PLP restores the processor status flags) just before returning to the host program. In some cases this is not essential, but it is not hard to do and can save hours of bug hunting later.

Simplifying The Task

It was not so very hard to do these things because the program has not yet taken control of the computer. A number of programs leave you some room to do similar initializing before jumping to their start-up procedures. After the program takes hold, on the other hand, things get rougher.

Visicalc, WordPro 4, and other popular machine language programs do take control of the computer. The STOP key is blocked. Once you're in the program, you follow their rules. A provision is usually made for exiting to BASIC (in WordPro 4 you type ESC, Shift, Q), but you also wipe out the program itself. So, to modify the program, the first thing to do is to determine where it is inputting your keyboard responses. Then, before it analyzes the meaning of what you type in, you can jump down to lower RAM again where another of your parasite routines awaits. You could then make Q mean anything you want it to. If some other letter were typed, the parasite throws program control back up to the host. This allows the addition of personalized features and commands to programs.

LETTER PERFECT T.M. LJK

WORD PROCESSING FOR THE *ATARI — 800 T.M.

MAIN - MENU

CURRENT DRIVE NUMBER #1

- → Editor ←
- Change Drive #
- Load
- Save
- Merge
- Screen Format
- Printer
- Lock
- Unlock
- Delete
- Format Disk
- Data Base Merge
- Quit

Press '<' or '>' to move cursor
Press (Return) for selection

USE: EPSON MX-80 and ATARI -825 PRINTERS

EASY TO USE: LETTER PERFECT is a character orientated word processor with the user in mind. The program (machine language) is very fast. It is a menu driven program that is very easy to operate. The program is a single load program and can work with one or more disk drives. It requires a minimum of 16K of memory and a single disk drive. With the Artari 825 printer you can print text with right hand justification. You may also use different type fonts (10 and 17 character per inch) within the body of the text itself. Boldface is printed as expanded print font. Underlining can be done as well as sending Escape characters within the body of the letter itself. All the formats are a default but you can change them all to desired values if you wish. Right Margin, left margin, top of form, line spacing, etc. are easily changed. Data Base Merge works with the sister program LETTER PERFECT—DATA BASE MANAGER. User may use this program to create mailing lists, and completely develop your own data base for your personal needs. All text packed before storage to diskette for greater storage capacity. Large Buffer allows you to pick up and move up to one full page of screen text and move it to any location in the text. Merge more than one file together for easy editing. Screen Format allows you to see on the video screen exactly how the text will appear on the printer. Automatic page numbering, headers and footers are easily accomplished. This program is easy to use because of its meaningful and easily mastered commands. Fully documented with a users manual that explains in simple language 'how to' completely use the program.

All this and more, for $149.95.

Features:

FULL CURSOR CONTROL
Home Cursor
Scroll Page Forward
Scroll Page Backward
Pause Scroll
Scroll Line at Time
Scrolling Speed Control
Move Cursor Down
Beginning of Text

MULTIFUNCTION FORMAT LINE
Standard Formats a Default
Formats Easily Changed
Right Justification
Left Margin
Page Width
Line Spacing
Lines Per Page
Form Stop
Set Page#
Top Margin
Bottom Margin

Delete a Character
Insert a Character
Delete a Line
Insert a Line
Headers and Footers
Shift Lock and Release
Global and Local Search and Replacement
Underlining and Boldface
Automatic Centering
Horizontal Tabs
Special Print Characters
Split Catalog
Page Numbering up to 65535
Prints up to 255 Copies of Single Text File
Non Printing Text Commenting

FUNCTIONS
Delete All Text
Delete All After Cursor
Delete All Before Cursor
Delete Next Block
Delete Buffer
Move Next Block to Buffer
Add Next Block to Buffer
Insert Block From Buffer
Merge Text Files

apple®

This program also available on the Apple in 40/80 Video (Super'R' Term, Smarterm, Videx, Bit-3). You may use any printer type. The Hays Micromodem II can be used to send files. Can be Reconfigured at any time to use different printer, 80 column board, or standard 40 column video. Much, Much, More!

DEALER INQUIRIES INVITED

COMPUTER BASED SOFTWARE — LJK ENTERPRISES Inc. TM

FREE CONTROL PAGE

LJK ENTERPRISES INC.,
P.O. Box 10827
St. Louis, MO 63129
(314) 846-6124

*Apple T.M. of Apple Computer Inc., — *T.M. Atari Computer Corporation.

Before going into the details of adding features to WordPro 4, it is worth noting that WordPro 4 is more than a program. Like Visicalc and some few others, it is a *language*. The bulk of ROM — BASIC and the operation systems, tape and machine language monitor, part of DOS (disk operating system), interrupt, screen, and keyboard control, and initialization — all of this uses up about the same space (in ROM) as WordPro 4 uses up (in its RAM and a crypto-ROM chip). In effect, when you RUN WordPro 4, it is to the computer what BASIC is. It has a set of commands and rules like BASIC. WordPro 4 might well be thought of as a wordprocessing language which temporarily dedicates the computer to writing jobs. BASIC is a language which develops computer programs. Visicalc, too, is a language. It models arithmetic relationships and is, in essence, an econometrics system simplified.

But back to our effort to examine WordPro 4. In the way that Toolkit adds new commands to BASIC and Supermon adds new features to the machine language monitor, we want to be able to break into the control loop of WordPro 4 and add a command or two. Commodore has thoughtfully provided us with an easy way to amplify the machine language monitor. Addresses 03FA and B can be poked with the address of a parasite routine. The machine language monitor, in the course of its activities, will go first to the address in this vector (target). Usually, the vector points right back to the monitor, but you can change that. As of now, though, no published software no matter how expensive, provides user extension vectors. You have to break in yourself. And, first, you have to find out where, in the jumble of twenty thousand bytes, this breakpoint is. And the stop key does not work.

Breaking Into Unknown Code

Doubtless there are other ways, but here are some techniques which I have found to be helpful in mapping machine language. The first one might be called ASCII Hunting. This is a two-step process.

Most programs will need to print messages (prompts, error listings, etc.) to the screen. And these messages are usually stored as a table (stored together) and separated from each other by a 0 which signals the end of any particular message.

A common way to print a canned message to the screen in machine language is:

```
     LDY #$00
2001 LDA $0730,Y    LOOP START
     BEQ $2010
     JSR $FFD2
     INY
     JMP $2001    END LOOP
2010 (continues with something else)
```

What is happening here is that the Y register is loaded with a zero (so it can be used as an increasing

counter to point to each letter of the message in turn). Then (at address 2001) we load the A register with whatever appears in the address $0730 *plus* the value of Y (so, the first time through this loop, it will load the letter found at address $0730 itself). The BEQ means Branch if EQual. In BASIC it would be: IF A=0 THEN GOTO (2010). So, if the A register ever gets a zero loaded into it, it will end the printout and jump to $2010 where more code will lead it into other aspects of the program. If not a zero, however, it falls through to the familiar jump-to-subroutine $FFD2 which prints that character to the screen. Next the Y register is incremented (1 is added to it) and we JMP (jump. BASIC would be GOTO) back to load the next letter in the message.

This routine can be placed in lower (between 860 and 1000, decimal) RAM and used to check through the entire WordPro 4 code for any ASCII texts. WordPro 4 starts at 0400 and ends at 55D0. After such checking, it will be discovered that most of the text messages are strung together between addresses 0700 and 1000. This is helpful information.

Most Monitor Extension programs, such as Extramon and Supermon, contain a pattern-matching routine. This allows you to specify a particular piece of program code and all addresses where the same code appears in a program will be printed out for you. So, to find out where WordPro 4 is printing out something, you can match the pattern: JSR $FFD2 INY JMP and you will find a number of such routines. In addition, you could specify particular ASCII, such as "SYNTAX ERROR" and find the location of that ASCII pattern. In this way, you are building a map of some addresses within WordPro 4 which will later help you to understand it.

With a list of the starting addresses of the various ASCII texts, you can pinpoint where they are called from within the program. It is likely that a message printout of "Clear all tabs" will be called from within the section of the program which handles tabulation. Knowing that "Clear all tabs0" appears at, say, 0800, you will try pattern matching this: LDA $0800,Y. If this code appears anywhere, it is a request for that particular message. A map can begin to take vague shape from such matches.

Life, though, is not perfect. While this traditional printout routine does appear within WordPro 4, another method is also used. A separate subroutine, dedicated to screen message prints, wants you to split the target address between the X and Y registers before JSRing to it. The LSB (least significant half) of an address is loaded into the X and the MSB into Y. Then JSR to $1633. This $1633 routine operates like our example routine above, looking for the delimiting zero and all. The problems is that your pattern match will have to be, for target 0800, LDX #00 LDY #08 JSR $1633. And, when the program wants some leading blanks and a carriage return, the JSR is $1624. But, never mind. Learning these little variations makes your next mapping job easier.

But, one might well ask, how can we modify the program without mucking it up? It was easy enough (well, sommewhat easy) to break in at the start, but how can we break in where the code is packed together, wall-to-wall bytes? A reasonable question. To demonstrate, we might as well do something useful to WordPro 4. Let's get rid of that annoying start-up quiz.

As mentioned, programs must be fairly general. The author of WordPro could not know in advance if you are using a standard disk drive (device #8) and a standard printer (device #4) in the usual Commodore configuration. What's more, he does not know which printer you are using. It might be a Diablo, a NEC, a Commodore, a Spinwriter, a Centronics or something else. This requires a quiz. When you first load WordPro 4, it always asks several questions so it knows what sort of text codes your printer wants and the locations of your peripherals. This quiz quickly becomes boring and you wish that you could just skip it. You wish that the program could incorporate this information without having to waste your time.

The first thing that WordPro 4 does is to print:

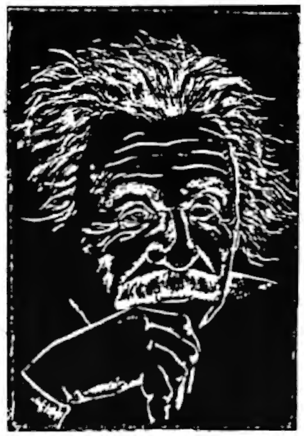
What Kind of Printer:
Spinwriter, Diablo, Qume, TEC, or Other?

We'll get rid of this question. We want to put an "O" for "Other" in, but automatically. First, we locate (as described above) the printout of this question. We also remember that a GET from screen (usually) is JSR FFE4. We track this down to a zone of code starting at address $54CB. (See example 1).

At once we can see something familiar about 54CB and following. It is the load X, load Y and subroutine to 1633 for printout. Note that this particular ASCII message starts at $5539 instead of from within the ASCII table at 0730-1000. There is no particular reason for this; the location only suggests that it was added to the program late and stuck near the top. In any event, after the printout, we GET a byte with FFE4. You would be typing a D here for Diablo, or a Q for Qume, etc. But we want neither the printout nor the GET. Just write over it. Cover what you want to eliminate with something you want done.

You could always resort to the 3-byte JSR $0380 cover, which, in tightly packed code might be necessary. You would change nothing about 54d2, for example, if you were to replace it with JSR $0380 and then at $0380 you put in JSR FFE4 and at $0383 RTS. What you would have done is to take a quick trip out of WordPro 4 and

An Intelligent Alternative



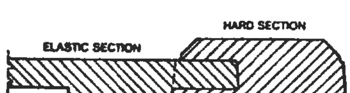
TYPRINTER 221

In the research you are doing before purchasing your computer printer, you are probably confused by the various claims, speeds, choices, shapes and prices. Well, we'd like to clear the air a bit and tell you about the most unusual computer-printer around — the TYPRINTER 221.

You see, it's unusual because it is **totally compatible** with **every** computer and word processing program . . . from the largest to the smallest. It's versatile to the point of incredibility . . . We'll discuss the broad advantages and explain the details.

THE DAISY WHEEL

The special daisy wheel supplied is of a unique design consisting of a 100 character carrying radii. Each radii is formed of two distinct types of plastic — an "elastic plastic" for the stalk of the radii, and a comparatively "hard plastic" used to form the character area. This, combined with a very narrow character profile and a special positioner on each of the 100 radii, guarantees a uniform character density. There is near perfect geometric positioning of the character with no character higher or lower than the others. And because of its unique dual material design, micro-vibrations have virtually been eliminated, leaving your final copy clean, clear and smudge free. The copy produced is comparable to that produced by metal daisy wheels and at a fraction of the cost.



THE KEYBOARD

The keyboard has been referred to as a triumph of human engineering - from the way the keys seem to have been custom designed to fit your fingers, to the way the special feature switches have been grouped. A flip of a switch (or under computer control of course) and the printer becomes a foreign language machine. Push a button, and like magic the printer automatically locates and lines up columns of figures, perfectly balanced between the margins. This incredibly fast, extraordinarily quiet electronic keyboard puts more programming power at you fingertips then printers costing five to ten times as much.

THE DISPLAY

The TYPRINTER 221 presents a new dimension in operator/machine communications. In the manual (typewriter) mode, the printer controls and verifies all entries before printing. The display exhibits the last 15 characters of the text, word-by-word, until the end of the line. The operator may control what will be printed before the actual printing takes place. This new found flexibility enables you to make modifications along the entire line and in both directions. This 20 character plasma display has the ability to scroll backwards as well as forwards; will give the operator a visual indication as to which print mode is currently being selected as well as the number of characters remaining before the right margin is reached. The display will also indicate to the operator:

The number of characters available in the memory
When the printer is in an error condition
When a pre programmed form lay out has been selected
When the printer is operating from the internal memory.
What characters will be inserted into an existing text.
When the memory for the previous line has been selected.
A warning message that the end of the page is being approached.
That a hyphenation decision must be made

PRINT MODE

The TYPRINTER 221 will allow you to automatically highlight individual characters, words or complete sentences. Whatever is entered from the keyboard or from the computer, even an existing text file, can be printed in one or more of the five different modes:

traditional printing;
underlined characters;
true bold characters where the horizontal component of the character is increased without disturbing the vertical component;
characters which are both bold and underlined, and;
a feature unique among computer printers-printing in reverse — white on black, sort of reverse video on paper.

MULTILINGUAL CAPABILITY

A unique and useful feature of the TYPRINTER 221 is its capability of being able to print in several languages without changing the daisy wheel. In addition to English, every standard daisy wheel has the ability and the necessary characters to print in French, Spanish, Italian and German.

THE FEATURES

Automatic justification of the right margin
The electronics of the TYPRINTER 221 have made right hand justification a simple, automatic operation.

Phrase and format storage
Phrases, dates, addresses, data, etc. that may be stored in your computer's memory may be sent over to the printer and stored in one of the "memory bins" of the printer. This information may then be used by the operator in the manual mode. This can save you hours when trying to get a form "just right."

Automatic centering
The TYPRINTER 221 will not only center any title between the pre-set margins, but will also center over one or more columns, or over any specific point and will even align copy with the right margin independent of the left margin.

Automatic vertical lines
A command from the computer enables an automatic feature which prints vertical lines at any point on the paper.

Automatic tab sequence recall
With the TYPRINTER 221 you may store and recall the most frequently needed margin and tab sequences for applications such as daily correspondence, statistical reports, etc. This guarantees consistent high quality appearance of each document.

Paragraph indent
A computer command instantly sets a temporary margin in order to print one or more indented paragraphs with respect to the right margin.

Automatic decimal point location
No matter how many figures to either the left or right of the decimal point, the TYPRINTER 221 will automatically line up the figures with the decimal point in any position you choose. Statistical printing has never been easier.

Column layout
This feature allows you to obtain automatic and perfect distribution of spaces between columns in respect to the margins. A perfect page balance is assured without the need to carry out calculations or additional operations.

There is a wide variety of options that you can add to TYPRINTER 221.

By now you are probably convinced that **we are sold** on our machine, and we hope you can understand why. In fact, why don't you use these facts to measure against any and/or all the other computer printers on the market.

When you do, you will realize the TYPRINTER 221 is an intelligent electronic typewriter, a text formatter — and a brilliant computer printer — available at a suggested list price of only $2850.

TYPRINTER 221 is available at your local computer shop — or we'll tell you where you can see and try one if you call us at

HOWARD INDUSTRIES
2051 E. CERRITOS AVE., 8C
ANAHEIM, CA 92806
714/778-3443

simply replicate what it would have done. Clearly, though, you would add some tasks of your own since the purpose of such a trip would be to modify the program in some way. And, too, you would want to save all register and processor flag conditions as described above.

But we need not resort to a quick trip. We've got the space here to overlay our additions within the program proper. So, our job is to first eliminate the printing out of the question itself. At $54CB we can put in an LDA #$4F. This has the effect of putting an "O" into the Accumulator just as if the GET had taken place and we had typed "O" into the keyboard. Now, we can put in a JMP $54DB. This jumps over the $1633 printout, the $FFE4 GET, and lands right where we want to continue. All those CMP's are comparing the Accumulator to the number which follows. It compares for an "S" (53) Spinwriter and a "D" (44), and so on. And, based on these comparisons, it branches to appropriate addresses which set conditions within Word-Pro 4 to harmonize with those printers. This is machine language for ON GOTO. Since we've already loaded the Accumulator with our "O" we just jump right to the branch comparison at $54DB.

At this point, carpers will ask why, if you can see that the branch will send you immediately to $551F, why not JMP $551F directly? This would be poor modification technique. A program as grand, as interwoven, as WordPro 4, flies all around within itself. As with a Fourth of July sparkler, it is devilishly difficult to know exactly what patterns are being set up, what causes lead to what effects. You have a shimmering uncertainty in large machine language programs, a case of what physicists call the uncertainty principle. Touch one variable the wrong way and you can send devastating vibrations undulating throughout the entire structure. Put another way, the less you mess with it the better.

A CMP instruction is central to machine language. It is the IF/THEN and ON/GOTO decision making event. It subtracts the number in the Accumulator from the argument. In our example, 4F-4F. The result is zero so we BEQ (Branch if EQual — the number will be equal if a subtraction results in zero). It is important to note that while the numbers are not themselves affected, other things are. A CMP changes the processor flags. These flags represent the effects of any operation, such as CMP, which disturbs them. Later testing of these flags by the program had better find them in the appropriate states. CMP will set the Z flag if there is an equality (as in our example) or it will reset itself if there is an inequality. The N flag responds to the sign bit (the seventh bit) in the case of negative result. And the C flag responds by a set when the Accumulator is equal to or greater than the argument. A CMP is generally followed by a BEQ, BCS, etc. branch instruction which tests these flag conditions (without affecting them). What's important about all this is that you want the flags to remain as they would have been had you never tampered with the program. So, let the CMP take place and let it set flags the way it would have. Sometimes, trying to be too smart will result in much wasted time tracking down odd program behaviors. This is what the British, in their wisdom, call "too clever by half."

Now the question and the GET have been eliminated. One thing remains to be done. Unlike INPUT, GET does not leave anything on screen to echo what you typed. Therefore Wordpro JSR $FFD2's later on to put the result of the GET on the screen. We don't want a stray "O" up there, so we can replace the three bytes starting $5524 with EA EA EA. This is the instruction NOP (no operation) and the computer slides right over it.

Eliminating The Rest Of The Quiz

Similar operations will take care of the other questions. Fix $18C3 to remove "Printer Device #?,"$18E6 for "Printer: CBM, ASCII, or Spinwriter?," and $193C for "Disk Device #?". We are deliberately leaving one input — the question which decides the number of lines for main versus extra text. This could be removed, too, if it does not serve any purpose. Most users, though, will want to define it.

If you want to add a feature for text input (following an ESC) look into the loop at $1E55. Be sure to reload Y with #$08 and X with #E0 just before you RTS out of your parasite code. And if your printer does not respond correctly to the underline convention (brackets), modify the code around $410F. The #$08 is a backspace convention. This routine must, of course, take into account the need to avoid underlining margins when an underline runs over from one line to the next.

There are a variety of other ways to map machine language. You can write a routine which drags nested subroutine addresses out of the stack via PLA TAX PLA. You could look for JSR's to known routines in BASIC ROM — very few appear in WordPro 4 though. If you are lucky and can avoid SEI or other problems, you might try single-stepping. And, of course you can set BRK (break, STOP in BASIC) breakpoints to check things and isolate program routines. Problems do arise. For example, large programs will need space to store variables. It will probably be the space which you are hoping to use for your monitor extension (Extramon) helper. Interrupt vectors will be tampered with. And so on. Trying to get a disassembly following a run of the program can be sticky. Variables may have overwritten your disassembler. But, with patience and some luck, you'll get your map, you'll learn alot about machine language, and you'll have more fun than any other computer game could provide. ©

A Floating Point Addition And Subtraction Routine

Marvin L. De Jong
The School of the Ozarks
Pt. Lookout, MO

I. Introduction

In previous articles in **COMPUTE!** we have described:

1) A program to convert a decimal number from the keyboard into a floating-point binary number,

Except for a few JSR and JMP instructions, the routine is relocatable. It would not be difficult to put all of these routines in PROM.

2) A program to convert a floating-point binary number to a decimal number and output the number.
3) A program to multiply two signed floating-point binary numbers,
4) A program to divide two signed floating-point binary numbers.

In this article we give a program that adds or subfracts two signed floating-point binary numbers. The programs complete a four-function package.

II. The Subtraction And Addition Routines

As before, three accumulators are used. The contents of accumulator A (ACCA in the program) are subtracted from the number in accumulator B (ACCB), and the result is stored in the result (RES) accumulator. Finally, the answer is moved back to a modified accumulator A that can be used by the output (floating-point binary to BCD routine) program. In the case of the addition program, the numbers in the two accumulators, A and B, are added rather than subtracted.

Accumulator A occupies locations $0000 through $0003 with a guard byte at $0004. The byte at $0000 is the most-significant byte. Accumulator B occupies locations $0020 through $0023 with a guard byte at $0024. The result accumulator is at $0010 to $0014. When the calculation is finished the answer is moved to the accumulator used by the floating-point binary to BCD routine to output the answer. Our accumulator architecture is identical in the four arithmetic function programs.

Here is the algorithm. It makes use of the fact that subtraction can be accomplished by changing the sign of the subtrahend and then adding. From algebra we know

$$a-b=a+(-b).$$

1. Entry point for subtraction. To subtract, complement the sign byte (ACCS) of A, then add.
2. Entry point for addition. Rotate smaller number right until exponents are the same (ACCX = BCCX).
3. Are the signs the same? Yes, go to 4. No, go to 8.
4. Sign of result = sign of addends.
5. Add the numbers.
6. If there is a carry, rotate right one place and increment exponent.
7. Go to round routine (part of multiplication listing).
8. Form the twos complement of the negative number.
9. Add the numbers.
10. If carry results, then the answer is +. Go to 7.
11. If no carry results, then the answer is –. Form the twos complement of the result. Go to 7.

These add and subtract routines use the same round instructions that the multiplication routine used, starting at DETOUR ($0C7D), and those instructions are not repeated here. Thus, you will find a JMP DETOUR instruction near the end of the routine. Except for a few JSR and JMP instructions, the routine is relocatable. It would not be difficult to put all of these routines in PROM. A driver program to test the routines is given in Listing 2.

Listing 2. An Input/Output/Add (or Subtract) Calling Program.

```
$0050 20 00 0E   JSR INPUT    Call the BCD to Floating-Point
                              Binary Routine.
$0053 30 B0 0F   JSR SUB1     Call the subroutine to modify
                              the accumulator.
$0056 20 C0 0F   JSR SUB2     Transfer ACCA to ACCB.
$0059 20 00 0E   JSR INPUT    Get the second number.
$005C 20 B0 0F   JSR SUB1     Fix the accumulator again.
$005F 20 00 09*  JSR SUB      Subtract the second number
                              from the first.
$0062 20 00 0B   JSR OUTPUT   Output the result using the
                              Floating-Point Binary to BCD
                              Routine.
$0065 00         BRK
*Change to 20 06 09 for addition.
```

```
SOURCE FILE: SUBADD
0C7D:            DETOUR   EQU   $0C7D
0027:            BCCS     EQU   $0027
0005:            ACCX     EQU   $0005
0007:            ACCS     EQU   $0007
0020:            ACCB     EQU   $0020
0025:            BCCX     EQU   $0025
0010:            RES      EQU   $0010
0000:            ACCA     EQU   $0000
----- NEXT OBJECT FILE NAME IS SUBADD.OBJ0
0900:                     ORG   $0900
0900:A5 07       SUB      LDA   ACCS        ;ENTRY POINT FOR SUBTRACTION
0902:49 FF                EOR   #$FF
0904:85 07                STA   ACCS
0906:A5 05       ADD      LDA   ACCX        ;ENTRY POINT FOR ADDITION
0908:C5 25                CMP   BCCX        ;COMPARE EXPONENTS
090A:F0 54                BEQ   OPRAT
090C:30 2A                BMI   ADJA
090E:A2 FB                LDX   #$FB
0910:A0 05                LDY   #05         ;CHECK FOR ZERO MANTISSA
0912:B5 25       BR1      LDA   ACCB+5,X
0914:D0 06                BNE   ROTB
0916:88                   DEY
0917:F0 10                BEQ   ZEROB
0919:E8                   INX
091A:D0 F6                BNE   BR1
091C:A2 FB       ROTB     LDX   #$FB        ;ROTATE MANTISSA RIGHT
091E:18                   CLC                ;AND INCREMENT EXPONENT
091F:76 25       BR2      ROR   ACCB+5,X
0921:E8                   INX
0922:D0 FB                BNE   BR2
0924:E6 25                INC   BCCX
0926:18                   CLC
0927:90 DD                BCC   ADD
0929:A0 08       ZEROB    LDY   #08
092B:A0 08                LDY   #08         ;MY MISTAKE. WHO NEEDS TWO LDY'S?
092D:A2 FB       UP       LDX   #$FB        ;MIGHT CATCH A COPYRIGHT VIOLATOR?
092F:76 05       HERE     ROR   ACCA+5,X
0931:E8                   INX
0932:D0 FB                BNE   HERE
0934:88                   DEY
0935:D0 F6                BNE   UP
0937:60                   RTS
0938:A2 FB       ADJA     LDX   #$FB        ;CHECK FOR ZERO MANTISSA AGAIN
093A:A0 05                LDY   #05
093C:B5 05       BR3      LDA   ACCA+5,X
093E:D0 06                BNE   ROTA
0940:88                   DEY
0941:F0 0F                BEQ   ZEROA
0943:E8                   INX
0944:D0 F6                BNE   BR3
0946:A2 FB       ROTA     LDX   #$FB        ;ROTATE MANTISSA RIGHT
0948:18                   CLC                ;AND INCREMENT EXPONENT
0949:76 05       BR4      ROR   ACCA+5,X
094B:E8                   INX
094C:D0 FB                BNE   BR4
094E:E6 05                INC   ACCX
0950:90 B4                BCC   ADD
0952:A5 25       ZEROA    LDA   BCCX        ;ADDEND IS ZERO
0954:85 05                STA   ACCX
0956:A2 03                LDX   #03
0958:B5 20       BACK     LDA   ACCB,X
095A:95 01                STA   ACCA+1,X
```

```
095A:95 01           STA   ACCA+1,X
095C:CA              DEX
095D:10 F9           BPL   BACK
095F:60              RTS
0960:A5 07    OPRAT  LDA   ACCS       ;CHECK THE SIGNS OF THE ADDENDS
0962:C5 27           CMP   BCCS
0964:D0 19           BNE   OPPOS
0966:20 DC 09        JSR   ADDNUM     ;ADD NUMBERS OF LIKE SIGN
0969:90 11           BCC   BR8
096B:A5 05           LDA   ACCX
096D:69 00           ADC   #00
096F:85 05           STA   ACCX
0971:50 01           BVC   BR6
0973:00              BRK
0974:A2 FB    BR6    LDX   #$FB
0976:38              SEC
0977:76 15    BR7    ROR   RES+5,X
0979:E8              INX
097A:D0 FB           BNE   BR7
097C:4C 7D 0C BR8    JMP   DETOUR
097F:A5 07    OPPOS  LDA   ACCS       ;COMPLEMENT THE NEGATIVE NUMBER
0981:F0 40           BEQ   CMPB       ;THEN ADD
0983:A2 04           LDX   #04
0985:B5 00    BR9    LDA   ACCA,X
0987:49 FF           EOR   #$FF
0989:95 00           STA   ACCA,X
098B:CA              DEX
098C:10 F7           BPL   BR9
098E:A0 04           LDY   #04
0990:38              SEC
0991:B5 00    BR10   LDA   ACCA,X
0993:69 00           ADC   #00
0995:95 00           STA   ACCA,X
0997:CA              DEX
0998:10 F7           BPL   BR10
099A:20 DC 09 FORTH  JSR   ADDNUM
099D:90 06           BCC   BR11
099F:A9 00           LDA   #00
09A1:85 07           STA   ACCS
09A3:F0 1B           BEQ   BR14
09A5:A9 FF    BR11   LDA   #$FF
09A7:85 07           STA   ACCS
09A9:A2 04           LDX   #$04
09AB:B5 10    BR12   LDA   RES,X
09AD:49 FF           EOR   #$FF
09AF:95 10           STA   RES,X
09B1:CA              DEX
09B2:10 F7           BPL   BR12
09B4:A2 04           LDX   #04
09B6:38              SEC
09B7:B5 10    BR13   LDA   RES,X
09B9:69 00           ADC   #00
09BB:95 10           STA   RES,X
09BD:CA              DEX
09BE:10 F7           BPL   BR13
09C0:4C 7D 0C BR14   JMP   DETOUR     ;GO TO ROUNDING ROUTINE
09C3:A2 04    CMPB   LDX   #04
09C5:B5 20    BR16   LDA   ACCB,X
09C7:49 FF           EOR   #$FF
09C9:95 20           STA   ACCB,X
09CB:CA              DEX
09CC:10 F7           BPL   BR16
```

SOFTWARE UNLIMITED

presenting the LARGEST SELECTION OF SOFTWARE EVER ASSEMBLED...

for ATARI® • APPLE® • PET® • and other Microcomputers

at SUPER DISCOUNT PRICES!

ATARI

- ☐ ATARI 800 COMPUTER ... 775.00
- ☐ ATARI 400 COMPUTER ... 390.00
- ☐ PHYSICS (AT) ... 24.50
- ☐ GREAT CLASSICS (AT) ... 24.50
- ☐ BASIC PHYSCOLOGY (AT) ... 24.50
- ☐ PRINCIPLES OF ECONOMICS (AT) ... 24.50
- ☐ SPELLING (AT) ... 25.50
- ☐ BASIC ELECTRICITY (AT) ... 24.50
- ☐ BASIC ALGEBRA (AT) ... 24.50
- ☐ 32 K RAMCRAM (AT) ... 195.00
- ☐ KINGDOM (AT) ... 13.55
- ☐ LEMONADE (AT) ... 13.55
- ☐ STATISTICS I (AT) ... 17.95
- ☐ BLACKJACK (AT) ... 13.55
- ☐ BIORYTHM (AT) ... 13.55
- ☐ HANGMAN (AT) ... 13.55
- ☐ SPACE INVADERS (AT) ... 17.95
- ☐ EUROPEAN CAPITALS (AT) ... 13.55
- ☐ MORTGAGE LOAN (AT) ... 13.55
- ☐ STATES & CAPITALS (AT) ... 13.55
- ☐ EDUCATION SYSTEM (AT) ... 22.50
- ☐ ATARI BASIC (AT) ... 53.95
- ☐ ASSEMBLER DEBUG (AT) ... 53.95
- ☐ BASKETBALL (AT) ... 35.95
- ☐ VIDEO EASEL-LIFE (AT) ... 35.95
- ☐ SUPER BREAKOUT (AT) ... 35.95
- ☐ MUSIC COMPOSER (AT) ... 53.95
- ☐ COMPUTER CHESS (AT) ... 35.95
- ☐ 3-D TIC TAC TOE (AT) ... 35.95
- ☐ STAR RAIDERS (AT) ... 53.95
- ☐ TELELINK (AT) ... 22.50
- ☐ PADDLES (AT) ... 17.95
- ☐ JOYSTICKS (AT) ... 17.95
- ☐ U.S. HISTORY (AT) ... 24.50
- ☐ U.S. GOVERNMENT (AT) ... 24.50
- ☐ SUPERVISERY SKILLS (AT) ... 24.50
- ☐ WORLD HISTORY (AT) ... 24.50
- ☐ BASIC SOCIOLOGY (AT) ... 24.50

ADVENTURE INTERNATIONAL

- ☐ ADVENTURE HINT SHEET ... 7.95
- ☐ ADVENTURE (1,2,3) (D) (AP) ... 35.95
- ☐ ADVENTURE (4,5,6) (D) (AP) ... 35.95
- ☐ ADVENTURE (7,8,9) (D) (AP) ... 35.95
- ☐ ADVENTURE #10 [D] ... 18.95
- ☐ ADVENTURE (specify 1-10) (AP,AT) ... 13.55
- ☐ PLANETOIDS [D] (AP) ... 17.95
- ☐ PLANETOIDS (AP) ... 13.55
- ☐ POKER (AP) ... 13.55
- ☐ POKER (AP) [D] ... 18.95
- ☐ KID-VENTURE #1 (AP) ... 13.55
- ☐ STAR TREK 3.5 (AT) ... 13.55
- ☐ MOUNTAIN SHOOT (AT) ... 8.95
- ☐ SUNDAY GOLF (AT) ... 8.95
- ☐ ANGLE WORMS (AT) ... 8.95
- ☐ DEFECTIONS (AT) ... 8.95

AVALON HILL

- ☐ MIDWAY (P,AP) ... 13.50
- ☐ NUKE WAR (P,AP) ... 13.50
- ☐ PLANET MINERS (P,AP) ... 13.50
- ☐ CONVOY RAIDER (P,AP) ... 13.50
- ☐ B1 BOMBER (P,AP) ... 13.50
- ☐ LORDS OF KARMA (P,AP) ... 18.00
- ☐ CONFLICT 2500 (AP,AT,P) ... 13.50
- ☐ COMPUTER ACQUIRE (AP,P) ... 18.00

AUTOMATED SIMULATION

- ☐ TUESDAY QUARTERBACK [D] (AP) ... 26.95
- ☐ STAR WARRIOR [C,D] (AP) ... 35.95
- ☐ THREE PACK [D] (AP,P) ... 45.00
- ☐ STARFLEET ORION [C,D] (AP) ... 22.50
- ☐ STARFLEET ORION [C] (P) ... 22.50
- ☐ INVASION ORION [C,D] (AP) ... 22.50
- ☐ INVASION ORION [C] (P) ... 22.50
- ☐ APSHAI [D] (AP) ... 35.95
- ☐ APSHAI [C] (P) ... 35.95

AUTOMATED SIMULATION

- ☐ RYN [D,C] (AP) ... 17.95
- ☐ RYN [C] (P,AT) ... 17.95
- ☐ MORLOC [C,D] (AP) ... 17.95
- ☐ MORLOC [C] (P,AT) ... 17.95
- ☐ RIGEL [C,D] (AP) ... 26.95
- ☐ RIGEL [C] (P) ... 26.95
- ☐ HELLFIRE [D] (AP) ... 35.95
- ☐ HELLFIRE [C] (P) ... 35.95

QUALITY SOFTWARE

- ☐ 6502 DISASSEMBLER (AT) ... 10.55
- ☐ ASTRO APPLE (AP) ... 13.55
- ☐ ASTRO APPLE (AP) [D] ... 17.95
- ☐ ASTEROIDS IN SPACE [D] (AP) ... 17.95
- ☐ ATARI ASSEMBLER (AT) ... 22.50
- ☐ BABBLE (AP) [D] ... 17.95
- ☐ BATTLESHIP COMMANDER (AP) ... 13.55
- ☐ BATTLESHIP COMMANDER [D] (AP) ... 17.95
- ☐ FASTGAMMON [D] (AP) ... 22.50
- ☐ FASTGAMMON (AP,AT) ... 17.95
- ☐ FRACAS ADVENTURE (AP) ... 17.95
- ☐ FRACAS ADVENTURE [D] (AP) ... 22.50
- ☐ LINKER (AP) [D] ... 44.00
- ☐ TANK TRAP (AT) ... 10.55
- ☐ TANK TRAP (AT) [D] ... 13.55
- ☐ TARI TREK (AT) ... 10.55
- ☐ TARI TREK (AT) [D] ... 13.55

PERSONAL SOFTWARE

- ☐ CCA MGMT [D] (AP) ... 85.00
- ☐ DESKTOP PLAN II [D] (AP) ... 175.00
- ☐ ZORK [D] (AP) ... 35.95
- ☐ MONTY MONOPOLY [D] (AP) ... 31.55
- ☐ VISICALC [D] (AP) ... 125.00
- ☐ VISICALC [D] (AT,P) ... 170.00

INSTANT SOFTWARE

- ☐ AIR FLIGHT SIMULATION (AP) ... 8.95
- ☐ APPLE FUN [D] (AP) ... 17.95
- ☐ CASINO (P) ... 7.25
- ☐ MORTGAGE (P) ... 7.25
- ☐ PADDLE FUN [D] (AP) ... 17.95
- ☐ PENNY ARCADE (P) ... 7.25
- ☐ SANTA PARAVIA FIUMACCIO (AP) [D] ... 17.95
- ☐ SAHARA WARRIOR (AP) ... 7.25
- ☐ SKY BOMBERS (AP) [D] ... 17.95
- ☐ SPACE WARS (AP) ... 7.25
- ☐ SUPERSHOOTERS (AP) ... 8.95

STRATEGIC SIMULATIONS

- ☐ COMPUTER AMBUSH [D] (AP) ... 51.50
- ☐ COMPUTER BISMARCK [D] (AP) ... 51.50
- ☐ COMPUTER CONFLICT [D] (AP) ... 35.00
- ☐ COMPUTER NAPOLEONICS [D] (AP) ... 51.50
- ☐ COMPUTER QUARTERBACK [D] (AP) ... 35.00
- ☐ AIR COMBAT [D] (AP) ... 51.50
- ☐ WARP FACTOR [D] (AP) ... 35.00
- ☐ CUTHROATS & CARTELS [D] (AP) ... 35.00
- ☐ OPERATION APOCALYPSE [D] (AP) ... 51.50
- ☐ TORPEDO FIRE [D] (AP) ... 51.50

SUB-LOGIC

- ☐ 3D GRAPHICS (AP) ... 40.00
- ☐ 3D GRAPHICS [D] (AP) ... 48.00
- ☐ A-2-FS1 FLIGHT SIMULATOR (AP) ... 22.00
- ☐ A-2-FS1 FLIGHT [D] (AP) ... 29.00

MICROSOFT SOFTWARE

- ☐ ADVENTURE [D] (AP) ... 25.50
- ☐ OLYMPIC DECATHALON [D] (AP) ... 20.00
- ☐ TYPING TUTOR (AP) [D] ... 17.95
- ☐ TYPING TUTOR (AP) ... 13.55
- ☐ Z-80 SOFTCARD [D] (AP) ... 280.00
- ☐ 16k RAM BOARD (AP) ... 165.00

ON LINE SYSTEMS

- ☐ HI-RES ADVEN. #0 (AP) [D] ... 17.95
- ☐ HI-RES ADVEN. #1 [D] (AP) ... 22.50
- ☐ HI-RES ADVEN. #2 [D] (AP, AT) ... 29.00
- ☐ HI-RES FOOTBALL #1 [D] (AP) ... 36.00
- ☐ HI-RES CRIBBAGE [D] (AP) ... 22.50
- ☐ PADDLE GRAPHICS [D] (AP) ... 36.00
- ☐ TABLET GRAPHICS [D] (AP) ... 44.95
- ☐ MISSILE DEFENSE [D] (AP) ... 26.95
- ☐ SUPERSCRIBE [D] (AP) ... 81.00

SIRIUS

- ☐ CYBER STRIKE [D] (AP) ... 36.00
- ☐ STAR CRUISER [D] (AP) ... 22.50
- ☐ BOTH BARRELS [D] (AP) ... 22.50
- ☐ PHANTOM FIVE [D] (AP) ... 36.00
- ☐ SPACE EGGS [D] (AP) ... 26.95

SYNERGISTIC SOFTWARE

- ☐ DUNGEON & WILDERNESS [D] (AP) ... 29.00
- ☐ DUNGEON (AP) ... 13.50
- ☐ DUNGEON [D] (AP) ... 15.75
- ☐ ODYSSEY [D] (AP) ... 27.00
- ☐ HIGHER GRAPHICS [D] (AP) ... 31.50
- ☐ WILDERNESS (AP) ... 15.75
- ☐ WILDERNESS [D] (AP) ... 18.00
- ☐ PROGRAM LINE EDITOR [D] (AP) ... 36.00

BRODERBUND

- ☐ GALACTIC EMPIRE (AP) [D] ... 22.50
- ☐ GALAXIAN (AP) [D] ... 22.50
- ☐ HYPER HEAD ON (AP) [D] ... 22.50
- ☐ GALACTIC REVOLUTION (AP) [D] ... 22.50
- ☐ TANK (AP) [D] ... 13.55
- ☐ TAWALA'S REDOUBT (AP) [D] ... 26.95
- ☐ GALACTIC TRADER (AP) [D] ... 22.50
- ☐ PUCKMAN (AP) [D] ... 22.50

MUSE COMPANY

- ☐ ABM [D] (AP) ... 22.50
- ☐ APPILOT II [D] (AP) ... 80.00
- ☐ BEST OF MUSE [D] (AP) ... 35.95
- ☐ GLOBAL WAR [D] (AP) ... 22.50
- ☐ SUPER TEXT II [D] (AP) ... 135.00
- ☐ THREE MILE ISLAND [D] (AP) ... 35.95
- ☐ U-DRAW II [D] (AP) ... 35.95
- ☐ THE VOICE [D] (AP) ... 35.95

IRIDIS

- ☐ IRIDIS 1 (AT) ... 8.95
- ☐ IRIDIS 1 (AT) [D] ... 11.75
- ☐ IRIDIS 2 (AT) ... 14.50
- ☐ IRIDIS 2 (AT) [D] ... 16.95

EDU-WARE

- ☐ ESP (AP) [D] ... 14.50
- ☐ NETWORK (AP) [D] ... 17.95
- ☐ PRISONER (AP) [D] ... 26.95
- ☐ SPACE (AP) [D] ... 26.95
- ☐ SPACE II (AP) [D] ... 22.50
- ☐ TERRORIST (AP) [D] ... 26.95

PROGRAMMA

- ☐ EXPAND-A-PORT (AP) ... 53.95
- ☐ JOYSTICK (AP) ... 44.95
- ☐ TINY PASCAL (AP) [D] ... 44.50
- ☐ WPS STANDARD (AP) [D] ... 117.00

HAYDEN

- ☐ SARGON II (AP) ... 25.00
- ☐ SARGON II (AP) [D] ... 30.00
- ☐ REVERSAL (AP) ... 25.00

ARTSCI

- ☐ MAGIC WINDOW [D] (AP) ... 89.95

If you don't see it listed, write...we probably have it in stock!

Check program desired.
Complete ordering information and mail entire ad.
Immediate Shipments from stock.

Ship the above programs as checked to:	Number of Programs Ordered .... ______
Mr./Mrs. ______	Amount of order .... ______
Address ______	N.Y. residents add Sales Tax ...... ______
City ______	Add shipping anywhere in the U.S. 2.00
State ______ Zip ______	Total amount enclosed .... ______
I have a ______ (name of Computer)	Charge my: ☐ Master Charge ☐ Visa
with ______ k memory	Signature ______
	Card No. ______ Expires ______

Personal Checks please allow 3 weeks.

KEY:
AT-Atari
AP-Apple
P-Pet
D-on Disc.
C-Cassette

If not marked-Cassette

ATARI is a trademark of ATARI INC.
APPLE is a trademark of APPLE COMPUTER, INC.
PET is a trademark of COMMODORE BUSINESS MACHINES
Prices subject to change without notice.

Mail to:

DIGIBYTE SYSTEMS CORP.

31 East 31st Street, New York, N.Y. 10016
Phone: (212) 889-8975

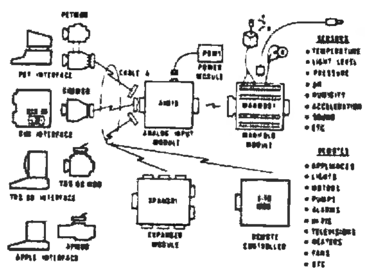
```
09CE:A2 04            LDX   #04
09D0:38               SEC
09D1:B5 20      BR15  LDA   ACCB,X
09D3:69 00            ADC   #00
09D5:95 20            STA   ACCB,X
09D7:CA               DEX
09D8:10 F7            BPL   BR15
09DA:30 BE            BMI   FORTH
09DC:A2 04      ADDNUM LDX  #04        ;SUBROUTINE THAT DOES THE ADDITION
09DE:18               CLC
09DF:B5 00      KCAB  LDA   ACCA,X
09E1:75 20            ADC   ACCB,X
09E3:95 10            STA   RES,X
09E5:CA               DEX
09E6:10 F7            BPL   KCAB
09E8:60               RTS

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

```
  00 ACCA         20 ACCB         07 ACCS         05 ACCX
0906 ADD        09DC ADDNUM     0938 ADJA       0958 BACK
  27 BCCS         25 BCCX       0991 BR10       09A5 BR11
09AB BR12       09C0 BR14       0912 BR1        09B7 BR13
09D1 BR15       09C5 BR16       091F BR2        093C BR3
0949 BR4        0974 BR6        0977 BR7        097C BR8
0985 BR9        09C3 CMPB       0C7D DETOUR     099A FORTH
092F HERE       09DF KCAB       097F OPPOS      0960 OPRAT
  10 RES        0946 ROTA       091C ROTB      ?0900 SUB
092D UP         0952 ZEROA      0929 ZEROB
```

©

Microcomputer Measurement And Control For PET,APPLE,KIM and AIM65



The world we live in is full of variables we want to measure. These include weight, temperature, pressure, humidity, speed and fluid level. These variables are continuous and their values may be represented by a voltage. This voltage is the analog of the physical variable. A device which converts a physical, mechanical or chemical quantity to a voltage is called a sensor.

Computers do not understand voltages: They understand bits. Bits are digital signals. A device which converts voltages to bits is an analog-to-digital converter. Our AIM 16 (Analog Input Module) is a 16 input analog-to-digital converter.

The goal of Connecticut microComputer in designing the uMAC SYSTEMS is to produce easy to use, low cost data acquisition and control modules for small computers. These acquisition and control modules will include digital input sensing (e.g. switches), analog input sensing (e.g. temperature, humidity), digital output control (e.g. lamps, motors, alarms), and analog output control (e.g. X-Y plotters, or oscilloscopes).

Connectors

The AIM 16 requires connections to its input port (analog inputs) and its output port (computer interface). The ICON (Input CONnector) is a 20 pin, solder eyelet, edge connector for connecting inputs to each of the AIM16's 16 channels. The OCON (Output CONnector) is a 20 pin, solder eyelet edge connector for connecting the computer's input and output ports to the AIM16.

The MANMOD1 (MANifold MODule) replaces the ICON. It has screw terminals and barrier strips for all 16 inputs for connecting pots, joysticks, voltage sources, etc.

CABLE A24 (24 inch interconnect cable) has an interface connector on one end and an OCON equivalent on the other. This cable provides connections between the uMACSYSTEMS computer interfaces and the AIM 16 or XPANDR1 and between the XPANDR1 and up to eight AIM 16s.

Analog Input Module

The AIM 16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM 16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the uMAC SYSTEMS special interfaces.

The input voltage range is 0 to 5.12 volts. The input voltage is converted to a count between 0 and 255 (00 and FF hex). Resolution is 20 millivolts per count. Accuracy is 0.5% ± 1 bit. Conversion time is less than 100 microseconds per channel. All 16 channels can be scanned in less than 1.5 milliseconds.

Power requirements are 12 volts DC at 60 ma.

POW1

The POW1 is the power module for the AIM16. One POW1 supplies enough power for one AIM16, one MANMOD1, sixteen sensors, one XPANDR1 and one computer interface. The POW1 comes in an American version (POW1a) for 110 VAC and in a European version (POW1e) for 230 VAC.

TEMPSENS

This module provides two temperature probes for use by the AIM16. This module should be used with the MANMOD1 for ease of hookup. The MANMOD1 will support up to 16 probes (eight TEMPSENS modules). Resolution for each probe is 1°F.

Remote Controller-Clock and Calendar

AN INEXPENSIVE CONTROL SOLUTION FOR

HOME SECURITY • ENERGY CONSERVATION • GREENHOUSES • ENVIRONMENTAL CONTROL • INDUSTRIAL CONTROL • LABORATORIES

SUPER X-10 MOD SPECS

1. Remote controller
   Controls up to 256 different remote devices by sending signals over the house wiring to remote modules. Uses BSR remote modules available all over the USA (Sears, Radio Shack, etc.). Does not require BSR control module. Does not use sonic link.
2. Clock/calendar
   Time of day - hours, minutes, seconds
   Date - month, day - automatically corrects for 28,29,30 and 31 day months. Day of the week.
3. Digital input/outputs
   8 inputs - TTL levels or switch closures. Can be used as a trigger for a stored sequence.
   8 outputs - TTL levels
   Power supply included 110VAC only.

XPANDR1

The XPANDR1 allows up to eight Input/Output modules to be connected to a computer at one time. The XPANDR1 is connected to the computer in place of the AIM16 or X10 MOD. Up to eight AIM16s or seven Aim 16s and one X10 MOD are then connected to each of the eight ports provided using a CABLE A24 for each module.

For your convenience the AIM16 and the X10 MOD come as part of a number of sets. The mimmum configuration for a usable system is the AIM16 Starter Set 1 which includes one AIM16, one POW1, one ICON and one OCON. The AIM16 Starter Set 2 includes a MANMOD1 in place of the ICON. The minimum configuration for a usable system is the X10 MOD Starter Set which includes one X10 MOD, one ICON and one OCON. These sets require that you have a hardware knowledge of your computer and of computer interfacing.

For simple plug compatible systems we also offer computer interfaces and sets for many computers.

AIM16 .... 179.00
SUPER X10 MOD (110 VAC only) 249.00
POW1a (POWer module-110 VAC) 14.95
POW1e (POWer module-230 VAC) 24.95
ICON (Input CONnector) 9.95
OCON (Output CONnector) 9.95
MANMOD1 (MANifold MODule) 59.95
CABLE A24 (24 inch interconnect cable) 19.95
XPANDR1 (allows up to 8 Input or Output modules to be connected to a computer at one time) 59.95
TEMPSENS2P1 (two temperature probes, -10°F to 160°F) 69.95
LIGHTSENS1P1 (light level probe) 89.95

The following sets include one AIM16, one POW1, one OCON and one ICON.

AIM16 Starter Set 1a (110 VAC) 189.00
AIM16 Starter Set 1e (230 VAC) 199.00

All prices and specifications subject to change without notice. Our 30-day money back guarantee applies.

The following sets include one AIM16, one POW1, one OCON and one MANMOD1.

AIM16 Starter Set 2a (110 VAC) 239.00
AIM16 Starter Set 2e (230 VAC) 249.00

The following modules plug into their respective computers and, when used with a CABLE A24, eliminate the need for custom wiring of the computer interface.

PETMOD (Commodore PET) 49.95
KIMMOD (KIM,SYM) 39.95
APMOD (APPLE II) 59.95
TRS-80 MOD (Radio Shack TRS-80) 59.95
AIM65 MOD (AIM 65) 39.95

The following sets include one AIM16, one POW1, one MANMOD1, one CABLE A24 and one computer interface module

PETSET1a (Commodore PET - 110 VAC) 295.00
PETSET1e (Commodore PET - 230 VAC) 305.00
KIMSET1a (KIM,SYM,AIM65 - 110 VAC) 285.00
KIMSET1e (KIM,SYM,AIM65 - 230 VAC) 295.00
APSET1a(APPLE II - 110 VAC) 295.00
APSET1e(APPLE II - 230 VAC) 305.00
TRS-80 SET1a (Radio Shack TRS-80 - 110 VAC) 295.00
TRS-80 SET1e(Radio Shack TRS-80 - 230 VAC) 305.00
AIM65 SET1a(AIM65-110 VAC) 285.00
AIM65 SET1e(AIM65-230 VAC) 295.00

The following sets include one X10 MOD, one CABLE A24, one ICON and one computer interface module.

PETSET2(Commodore PET) 295.00
KIMSET2(KIM,SYM) 285.00
APSET2(APPLE II) 295.00
TRS-80 SET2 (Radio Shack TRS-80) 295.00
AIM65 SET2 (AIM65) 285.00
SUPER X10 MOD/XPANDR1 SET2 (if you already have a SET1) 295.00

Finding Square Roots In Assembly Language

Leo J. Scanlon
El Toro, CA

Many people who have transitioned into assembly language programming from a background in Basic have done so with the intention of speeding up their program's execution time (for a real-time application, perhaps) or reducing the program in an effort to get "closer" to the microprocessor machine code, for the sake of personal education.

The discussion given here should give you a sufficient background to develop a square root program for a binary number of any length.

In making such a transition, however, programmers quickly come to the stark realization that they no longer have FOR, NEXT, PRINT and all of the other nifty functions that are provided by Basic. All of these operations are still available to assembly language programmers, of course, but not in the form of a few simple words. If required, they must be *simulated* with an appropriate series of assembly language instructions.

The simpler Basic statements, such as FOR and NEXT, can usually be simulated with just a few assembly language instructions. More complex statements, such as PRINT, may require a dozen or more instructions in assembly language. The most complex Basic statements (RND, SIN, TAN and so on) can require a much more extensive routine in assembly language. In this article, we will discuss a possible approach to simulating a function of moderate complexity, **SQR** (square root) in 6502 assembly language. Specifically, *we will develop an assembly language program to extract the 8-bit square root of a 16-bit, unsigned, binary number.* The discussion given here should give you sufficient background to develop a square root program for a binary number of any length.

A Square Root Algorithm

The available literature includes a variety of algorithms for extracting the square root of a binary number. Perhaps the simplest is contained in one of my own books, **6502 Software Design** (Howard W. Sams & Co., Inc.; Indianapolis, IN; 1980). That algorithm states: *The square root of an integer is equal to the number of successively higher odd numbers that can be subtracted from it.* That is, you subtract -1, then -3, then -5 ... and so on, until the remainder becomes zero or negative; the count of odd numbers that have been subtracted represent the integer square root.

Although the algorithm works, it can be extremely slow, because each square root count must be preceded by one execution of the subtraction sequence. If we are processing a small number, such as 25, the 6502 will only execute the subtraction sequence five times. However, with progressively larger values, the 6502 will have to make more and more executions of this sequence. At the extreme, the square root of the largest number that can be represented in 16 bits (65,535) requires the microprocessor to execute this same double-precision subtract sequence *255 times*! As an alternative, let's discuss an algorithm that permits a square root to be found in exactly *eight executions* of the main program loop.

Raise Your Hand If You Know The Answer!

Many readers will recall the method we learned in elementary school or junior high to find the square root of a number using pencil and paper (don't all groan at once), the one where we pair off the digits to the left of the decimal point and find the square root of each pair. Well, as it turns out, this method is one of the best ways for calculating square roots on a computer, too. (Sonofagun, maybe old Miss McDonald was actually *ahead* of her time!) In fact, you'll see this algorithm written up in such blue-blooded references as Kai Hwang's **Computer Arithmetic** (John Wiley & Sons; New York; 1979).

For example, to take the square root of 4536, you would find the most-significant digit with this kind of procedure:

```
           6
         √45 36.
          36
   12_     9
```

What is the "12_" sitting off to the left? It is the first digit of the square root (6), doubled with a space reserved for the next digit. The next step looks like this:

```
           6  7
         √45 36.
          36
   127     9 36
           8 89
   134_      47
```

If greater accuracy is desired, the process could be

continued to the right, to pick up the fractional square root. Ask Miss McDonald if you don't remember how to do it.

Essentially, the preceding algorithm is as follows:

1. Starting at the decimal point, pair off the digits to the left and (if appropriate) to the right. If the integer portion of the number contains an odd number of digits, the most-significant digit will be processed individually.
2. Find the square root of the most-significant digit(s), and enter this digit in the result.

... a fixed-point number is just a string of digits — ones and zeros, in this case ...

3. Subtract the square of this root from the most-significant digit(s) of the original number.
4. If there are more significant digits, continue; otherwise we are done.
5. Form a "dividend," by appending the next digit pair to the remainder from Step 3.
6. Form a "divisor," by doubling the current square root and reserving a least-significant digit position.
7. Compare the "divisor," to the dividend (assume 0 for the reserved digit position in the divisor), and proceed as follows:
   - **A.** If the divisor is greater than or equal to the dividend, enter a 0 in the result and return to Step 4.
   - **B.** If the divisor is less than the dividend, find the largest digit that will make the product of square root times divisor less than or equal to the dividend. Enter this digit in the result and the reserved position of the divisor, multiply, subtract the product from the dividend, and return to Step 4.

I don't know how you feel, but to me, that description is almost as difficult as memorizing how it's done!

Extracting The Square Root Of A Binary Number

The preceding algorithm applies to binary numbers as well as decimal numbers. And, with binary numbers, you gain one distinct advantage: because binary numbers are comprised of only ones and zeroes, their square roots are also comprised of only ones and zeroes. This means that while you're constructing the square root, if a result digit is not a one, it has to be a zero — and vice versa.

For this article, we'll use our pencil-and-paper algorithm to extract the 8-bit square root of a 16-bit unsigned, fixed-point number in memory. (Recall that a *fixed-point number* is just a string of digits — ones and zeroes, in this case — in which the binary point is assumed to be at some user-specified location in the string. The location of the binary point has no effect on the operations being discussed.) Let us assume that the low-order and high-order bytes of the 16-bit number are initially held in locations LOBYTE and HIBYTE, and that the 8-bit square root is to be returned in location ROOT. Based on these requirements, Example 1 shows a 6502 routine that will do the job.

This routine, called SQRT, begins by loading a count of 8 into the X register (since there are eight pairs of bits in a 16-bit number) and then clearing the dividend register (the A register, here) and the location ROOT. With this initialization completed, the 6502 enters the main program loop and rotates the first pair of bits into the dividend register (see Figure 1). In the next step — which is meaningless for the first bit pair, but required for all remaining pairs — ROOT is left-shifted, to make room for the next result bit, and then doubled to form the divisor.

Example 1. A 16-Bit Square Root Routine

This routine extracts the square root of an unsigned, fixed-point number in memory. The number is contained in locations LOBYTE and HIBYTE, and the 8-bit square root is returned in location ROOT. This routine affects the A, X and Y registers.

```
0000            LOBYTE   *=*+1            Low byte of number
0001            HIBYTE   *=*+1            High byte of number
0002            ROOT     *=*+1            Square root location
0003            DIVISR   *=*+1            Divisor location
0003                     *=$200
0200  A2  08    SQRT     LDX  #8          Count = 8
0202  A9  00             LDA  #0          Clear dividend register
0204  85  02             STA  ROOT          and square root location
0206  06  00    LOOP     ASL  LOBYTE      Rotate two MSB's into A
0208  26  01             ROL  HIBYTE
020A  2A                 ROL  A
020B  06  00             ASL  LOBYTE
020D  26  01             ROL  HIBYTE
020F  2A                 ROL  A
0210  06  02             ASL  ROOT        Left-shift current square root
0212  A4  02             LDY  ROOT
0214  84  03             STY  DIVISR
0216  06  03             ASL  DIVISR      and double it to form divisor
0218  C5  03             CMP  DIVISR      Dividend greater than divisor?
021A  F0  08             BEQ  DECCNT        No. Square root bit = 0
021C  90  06             BCC  DECCNT
021E  E6  02             INC  ROOT          Yes. Square root bit = 1
0220  E6  03             INC  DIVISR         and divisor LSB = 1
0222  E5  03             SBC  DIVISR      Calculate remainder (in A)
0224  CA        DECCNT   DEX              Loop until 16 bits
0225  D0  Df             BNE  LOOP          have been processed.
```

At this point, the 8-bit dividend (in A) is compared with the 8-bit divisor (in DIVISR), to determine whether the next bit of the result should be a 0 or a 1. If the dividend is less than or equal to

the divisor, the square root bit is 0, so the 6502 branches to DECCNT, where counter X is decremented. Otherwise, if the dividend is greater than the divisor, the square root bit is 1, so both ROOT and DIVISR are incremented, and a remainder (dividend - divisor) calculated, before the decrement operation takes place. Why don't we need to set the Carry bit before the subtraction? We don't need to do this because the fact that the branch instructions BEQ DECCNT and BCC DECCNT both failed *guarantees* that Carry is already set at this time!

Incidentally, note that the SQRT routine not only affects the A, X and Y registers, but destroys the original number in LOBYTE and HIBYTE as well. It's quite possible to save any or all of these parameters by pushing them into the stack before you execute the SQRT routine, and pulling them off the stack after SQRT has been executed. For example, to save A, X, Y, LOBYTE and HIBYTE on the stack, insert this sequence as the first nine instructions in the routine:

```
SQRT PHA            Save A
     TXA            Save X
     PHA
     TYA            Save Y
     PHA
     LDA  LOBYTE    Save LOBYTE
     PHA
     LDA  HIBYTE    Save HIBYTE
     PHA
```

As you know, data on the stack must be retrieved in the opposite order from which it was stored. To retrieve the "pushed" parameters, add this sequence to the end of the SQRT routine:

```
PLA            Pull HIBYTE
STA  HIBYTE
PLA            Pull LOBYTE
STA  LOBYTE
PLA            Pull Y
TAY
PLA            Pull X
TAX
PLA            Pull A
```

Trouble In Paradise

Since 16 bits in memory can hold hexadecimal values from 0 to $FF (where "$" means hexadecimal), the routine in Example 1 should be able to extract the square root of any number between 0 and 65,535. But, as I found out during the debugging process, *the routine returns a wrong answer if your original number is greater than $9FFB (decimal 40,952)*! It fails at that point because we're using an 8-bit divisor, so when ROOT has a 1 in the most-significant bit position and we execute the sequence

```
LDY  ROOT
STY  DIVISR
ASL  DIVISR
```

at locations $0212 through $0216, the 1 gets *lost* when we execute the instruction ASL DIVISR. Obvoiusly, what we need to handle the full range of 16-bit values is a routine that has a 16-bit divisor and a 16-bit dividend. Let's see how that's done.

An Improved 16-Bit Square Root Routine

A 16-bit divisor and dividend can be formed by simply taking the 8-bit divisor and dividend from Example 1 and adding a byte to each of them. For the dividend, we'll retain the A register to hold the high-order byte, and allocate a new memory location (DIVDND) to hold the low-order byte. For the divisor, we'll just make DIVISR into a two-byte parameter; DIVISR holds the low-order byte and the next location, DIVISR + 1, holds the high-order byte. Figure 2 illustrates the new 16-bit dividend and divisor.

With these new, double-size parameters, you would anticipate a considerably longer program — and you'd be right! The improved version of the SQRT routine is shown in Example 2. Except for the additional data manipulation, this routine is constructed exactly like Example 1, so I won't bore you with another detailed description of its operation. You may be interested in how fast this routine can extract a square root, however, If the base number is $0000, SQRT takes 605 machine cycles to extract the square root. If the base number is $FFFF, the square root will be extracted in 825 machine cycles. (If your computer has a 1-MHz 6502, these times translate to 605 s and 825 s.) Therefore, any number you use will execute somewhere within these extremes.

Example 2. An Improved 16-Bit Square Root Routine

This routine extracts the square root of an unsigned, fixed-point number in memory. The number is contained in locations LOBYTE and HYBYTE, and the 8-bit square root is returned in location ROOT. This routine affects the A, X and Y registers.

```
0000            LOBYTE  G=*+1           Low byte of number
0001            HIBYTE  *=*+1           High byte of number
0002            ROOT    *=*+1           Square root location
0003            DIVISR  *=*+2           16-bit divisor
0005            DIVDND  *=*+1           Low byte of dividend
0006                    *=$200
0200 A2 08      SQRT    LDX  #8         Count = 8
0202 A9 00              LDA  #0         Clear dividend,
0204 85 05              STA  DIVDND
0206 85 04              STA  DIVISR+1    high byte of divisor
0208 85 02              STA  ROOT        and square root location
020A 06 00      LOOP    ASL  LOBYTE     Rotate two MSB's into
                                        dividend
020C 26 01              ROL  HIBYTE
020E 26 05              ROL  DIVDND
0210 2A                 ROL  A
0211 06 00              ASL  LOBYTE
0213 26 01              ROL  HIBYTE
0215 26 05              ROL  DIVDND
0217 2A                 ROL  A
0218 06 02              ASL  ROOT       Left-shift current square root
021A A4 02              LDY  ROOT
021C 84 03              STY  DIVISR
021E 06 03              ASL  DIVISR      and double it to form divisor
0220 26 04              ROL  DIVISR+1
0222 C5 04              CMP  DIVISR+1   Dividend greater than divisor?
0224 90 17              BCC  DECCNT
0226 D0 08              BNE  INCSQ
0228 A4 05              LDY  DIVDND
022A C4 03              CPY  DIVISR
022C F0 0F              BEQ  DECCNT     No. Square root bit = 0
022E 90 0D              BCC  DECCNT
0230 E6 02      INCSQ   INC  ROOT       Yes. Square root bit = 1
0232 E6 03              INC  DIVISR      and divisor LSB = 1
0234 48                 PHA             Calculate remainder
0235 98                 TYA
0236 E5 03              SBC  DIVISR
0238 85 05              STA  DIVDND
023A 68                 PLA
023B E5 04              SBC  DIVISR+1
023D CA         DECCNT  DEX             Loop until done
023E D0 Ca              BNE  LOOP
```

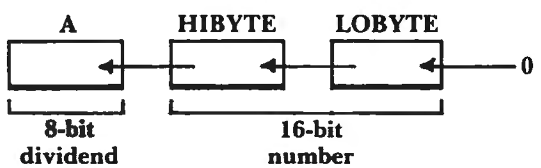


Figure 1. Forming a Dividend in Example 1

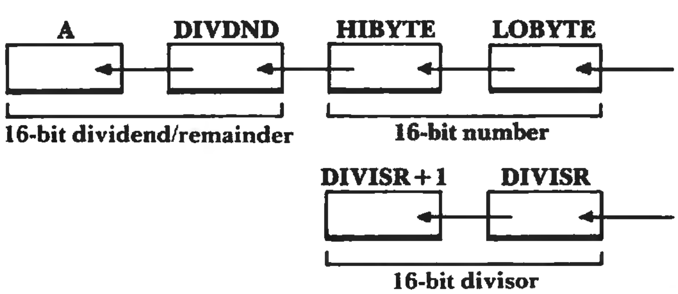


Figure 2. 16-Bit Dividend and Divisor

©

Ideal-Gas Law

Arthur L. McNeill
Department of Chemistry
Seattle University
Seattle, WA 98122

Gases are tricky things to work with. In the case of solids you can weigh them and know how much is present. Liquids are easier to work with because you can not only weigh them but you can also measure their volume and by multiplying by the density find the amount of the material present. With gases the problem is more complicated. To know how much gas you are dealing with, it is necessary to know the pressure of the gas, its volume, its temperature and the number of moles present. R is the Ideal-Gas constant. Luckily a relationship exists between these variables so that any one variable can be determined, provided you know the other three. This relationship is known as the Ideal-Gas law, which is simply stated as follows:

PxV = MxRxT

P is the pressure under which gas exists, **V** is the volume of the gas, **T** is the temperature of the gas and **M** is the moles of the gas (the mole of a gas is the molecular weight of the gas expressed in some units). Example: Nitrogen (mol. wt. = 28.0), therefore, 28.0 g would equal 1.0 gram-mole, or 28.0 lbs. would equal 1.0 lb.-mole. Or if the moles of a gas are known, the grams or lbs. may be found by multiplying by the molecular weight.

This relationship works very well for gases under ordinary pressures, say under a few atmospheres of pressure and at room temperature or higher. These are the conditions that we will be dealing with in the development of this program. There are improvements which can give better results at conditions other than those outlined above. These improvements are the Van der Waal's equation and the Varial equation. These, hopefully, will be dealt with in future articles.

However, when we speak of pressure, temperature, volume, moles and the gas constant, various types of units may be employed. Thus, for the physical scientist he may express the pressure in atmosphere units, the temperature in degrees Celsius (Centigrade), the volume in liters and the moles in gram-moles. The engineer, on the other hand, may express the pressure in lbs./sq. in., the volume in cubic feet, the temperature in degrees Fahrenheit, the moles as pound-moles and the universal gas constant R in the above units. So, if a universal program is to be written, it must take these possibilities into consideration. The numerical value of the Ideal-Gas constant R for scientific data is 0.08205 liter x atm/g-mole x K^0 and that for the engineer units is 10.731 cu. ft. x lbs./sq. in. x lb.-moles x R^0.

Discussion Of The Program

St. No.'s 130-180 (St. No. is the statement number) sets up a small menu so the units used in solving the equation may be either scientific units or engineering units. Once the choice has been typed in the program branches to those locations which will solve the equation in appropriate units.

Thus, if the program branches to St. No. 220, scientific units are used or if branching to St. No. 1220 occurs, engineering units are utilized. Then another choice is offered so that the program can solve for the volume of a gas, or its temperature, or its pressure or the number of moles present provided the other three variables are known. This procedure is outlined in St. Nos. 230-290.

The choice taken above will direct the program via St. No. 300 to locations which will solve the problem and print out the results as follows:

Given the following data:
The volume is 22.654 liters
The temperature is 31.04 deg. C.
The gram-moles present are 1.333
The pressure developed is 1.467 Atm.

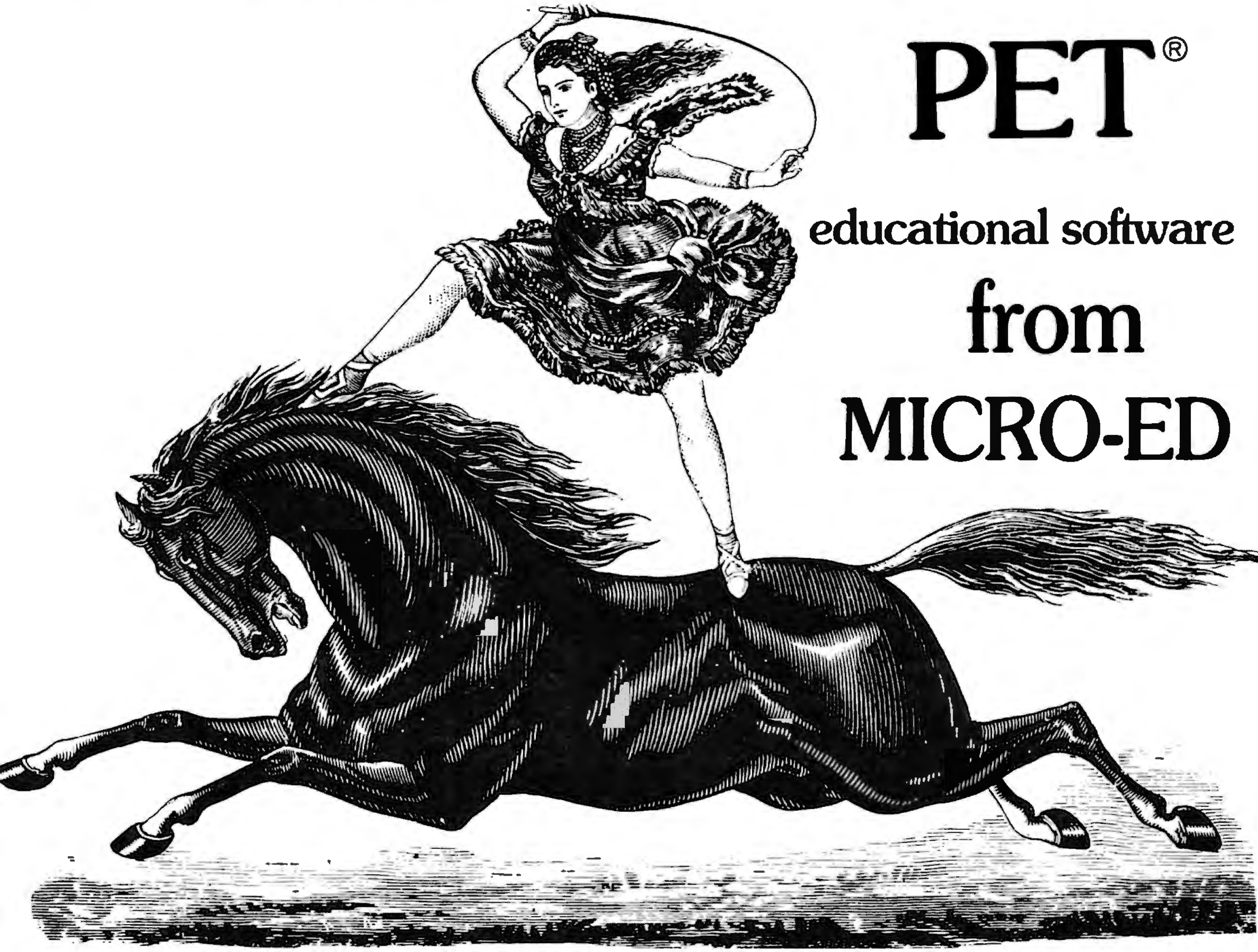
Once the results are printed out, the program jumps to St. No. 990 and inquires if other variables are to be solved for. If so, the program again displays the choices and the program repeats. If engineering units are now to be used, the program is terminated with a "No" to question in St. No. 990 and the program is rerun. The choice between scientific or engineering units is offered again and if engineering units are chosen, choices of variables will again be presented and a print-out like the following will be displayed on the printer:

Given the following data:
The volume is 38.5 cu. ft.
The temperature is 84.5 deg. F.
The lb.-moles are 1.06
The pressure developed is 16.07 lbs./sq. in.

The program is written in Basic, which with very few modifications, can be used with most personal computers. The author used a "PET"* personal computer and a TTY Model 43 printer to process the program.

```
100 OPEN 2,3:CMD 2
110 REM THIS PROGRAM CALCULATES  THE ¬
       ¬VARIABLES  OF THE PERFECT GAS LAW ¬
       ¬GIVEN
120 REM THE NECESSARY DATA AND THE ¬
       ¬BASIC EQUATION, PV=MRT
130 PRINT"DATA MAY BE GIVEN IN THE CGS ¬
       ¬SYSTEM OR IN ENGINEERING UNITS":
       ¬PRINT
140 PRINT"TYPE THE NO. CORRESPONDING TO ¬
       ¬THE WAY THE DATA IS EXPRESSED":
```

Perfectly Balanced



PET®

educational software

from

MICRO-ED

More than 200 instructional programs

Unless otherwise specified, each one can be purchased for $7.95

You may wish to order tapes by the MICRODOZEN. Any twelve $7.95 tapes can be purchased for $84.00.

All programs work with any 8K PET, old or new.

Send for free catalogue:

MICRO-ED, Inc. • P.O. Box 24156 • Minneapolis, MN 55424

or telephone us at (612) 926-2292

```
     ¬PRINT
150 PRINT"THE DATA IS GIVEN AS FOLLOWS":
     ¬PRINT:PRINT
160 PRINT"1.-TEMPERATURE IN DEG. ¬
     ¬CELSIUS,PRESSURE IN ATM. & VOLUME ¬
     ¬IN LITERS":PRINT
170 PRINT"2.-TEMPERATURE IN DEG. F.,
     ¬ PRESSURE IN LBS/IN SQ., VOLUME ¬
     ¬IN CU.FT."
180 INPUT D
190 PRINT :PRINT
200 ON D GOTO 220,1200
210 PRINT
220 PRINT"n̂","↓↓↓↓"
230 PRINT"TYPE THE NUMBER CORRESPONDING ¬
     ¬TO THE VARIABLE TO BE CALCULATED":
     ¬PRINT
240 PRINT,"1. VOLUME
250 PRINT,"2. TEMPERATURE"
260 PRINT,"3. PRESSURE"
270 PRINT,"4. MOLES PRESENT"
280 INPUT N
290 R =.08205:PRINT
300 ON N GOTO 320,460,640,800
310 PRINT
320 PRINT"TYPE THE PRESSURE IN ATM.":
     ¬INPUT P:PRINT
330 PRINT"TYPE THE TEMPERATURE IN ¬
     ¬DEG.C.":INPUT T:PRINT
340 PRINT"TYPE THE NO. OF GRAM-MOLES":
     ¬INPUT M:PRINT
350 V = M*R*(T+273)/P
351 V=INT(V*100)/100
360 CLOSE 2
370 OPEN 2,5:CMD 2
380 PRINT"GIVEN THE FOLLOWING DATA:"
390 PRINT
400 PRINT,"THE PRESSURE IS ";P;"ATM."
410 PRINT,"THE TEMPERATURE IS ";T;"DEG.C
     ¬."
420 PRINT,"THE NO. OF MOLES ARE ";M:
     ¬PRINT:PRINT
430 PRINT"THE VOLUME OF THE GAS ¬
     ¬IS";V;"LITERS"
440 PRINT:PRINT"-----------------------
     ¬----------------------------------
     ¬"
450 GOTO 960
460 PRINT:PRINT
470 PRINT"TYPE THE PRESSURE IN ATM."
480 INPUT P:PRINT
490 PRINT"TYPE THE VOLUME IN LITERS"
500 INPUT V:PRINT
510 PRINT"TYPE THE NO. OF MOLES USED"
520 INPUT M:PRINT
530 T2=273
540 T1=(P*V)/(M*R):T=T1-T2
541 T=INT(T*100)/100
550 CLOSE 2:OPEN 2,5:CMD 2
560 PRINT"GIVEN THE FOLLOWING DATA:":
     ¬PRINT
570 PRINT,"THE PRESSURE IS";P;"ATM."
580 PRINT,"THE VOLUME IS ";V;"LITERS"
590 PRINT,"THE NO. OF MOLES ARE ";M:
     ¬PRINT:PRINT
600 PRINT"THE TEMPERATURE IS ";T;"DEG. ¬
     ¬C."
610 PRINT"---------------------------
     ¬----------------------------------
     ¬-"
620 PRINT:PRINT
630 GOTO 960
640 PRINT"TYPE THE VOLUME IN LITERS"
650 INPUT V:PRINT
660 PRINT"TYPE THE TEMPERATURE IN DEG. ¬
     ¬C."
670 INPUT T:PRINT
680 PRINT"TYPE THE NO. OF G-MOLES USED"
690 INPUT M:PRINT
700 P = M*R*(T+273)/V
701 P=INT(P*1000)/1000
710 CLOSE 2
720 OPEN 2,5:CMD 2
730 PRINT"GIVEN THE FOLLOWING DATA:"
740 PRINT:PRINT,"THE VOLUME IS";V;"LITER
     ¬S"
750 PRINT,"THE TEMPERATURE IS";T;"DEG.C.
     ¬"
760 PRINT,"THE MOLES PRESENT ARE";M:
     ¬PRINT:PRINT
770 PRINT"THE PRESSURE DEVELOPED ¬
     ¬IS";P;"ATM."
780 PRINT:PRINT"-----------------------
     ¬----------------------------------
     ¬-"
790 GOTO 960
800 PRINT"TYPE THE PRESSURE IN ATM."
810 INPUT P:PRINT
820 PRINT"TYPE THE VOLUME IN LITERS"
830 INPUT V:PRINT
840 PRINT"TYPE THE TEMPERATURE IN DEG. ¬
     ¬C."
850 INPUT T:PRINT
860 M = (P*V)/(R*(T+273))
861 M=INT(M*100)/100
870 CLOSE 2
880 OPEN 2,5:CMD 2
890 PRINT"GIVEN THE FOLLOWING DATA:":
     ¬PRINT
900 PRINT,"THE PRESSURE IS";P;"ATMOSPHER
     ¬ES."
910 PRINT,"THE VOLUME IS";V;"LITERS
920 PRINT,"THE TEMPERATURE IS";T;"DEG.C.
     ¬":PRINT:PRINT
930 PRINT"THE NUMBER OF MOLES PRESENT ¬
     ¬ARE";M
940 PRINT:PRINT"-----------------------
     ¬----------------------------------
     ¬-"
950 GOTO 960
960 CLOSE 2
970 OPEN 2,3:CMD 2
980 PRINT"n̂","↓↓↓↓↓↓"
990 PRINT"DO YOU WISH TO SOLVE OTHER ¬
     ¬PROBLEMS? IF SO TYPE YES OTHERWISE
     ¬ NO"
1000 INPUT A$:PRINT
1010 IF A$="YES" THEN 220
1020 PRINT"===========================
     ¬=========="
1030 PRINT"BYE NOW, HAVE A GOOD DAY!"
1040 PRINT"===========================
     ¬=========="
1050 CLOSE 2:END
1200 ON D GOTO 220,1220
1210 PRINT
1220 PRINT"n̂","↓↓↓↓"
1230 PRINT"TYPE THE NUMBER CORRESPONDING
     ¬ TO THE VARIABLE TO BE CALCULATED"
     ¬:PRINT
```

```
1240 PRINT,"1. VOLUME
1250 PRINT,"2. TEMPERATURE"
1260 PRINT,"3. PRESSURE"
1270 PRINT,"4. MOLES PRESENT"
1280 INPUT N
1290 R =10.731:PRINT
1300 ON N GOTO 1320,1460,1640,1800
1320 PRINT"TYPE THE PRESSURE IN ¬
     ¬LBS./IN.SQ.":INPUT P:PRINT
1330 PRINT"TYPE THE TEMPERATURE IN ¬
     ¬DEG.FAHRENHEIT":INPUT T:PRINT
1340 PRINT"TYPE THE NO. OF LBS.-MOLES":
     ¬INPUT M:PRINT
1350 V = M*R*(T+459.690)/P
1351 V=INT(V*100)/100
1360 CLOSE 2
1370 OPEN 2,5:CMD 2
1380 PRINT"GIVEN THE FOLLOWING DATA:"
1390 PRINT
1400 PRINT,"THE PRESSURE IS";P;"LBS/IN.S
     ¬Q."
1410 PRINT,"THE TEMPERATURE IS ";T;"DEG.
     ¬F."
1420 PRINT,"THE NO. OF LBS.-MOLES ARE ¬
     ¬";M:PRINT:PRINT
1430 PRINT"THE VOLUME OF THE GAS ¬
     ¬IS";V;"CU.FT."
1440 PRINT:PRINT"-----------------------
     ¬-----------------------------------
     ¬-"
1450 GOTO 1960
1460 PRINT:PRINT
1470 PRINT"TYPE THE PRESSURE IN ¬
     ¬LBS./SQ.IN."
1480 INPUT P:PRINT
1490 PRINT"TYPE THE VOLUME IN CU. FT."
1500 INPUT V:PRINT
1510 PRINT"TYPE THE NO. OF LBS.-MOLES ¬
     ¬USED"
1520 INPUT M:PRINT
1530 T2=459.69
1540 T1=(P*V)/(M*R):T=T1-T2
1541 T=INT(T*100)/100
1550 CLOSE 2:OPEN 2,5:CMD 2
1560 PRINT"GIVEN THE FOLLOWING DATA:":
     ¬PRINT
1570 PRINT,"THE PRESSURE IS ";P;"LBS./SQ
     ¬.IN."
1580 PRINT,"THE VOLUME IS ";V;"CU.FT."
1590 PRINT,"THE NO. OF LBS.-MOLES ARE ¬
     ¬";M:PRINT:PRINT
1600 PRINT"THE TEMPERATURE IS ";T;"DEG. ¬
     ¬F."
1610 PRINT"----------------------------
     ¬-----------------------------------
     ¬--"
1620 PRINT:PRINT
1630 GOTO 1960
1640 PRINT"TYPE THE VOLUME IN CU.FT."
1650 INPUT V:PRINT
1660 PRINT"TYPE THE TEMPERATURE IN DEG. ¬
     ¬F."
1670 INPUT T:PRINT
1680 PRINT"TYPE THE NO. OF LBS.-MOLES ¬
     ¬USED"
1690 INPUT M:PRINT
1700 P = M*R*(T+459.69)/V
1701 P=INT(P*100)/100
1710 CLOSE 2
1720 OPEN 2,5:CMD 2
1730 PRINT"GIVEN THE FOLLOWING DATA:"
1740 PRINT:PRINT,"THE VOLUME IS";V;"CU.F
     ¬T."
1750 PRINT,"THE TEMPERATURE IS";T;"DEG.F
     ¬."
1760 PRINT,"THE LB-MOLES PRESENT ARE";M:
     ¬PRINT:PRINT
1770 PRINT"THE PRESSURE DEVELOPED ¬
     ¬IS";P;"LBS./SQ.IN."
1780 PRINT:PRINT"-----------------------
     ¬-----------------------------------
     ¬--"
1790 GOTO 1960
1800 PRINT"TYPE THE PRESSURE IN ¬
     ¬LBS/SQ.IN."
1810 INPUT P:PRINT
1820 PRINT"TYPE THE VOLUME IN CU.FT."
1830 INPUT V:PRINT
1840 PRINT"TYPE THE TEMPERATURE IN DEG. ¬
     ¬F."
1850 INPUT T:PRINT
1860 M = (P*V)/(R*(T+459.69))
1861 M=INT(M*100)/100
1870 CLOSE 2
1880 OPEN 2,5:CMD 2
1890 PRINT"GIVEN THE FOLLOWING DATA:":
     ¬PRINT
1900 PRINT,"THE PRESSURE IS";P;"LBS./SQ.
     ¬IN."
1910 PRINT,"THE VOLUME IS";V;"CU.FT.
1920 PRINT,"THE TEMPERATURE IS";T;"DEG.F
     ¬.":PRINT:PRINT
1930 PRINT"THE NUMBER OF LB-MOLES ¬
     ¬PRESENT ARE";M
1940 PRINT:PRINT"-----------------------
     ¬-----------------------------------
     ¬--"
1950 GOTO 1960
1960 CLOSE 2
1970 OPEN 2,3:CMD 2
1980 PRINT"ħ","↓↓↓↓↓↓"
1990 PRINT"DO YOU WISH TO SOLVE OTHER ¬
     ¬PROBLEMS? IF SO TYPE YES OTHERWISE
     ¬ NO"
2000 INPUT A$:PRINT
2010 IF A$="YES" THEN 1220
2020 PRINT"===========================
     ¬=========="
2030 PRINT"BYE NOW, HAVE A GOOD DAY!"
2040 PRINT"===========================
     ¬=========="
2050 CLOSE 2:END
READY.
```

```
PRINT-OUT - IDEAL GAS LAW


GIVEN THE FOLLOWING DATA:

          THE PRESSURE IS  2.12 ATM.
          THE TEMPERATURE IS  36 DEG.C.
          THE NO. OF MOLES ARE  3.52


THE VOLUME OF THE GAS IS 42.09 LITERS

---------------------------------------------

GIVEN THE FOLLOWING DATA:

          THE PRESSURE IS 2.87 ATM.
          THE VOLUME IS  50.6 LITERS
          THE NO. OF MOLES ARE  4.16


THE TEMPERATURE IS  152.46 DEG. C.
---------------------------------------------



GIVEN THE FOLLOWING DATA:

          THE VOLUME IS 150.2 LITERS
          THE TEMPERATURE IS 52 DEG.C.
          THE MOLES PRESENT ARE 6.45


THE PRESSURE DEVELOPED IS 1.145 ATM.

---------------------------------------------

GIVEN THE FOLLOWING DATA:

          THE PRESSURE IS 1.54 ATMOSPHERES.
          THE VOLUME IS 30.65 LITERS
          THE TEMPERATURE IS 25.9 DEG.C.


THE NUMBER OF MOLES PRESENT ARE 1.92

---------------------------------------------

GIVEN THE FOLLOWING DATA:

          THE PRESSURE IS 15.9 LBS/IN.SQ.
          THE TEMPERATURE IS  44 DEG.F.
          THE NO. OF LBS.-MOLES ARE  1.22


THE VOLUME OF THE GAS IS 414.73 CU.FT.
```

©

THE apple GAZETTE

Apple II High Resolution Character Generator

Peter Gehris and Ken Reinert
Wyomissing, PA

Are you dissatisfied with the small size of the characters on the video screen? Do you have need for a character generator for TV or videotape recording? Would you like to display your own shapes in variable size and position on the screen? Here is a way to achieve these goals using the shape table provision of the APPLE II. The amazing aspects of this program are: (1) you can vary the size of each shape; (2) you can change or add shapes as desired; (3) you can rotate the shapes on the screen even to the point of displaying them upside down or backwards; and (4) you can customize this program to your own programs to yield variable letter forms, sizes and positions for neater looking displays of data and text. The program which positions and draws each character uses high resolution graphics on the video monitor. A copy of the program is in diagram 2. All of the keyboard characters are available except the ›@› which is used as an underline. The instructions to draw each character are stored in a shape table. The creation of the shape table, which is in Chapter 9 of the APPLESOFT II Basic Programming Reference Manual, will not be discussed here.

The Shape Table

In diagram 1, you will see the hexadecimal codes for a shape table beginning at address 37000 (hex 9088) and ending at address 37944 (hex 9440). Please note these addresses are for a 48K system. This table provides the shapes for all characters on the keyboard, except the ›@› key, which is an underline. The shape table should fit right below DOS. HIMEM must then be set at the beginning of the table to protect it from being written over by APPLESOFT variables.

Use this chart to see where the table should be loaded into your system:

Memory size	Address dec.	hex.	HIMEM:	POKE 232,	POKE 233,
32K	21000	$5208	21000	8	82
36K	25000	$61A8	25000	168	97
48K	37000	$9088	37000	136	144

(NOTE: the POKEs will be explained later.)

Loading The Shape Table

By now, the shapes should be defined and converted into hexadecimal code. Now, the codes can be typed into the memory. (It is best if two people do this: one to read the codes and another to type them in.) First, get into the monitor by typing CALL -151. Then, type the address of the shape table: 9088 (for a 48K system) followed by a colon and up to 127 hex codes. The start address will vary according to system size (see chart). The display should look like this:

```
9088: 3B 00 78 00 7A 00 81 00 ...
```

(see page 44 of the APPLE II Reference Manual.) After pressing ‹RETURN›, type the new address (start address plus the number of codes entered) followed by a colon and more hex codes.

To save the shape table, type: 9088,9437W (for tape) or BSAVE (name), A$9088,L$3B0 (for disk). (These addresses are for the shape table listed.) Then, to load it back into memory, type: 9088,9437R (for tape) or BLOAD (name) (for disk), and the table will load into memory, ready for use.

Shape Table Demo Program

The list of the program in diagram 2 will demonstrate the use of the shape table in developing characters on the video screen. Note that for this demo you can control the rotation, scale, position and space between letters through input. We have used a GOSUB in line 60 and a RETURN to show that this can be used as a subroutine within a program. The variables P$, A, B, H, V and Z can be defined by ways other than input prior to each use of the subroutine. The screen is divided into 31 horizontal positions and 17 vertical positions. The rotation angles start at 0 for an upright character. Increasing the rotation equal to 16 will set the shape on its right side (90 degrees); 32 will invert the character 180 degrees, etc. A rotation of 64 will return the shape to an upright position.

A scale of one prints the characters at their normal size (double the size of text). A scale of two doubles the size, a scale of three triples the size, etc. A normal hi-res screen has 280 horizontal dots and

192 vertical dots. Line 1030 converts the H,V coordinates of the 31x17 screen to the X,Y coordinates of the 280x192 hi-res screen. The FOR-NEXT loop at line 1050 prints out the string, one character at a time. Each character is picked out of the string by the MID$ function. That character's ASCII code, minus 31, is used to identify which shape number to draw. Line 1060 increments the X-coordinate to the new position to print the next character in the string.

Line 20 tells the computer where the shape table begins. This is where POKE 232 and POKE 233 are used. Line 40 sets himem to the beginning of the table, so the table is not written over by variables.

Diagram 3 shows another program which uses the secondary page (page 2) of hi-res graphics, which is available only on a 36K or a 48K system. The variables are defined within the program instead of by using input. The routine to enlarge the letters is located at lines 1000-1020.

Customizing The Program

The routine for enlarging the characters can be added to most any program. However, the display cannot be output to a printer, since it is in the hi-res graphics mode. Just remember that there are only 17 lines of 31 characters per screen page (at a scale of 1). With this in mind, you should be up and running with the enlarged characters in no time.

Shapes can be added to the shape table; however only the more advanced programmers should attempt this, as it does require moving blocks of the shape table around and manipulating the shape table index. More on this can be found in Chapter 9 of the APPLESOFT Manual.

Summary

Without a doubt, the work for this program involves the construction of the shape table. Many hours can be spent drawing, plotting vectors, coding in binary and hexadecimal and typing the codes. The program to use the shape table is fairly simple. Uses for the creation of shapes of both usual and unusual types are numerous. Besides, generating the normal alphanumeric and graphics characters, special characters can be created for math/science, foreign languages, etc., and the basic ideas can also be applied to music, art and in creating games.

```
APPLE II HI RESOLUTION GRAPHICS GENERATOR
P. GEHRIS AND K. REINERT   4/8/81


1  REM
2  REM      HI RES DEMO PROGRAM #1
3  REM          FOR 32K SYSTEM
4  REM
10  REM   POKE SHAPE TABLE ADDRESS
20  POKE 232,8: POKE 233,82
30  REM   SET HIMEM TO BEGINNING OF TABLE
40  HIMEM: 21000
50  REM   INPUT DATA
60  TEXT : HOME : INPUT "ENTER STRING : ";P$
70  INPUT "ENTER ROTATION : ";A
80  INPUT "ENTER SCALE : ";B
90  INPUT "ENTER HORIZONTAL POSITION : ";H
100  INPUT "ENTER VERTICAL POSITION : ";V
110  INPUT "ENTER SPACES BETWEEN LETTERS";Z
120  REM   CALL SUB. TO PRINT STRING
130  GOSUB 1000
140  REM   DELAY, THEN CLEAR SCREEN
150  FOR T = 1 TO 7500: NEXT T: HOME : PRINT  CHR$ (7): GOTO 50

1000  REM    CLEAR SCREEN OF TEXT, PRODUCES FULL PAGE OF HI-RES
      GRAPHICS,SETS COLOR, ROT-ATION AND SCALE
1010  HOME : HGR : POKE 49234,0: HCOLOR= 3: ROT= A: SCALE= B
1020  REM   COMPUTE LOCATION TO PRINT STRING
1030 X = 9 * H - 7:Y =  INT (11 * V - 8):P =  LEN (P$)
1040  REM   PRINT STRING ONE CHARACTER AT A TIME
1050  FOR I = 1 TO P: DRAW  ASC ( MID$ (P$,I,1)) - 31 AT X,Y
1060 X = X + ((9 * B) + Z)
1070  NEXT I
1080  RETURN
```

```
APPLE II HI RESOLUTION GRAPHICS GENERATOR
P. GEHRIS AND K. REINERT   4/8/81

1  REM
2  REM       HI RES DEMO PROGRAM #2
3  REM           FOR 48K SYSTEM
4  REM
10  POKE 232,136: POKE 233,144: REM  POKES SHAPE TABLE STARTING
     ADDRESS
20  HIMEM: 37000: REM   SETS HIMEM TO BEGINNING OF SHAPE TABLE

30  IF  PEEK (37000) <  > 59 THEN  PRINT  CHR$ (4)"BLOAD SHAPES
    ": REM  LOADS SHAPE TABLE INTO MEMORY IF NOT ALREADY IN ME
    MORY
40  HOME : HGR2 : POKE 49234,0: HCOLOR= 3: ROT= 0: SCALE= 1: REM
      SETS HI-RES GRAPHICS PARAMETERS; USES PAGE 2 OF HI-RES M
    EMORY
50 P$ = "THIS DEMONSTRATION PROGRAM":H = 4:V = 2: GOSUB 1000
60 P$ = "USES THE HI-RES":H = 9:V = 5: GOSUB 1000
70 P$ = "CHARACTER GENERATOR":H = 7:V = 8: GOSUB 1000
80 P$ = "AND SHAPE TABLE":H = 9:V = 11: GOSUB 1000
90 P$ = "TO PRINT THIS":H = 10:V = 14: GOSUB 1000
100 P$ = "DISPLAY":H = 13:V = 17: GOSUB 1000
110  FOR W = 1 TO 5000: NEXT
120  TEXT : HOME : END
1000 X = 9 * H - 7:Y =  INT (11 * V - 8):P =  LEN (P$)
1010  FOR I = 1 TO P: DRAW  ASC ( MID$ (P$,I,1)) - 31 AT X,Y:X =
    X + 9
1020  FOR T = 1 TO 50: NEXT T: NEXT I: RETURN
```

```
9088- 3B 00 78 00 7A 00 81 00
9090- 88 00 99 00 AD 00 C3 00
9098- D6 00 DA 00 E5 00 EE 00
90A0- 01 01 0C 01 14 01 1A 01
90A8- 21 01 2B 01 3C 01 48 01
90B0- 59 01 6A 01 7B 01 8B 01
90B8- 9F 01 AB 01 BE 01 D3 01
90C0- DA 01 E3 01 EF 01 F9 01
90C8- 03 02 0E 02 15 02 27 02
90D0- 3B 02 4A 02 5A 02 6C 02
90D8- 7B 02 8D 02 9E 02 AA 02
90E0- B7 02 C8 02 D1 02 E3 02
90E8- F6 02 06 03 16 03 28 03
90F0- 3C 03 4D 03 57 03 65 03
90F8- 73 03 85 03 95 03 A1 03
9100- 01 00 49 36 36 36 96 06
9108- 00 31 36 4D 21 24 04 00
9110- 89 36 36 36 6E 21 24 24
9118- 24 95 1F FF 96 0D 6D 05
9120- 00 49 36 36 36 36 26 D8
9128- AB 6D 2D 20 E4 FF 3F 20
9130- 0C 6D AD 05 00 32 0E 05
9138- 20 07 28 2D 2D 36 1E 1E
9140- 1E 1E 1E 1E 4D 21 0C 15
9148- F6 05 00 12 76 0E 0E 0E
9150- 0D 16 1C 1E 3F 3F 20 0C
9158- 0C 08 24 1C 3F 00 49 36
9160- 36 00 49 09 1E 1E 1E 36
9168- 76 0E 0E 06 00 A9 15 15
9170- 36 F6 1E 1E 06 00 72 0E
9178- 8E 71 0E DE 23 24 24 24
9180- 6C 11 17 17 D7 BA 17 07
9188- 00 92 2A 2D 2D B5 DA 23
9190- 24 20 24 00 49 92 92 37
9198- 35 1E 06 00 92 52 2D 2D
91A0- 5A 00 92 92 52 29 3E 04
91A8- 00 49 49 F2 1E 1E 1E 1E
91B0- 1E 06 00 12 36 36 76 0E
91B8- 2D 05 28 20 24 24 1C 1C
91C0- 3F 17 05 00 2D 35 36 36
91C8- 36 36 3F 6F 09 2D 05 00
91D0- 12 0C 0C 2D 2D 32 1E 1E
91D8- 1E 1E 1E 1E 2E 2D 2D 2D
91E0- 00 2A 28 2D AD 36 1E 3F
91E8- 37 49 31 36 1E 3F 3F 07
91F0- 20 00 49 29 36 36 2E 96
91F8- 1A 24 24 3F 3F 27 05 28
9200- 28 28 00 2D 2D 2D DE DB
9208- 33 36 2D 2D AD 36 F6 3F
9210- 3F 27 00 92 32 36 76 2D
9218- 2D 05 20 24 1C 3F 3F 07
9220- 20 64 2D 2D 15 05 00 2D
9228- 2D 2D 36 1E 1E 1E 1E 1E
9230- 36 05 00 32 B6 36 76 2D
9238- 2D 05 20 24 04 20 E4 3F
9240- 3F 96 2A 2D 2D 00 02 36
9248- 96 12 0E 2D 2D 05 20 24
9250- 24 24 1C 3F 3F 16 12 2D
```

```
9258- 2D 05 00 52 29 3E 96 35
9260- 37 00 52 29 3E 96 35 77
9268- 1E 06 00 49 09 1E 1E 1E
9270- 1E 0E 0E 0E 0E 06 00 12
9278- 2D 2D 2D 96 3A 3F 3F 3F
9280- 00 A9 15 15 15 1E 1E 1E
9288- 1E 06 00 2A 28 2D AD 36
9290- 1E BF 36 16 05 00 93 2D
9298- 2D 2D 2D 05 00 32 36 36
92A0- 36 6E 49 21 24 24 24 E4
92A8- 3F 3F 96 2A 2D 2D 00 36
92B0- 36 36 36 2E 2D 2D 05 20
92B8- 24 04 20 E4 3F 3F 96 2A
92C0- 2D 2D 00 29 2D AD B6 92
92C8- F6 3F 3F 07 20 24 24 24
92D0- 04 00 2D 2D 15 15 36 36
92D8- F6 1E 3F 3F 24 24 24 24
92E0- 24 00 2D 2D 2D DE DB 33
92E8- 36 2D 2D DE 1B 36 36 2D
92F0- 2D 2D 05 00 2D 2D 2D DE
92F8- DB 33 36 2D 2D DE 1B 36
9300- 36 05 00 29 2D AD DF 92
9308- 2A AD 36 1E 3F 3F 07 20
9310- 24 24 24 04 00 36 36 36
9318- 36 6E 49 21 24 24 3F 3F
9320- 67 49 21 24 04 00 29 2D
9328- F5 33 36 36 36 06 3F 4D
9330- 2D 00 49 49 36 36 36 36
9338- 1E 3F 3F 07 20 04 00 36
9340- 36 36 36 6E 49 E1 1C 1C
9348- 1C 1C 0C 0C 0C 0C 05 00
9350- 36 36 36 36 2E 2D 2D 2D
9358- 00 36 36 36 36 6E 49 21
9360- 24 24 24 24 17 17 17 07
9368- 38 28 00 36 36 36 36 6E
9370- 49 21 24 24 24 24 DF 9B
9378- 15 15 15 15 05 00 29 2D
9380- AD 36 36 36 F6 3F 3F 07
9388- 20 24 24 24 04 00 2D 2D
9390- AD 36 1E 3F 3F 04 C0 30
9398- 36 36 36 36 05 00 32 36
93A0- 36 36 0E 2D 6D 1C 07 68
93A8- 24 24 24 1C 3F 3F 07 00
93B0- 36 36 36 36 6E 49 E1 1C
93B8- 1C 1C 07 28 2D 2D 20 E4
93C0- 3F 3F 07 00 29 2D AD DF
93C8- DB 36 0E 2D 2D 15 36 F6
93D0- 3F 3F 07 28 00 2D 2D 2D
93D8- DE 33 36 36 36 36 00 36
93E0- 36 36 36 0E 2D 2D 05 20
93E8- 24 24 24 24 00 36 36 36
93F0- 0E 0E 0E 05 28 28 20 24
93F8- 24 24 00 36 36 36 36 0E
9400- 2D 20 24 24 24 6C 31 36
9408- 36 36 F6 3F 00 76 0E 96
9410- 1E 1E 6E 49 21 1C 1C 1C
9418- 64 0C 0C 24 00 76 0E 16
9420- 92 0A 24 24 64 0C 0C 24
9428- 00 2D 2D 2D 36 1E 1E 1E
9430- 1E 1E 1E 2E 2D 2D 2D 00
```

©

How Do I Fit A 16K Program Into A 6K Space?

(Simple – You Don't)

J. F Johnson
University of Notre Dame
Dept. of Chemistry
Notre Dame, IN

When an Apple II Plus is turned on, default values are assigned to a region in random access memory which is used to control certain system functions. Examples of these functions include system machine language programs, "stacks", input lines by programs, and certain visual display modes of the Apple. The first 2048 bytes (or pages 0 through 7, each page comprising 256 bytes) are reserved for these functions, with the balance of RAM utilized for BASIC, machine language programs, binary data, or DOS. Figure 1 depicts the outlay of user-accessible memory assuming a 48K Apple, with both HI-RES pages also included.

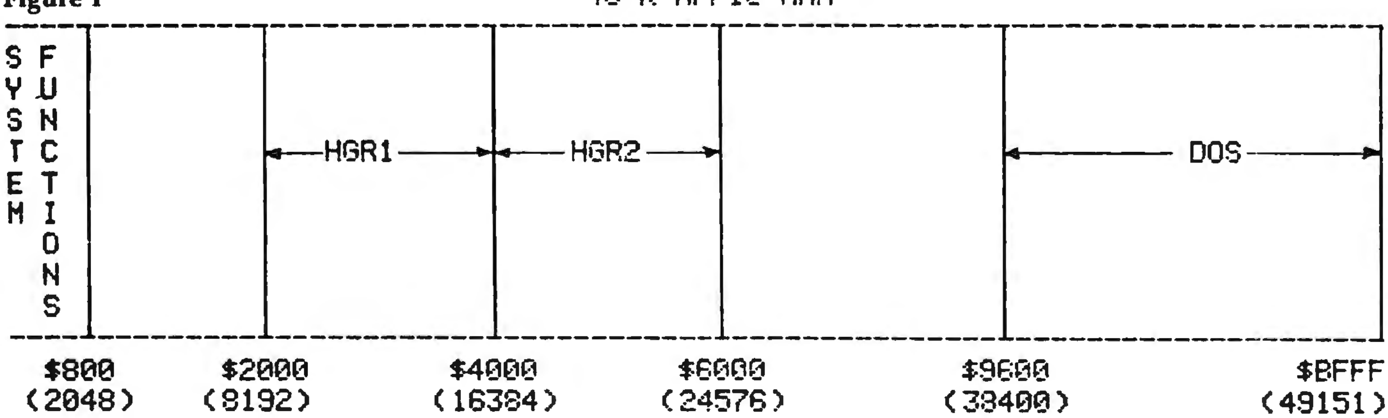
Other areas of RAM may also be used by peripheral devices or functions that are included in Applesoft (the version of BASIC designed for the Apple). If the Apple is interfaced to a disk drive, the disk operating system (DOS) is loaded automatically into the top portion of RAM (38400K-49151K on a 48K Apple; a corresponding upper RAM region is used on Apples with less memory). This upper 10K region can be used by a BASIC program if your Apple is cassette supported. The contiguous portion of memory from 8192K-24576K however can generate frustrating problems. If your program exceeds 6K and extends into the first or second HI-RES page, this problem becomes self-evident when the program attempts to use a "BASIC-loaded" HI-RES page.

For example, the Apple by default starts loading a BASIC program at $800 (2048k). And of course one of the excellent features of the Apple is the use of the HI-RES pages for animation or detailed graphics effects. However, when either HI-RES page is used from BASIC (by the use of either the HGR or HGR2 command), the respective HI-RES region in memory is literally filled with zeroes. If your program is larger than 6K bytes (that region from 2K-8K), then the portion of BASIC extending into the first HI-RES page memory region will be destroyed upon use of the HGR command since the 8K-16K region will be "erased". An analogous situation occurs if the program extends into the 16K-24K region of RAM and the HGR2 command is used. A solution to this problem involves loading the BASIC program into a different region of RAM, and hence overriding certain default values that the Apple assigns when it is turned on.

There are two memory locations in the zero page of Apple's RAM which dictate where BASIC programs are loaded (See Applesoft Memory Map (Page 0) by Jim Butterfield, Issue 6 of **COMPUTE!** or page 140 in the Applesoft BASIC Programming Reference Manual). The pointer to the start of a BASIC program is comprised of the most significant byte 104 ($68) and the least significant byte 103 ($67), and their contents may be changed by using the POKE command. By POKEing 104 and 103 prior to loading a program, the portion of RAM used for BASIC storage can be altered.

Most of my programs occupy about 16K of RAM, require the first HI-RES page for graphics, and operate on a 32K diskless Apple II Plus. It should now be recognized that if the pointer to the beginning of the program (locations 104 and 103) is left at the default value of 2048 ($800), approximately the last 10K of my program would be destroyed upon usage of the first HI-RES page. The simplest solution is to load the BASIC program immediately above this HI-RES page in memory, which can be accommmplished on either a cassette or disk-supported machine, but must be done prior to loading the program.

Figure 1 48 K Apple RAM



For cassette supported systems, after turning on your Apple type the following, then press return.

POKE 104,64:POKE 103,1:POKE 16384,0 ‹RET›

LOADing of a BASIC program now starts at decimal location 16385 (= 256*64+1; the most significant byte is multiplied by 256 and added to the least significant byte for most pointers), and can occupy the upper 16K of RAM (16385K-32768K). The byte immediately before the memory location where LOADing is initiated must be a zero, hence location 16384 contains a zero. (To change this pointer to any RAM location divide the chosen decimal value by 256, POKE the integer portion of the answer into 104 and POKE the remainder into 103.)

The identical results can be obtained on a 48K disk supported Apple, using a slightly more sophisticated method. An EXEC file may be created, which when EXECed by the HELLO program initializes the pointer to the beginning of the program and then loads the program. The following program creates the EXEC file "LOADER".

```
10 D$ = CHR$(4)
20 PRINT D$"OPEN LOADER"
30 PRINT D$"WRITE LOADER"
40 PRINT "POKE 104,64"
50 PRINT "POKE 103,1"
60 PRINT "POKE 16384,0"
70 PRINT "LOAD TITRATION"
80 PRINT "CLOSE LOADER"
```

The following HELLO program would then EXEC the file LOADER.

```
10 D$ = CHR$(4)
20 PRINT D$"EXEC LOADER"
```

The same diskette must of course contain the HELLO program (which is run when the system is booted), the BASIC program "TITRATION", and the EXEC file "LOADER". The use of an EXEC file saves the tedium of POKEing locations from the keyboard, followed by a "LOAD TITRATION" DOS command (which gives the same results but is considered less time-efficient). This now BASIC-vacated region from 2K-8K usually stores shapes which are displayed on the first HI-RES page under control of the newly located BASIC program. ©

Editor's Note: Here's the first in a series of assembler programs to enhance your Integer BASIC. RCL

Ever Expanding Apple Power

Mitchell Bushin
Scarsdale, NY

I am an Apple II owner who found Apple Integer Basic a rather limited language. In order to improve the language's power, I have written assembler programs to "attach" to Integer Basic programs.

The first of my sample programs is an idea stolen from Wang Labs. They have a row of Special Function keys on their machine. These keys can be used to allow a user to type a whole word in answer to an input statement by touching one key. I liked the feature and wrote a program that allows an Apple-ite (Apple-user) to use the number keys as special function keys.

This is the Basic part:

```
1   w=0: rem this must be the first statement
100 rem this is a piece of a home accounting program
110 respond=2048: rem address of routine
500 call -936: rem clear screen
510 vtab 10:print "commment on expense:"
520 call respond: if w#1 or w#2 or w#3 then goto 600
530 dim ans$(30): rem the answer we want
540 if w#1 then goto 550:ans$="oil":s=24
550 if w#2 then goto 560:ans$="petty-cash refill":
    s=36
560 if w#3 then goto 570:ans$="electric":s=29
570 vtab 10:tab 20:print ans$: rem automatic ans is
    printed
580 vtab 10:tab s:input a$: rem wait for return to
    continue
590 goto 610
600 vtab 10:tab 20:input ans$
610 rem rest of program
```

Together with the assembler program, this program types an answer automatically when a pseudo-S.F. key is hit. There can be up to 9 different automatic answers. This is good for an office Apple that runs a program that would normally have a specific number typed to give an answer. In addition, with the S.F. keys, hitting the space bar allows the Apple-ite to type in any answer he wants.

The assembler part:

```
loc in hex
800     JSR    $FF4A    SAVE REG.
803     BIT    $C000    IS KEY PRESSED
        BPL    $0803    NO-GOTO 803
        LDA    $C000    WHAT IS CODE
        CMP    #$BA     IS IT A NUMBER
        BCS    $0813    NO-GOTO 813
```

```
          AND    #$0F      WHAT IS THE
                           NUMBER
          BPL    $0815     GOTO 815
813       LDA    #$00      LET W BE ZERO
815       LDY    #$04      W IS THE 4TH BYTE
                           IN VARIABLE LIST
          STA    (4A),Y    4A IS ADDR. OR VAR.
                           LIST
          BIT    $C010     RESET KEY STROBE
          JMP    $FF3F     RETURN REGISTERS
81F
```

After loading the assembler program set lomem so that it does not interfere with the variable list. Lomem:2079 is good enough. This program is portable and can be placed anywhere in memory. ©

THE ATARI® GAZETTE

A Tutorial

Plotting In Atari Basic

N. L. Wheeler
Charleston, WV

This article gives a program for making (x,y) plots on the ATARI, and shows how to scale data for plotting on any computer. By changing only lines 20 and 25, you can plot any function that you can write in BASIC.

Perhaps your first experience with plotting on the ATARI was like mine. I set out to make a straight line (what could be simpler?). Six hours later I finally had a plot, but the thrill was gone. Two types of errors occured. First frequent occurrences of Error 141 "value out of range." Then the line did not slope in the direction that it should. The program listed here solves these problems and includes a general purpose subprogram for scaling and plotting data. In fact, the problem of wrong slope is solved simply by saying PLOT X,Y-GRAPHICS-Y instead of PLOT X,Y, where YGRAPHICS is the largest Y value that can be plotted in the graphics mode which you are using.

Before explaining the program, some words about plotting. Whether you are using a personal computer, or a piece of paper, the concepts are the same. You must scale your data to fit the medium where you plan to plot it. To illustrate this, suppose you are drawing a map to scale and you wish to represent a road 20 miles long with a landmark at mile 15, but you have to fit this on a 4 inch piece of paper. Here's one way you might solve the problem:

0 5 10 15 20 Miles

0 1 2 3 4 Inches

To find the SCALE, take (20–0)/4–0) = 5 miles/inch. To calculate where, in inches, to place 15 miles, calculate (15–10) miles / (5 miles/inch) = 3 inches. This is easy, partly because we have specific numbers to work with. If you do this process frequently, with different mileages and inches, you might develop the following formula:

SCALE = (LARGEST –SMALLEST values of real data) / (LARGEST –SMALLEST value of plot medium).

Any point to be plotted would be placed at point M in the plot medium, where M = SMALLEST value of plot medium (usually 0) + (point to be plotted – SMALLEST value of real data)/SCALE.

Further complications in computer plotting are that we don't know the smallest and largest values if we are using the computer to generate the points to plot, so we have to calculate the SMALLEST and LARGEST values. Also, we must use a slightly bigger number than SCALE to prevent roundoff error. (I multiply SCALE by 1.01).

Here then are the steps to go through in producing a computer plot. The numbers correspond to comments in the program.

1. Choose the graphics mode and determine the largest X and Y values that you can plot in this mode. For the ATARI these values are 159 and 79 for GRAPHICS 7 and 319 and 159 for GRAPHICS 8. (Always use 1 less than the value in the manual).

2. Choose the number of points to plot and write the DIM statement.

3. Generate the points to be plotted and store them in an X and Y array. Thus X(1) will be plotted vs. Y(1), etc.

4. Find the largest and smallest values of your X and Y values.

5. Determine the scale as described above, and multiply by a small fudge factor.

6. Do the actual plotting. First you must scale each x and y value, then plot them.

7. Put in axes and label them. Because labeling the Y axis requires PEEK and POKE, I chose not to illustrate it in this tutorial. I used the text window to label the X axis.

The following program uses these techniques to plot the sine of X in radians. You can easily

ATARI 400 AND 800 OWNERS:

ATARI is a registered trademark of Warner Communications

IT'S FREE!

The Mosaic catalog.

That's right — it's free. In it you'll find the latest in software and hardware available for Atari computers. Just let us know where to send it and it's yours.

Atari computers are the most advanced on the market. As an Atari owner you recognized that and expect the best for your computer. At Mosaic, Atari computers are our only concern. It's our specialty.

Because of our complete understanding of the Atari computer you can count on our products. We require the best materials, we have tough quality control and we listen.

At Mosaic, we are continually developing and researching new products to give you the best as well as the latest in Atari products. So don't miss out on the best for your Atari. Write today for the Mosaic Electronics catalog.

For Example:

Frankly, our 32K board is the only choice for serious Atari owners. The advantages of our board over others is as obvious as the Atari computer's superiority over its competitors'.

We designed our 32K board with full flexibility in mind because Atari owners should not be restricted to one memory configuration. Our 32K board is built with the same flexibility as Atari memory boards. The Mosaic 32K RAM can be plugged into any RAM slot along side any other Atari memory board without damaging your computer. Other boards can't make the same claim.

Because the Mosaic 32K RAM is designed with the latest techniques you can be sure it will provide years of satisfaction without becoming incompatible with future products. We're so sure our 32K RAM board is the best available we guarantee it the longest . . . one full year.

Some people believe memory access speed is an important consideration. While some boards may be slower, no one can make a memory board for the Atari computer that is accessed faster than the Mosaic 32K RAM board.

The Mosaic 32K RAM. It's the board you've been waiting for. Designed to safely take advantage of the built-in capabilities of the Atari computer. Better built, higher quality, superior design, and the best guarantee. What more needs to be said?

Write for the location of your nearest Mosaic dealer.

Memory Configuration	MOSAIC	The Competition
Empty / 32K / 16K	48K RAM 40K With BASIC Cartridge	48K RAM 40K With BASIC Cartridge
Empty / 32K / 8K	40K RAM	**Danger!** This Configuration Can Damage Computer
Empty / 16K / 32K	48K RAM 40K With BASIC Cartridge	**Danger!** This Configuration Can Damage Computer
Empty / 8K / 32K	40K RAM	**Danger!** This Configuration Can Damage Computer
8K / 32K / 8K	48K RAM 40K With BASIC Cartridge	**Danger!** This Configuration Can Damage Computer
Empty / Companion / 32K	32K RAM	**Danger!** This Configuration Can Damage Computer

Yes, send me the Mosaic catalog with the latest in Atari accessories.

Name ______________________

Address ______________________

City/State ______________ Zip ________

MOSAIC
ELECTRONICS

Mail to: Mosaic Electronics ■ P.O. Box 748
Oregon City, Oregon 97045 ■ (503) 655-9574

adapt it to any function by changing lines 20 and 25. Line 20 gives the range of values you want X to take on and line 25 describes how to calculate Y from X.

The program also illustrates what professional programmers call "defensive code". If you try to calculate more values than you have specified with MXPTS, it will warn you. If for some reason, you do try to plot outside the valid values (which shouldn't happen if you type carefully), it will tell you.

What might you like to add to this or change? In GRAPHICS 7, you can have four colors. You could introduce a new variable Z(I), where Z stored the color of the point (0 to 3). Then add a line 5125 COLOR Z(I). I used this program to plot the maximum scores for each level of Star Raiders, but I have to report that it did not help my game.

NOTE TO APPLE OWNERS: A friend who owns an Apple tried an earlier version of this program and reported that it worked. YGRAPHICS should be 191, and XGRAPHICS 279. You may have to change the X labeling in lines 5156 to 5168.

```
1 GRAPHICS 0:PRINT :PRINT "A GENERAL PLOTTING PROGRAM":PRINT :PRINT "CHANGE LINE
 20 TO SPECIFY RANGE OF X "
2 PRINT "AND LINE 25 TO SPECIFY CURVE.":PRINT "ENTER TITLE AT 19":PRINT :PRINT "
LINE 25 MUST BE A VALID EQUATION"
3 PRINT "DON'T DIVIDE BY ZERO OR ANYTHING":PRINT "LIKE THAT":PRINT :PRINT "WHEN
YOU HIT ENTER, WE'LL DO A SAMPLE"
4 DIM Z$(1):INPUT Z$
5 REM *** STEP1 *** Choose the graphics mode, and determine maximum values you c
an plot based on mode
6 GRAPHICS 8:XGRAPHICS=319:YGRAPHICS=159:REM . Use either statement 6 or 7
7 GRAPHICS 7:XGRAPHICS=159:YGRAPHICS=79
8 REM *** STEP2 *** Choose the number of points to plot and write DIM statement
9 MXPTS=200:BAD=0
10 DIM X(MXPTS),Y(MXPTS)
15 REM *** STEP3 *** Generate points to plot, store in an X and Y array.
16 REM . Count the points with I as you go, be sure I does not exceed MXPTS set
earlier.
17 DIM LEGEND$(60),LABEL$(20):REM used in line 5160 to approximate X axis label
18 I=0:DIM TITLE$(40):REM .Initialize I, counter for the arrays X and Y.
19 TITLE$="            A SINE WAVE ":REM Leave some spaces at front to center
20 FOR X=0 TO 2*3.1416 STEP 0.2:REM from 0 to 2  PI
25 Y=SIN(X)
31 IF I=MXPTS THEN PRINT "> YOU MUST INCREASE MXPTS IN LINE 8,PLOTTING CONTINUES
 ":FOR J=1 TO 500:NEXT J:GO TO 41
32 I=I+1
33 IF I=1 THEN PRINT TITLE$:PRINT "GENERATING DATA, (X,Y)=";
34 PRINT "(";X;",";Y;") ";:REM You may wish to take this out, I like to see how
I am progressing
35 X(I)=X:Y(I)=Y:REM store the values
39 NEXT X
41 COUNT=I:REM Count is the total number of points to be plotted
60 GOSUB 5000:REM call PLOTTING routine
70 PRINT TITLE$;
80 INPUT Z$:GRAPHICS 0:PRINT :PRINT "DO YOUR GRAPH NOW":LIST 19,25:PRINT "RUN
 ":END
4999 REM . What follows should not change for any plots
5000 REM *** PLOTTING SUBROUTINE ***
5001 REM *** STEP4 *** Find the smallest and largest values of X and Y
5002 XSML=X(1):YSML=Y(1):XLRG=X(1):YLRG=Y(1):REM Set the maximum and minimum to
the first value,then compare
5010 FOR I=1 TO COUNT
5015 IF X(I)>XLRG THEN XLRG=X(I)
5020 IF X(I)<XSML THEN XSML=X(I)
5025 IF Y(I)<YSML THEN YSML=Y(I)
5030 IF Y(I)>YLRG THEN YLRG=Y(I)
5040 NEXT I
5050 COLOR 1:REM .You must have this for the graphs to show
5055 REM *** STEP 5 *** Calculate the scaling factor as discussed in the article
5060 YSCALE=(YLRG-YSML)/YGRAPHICS*1.01:IF YSCALE=0 THEN YSCALE=1
5065 XSCALE=(XLRG-XSML)/XGRAPHICS*1.01:IF XSCALE=0 THEN XSCALE=1
5068 IF ABS(LOG(XSCALE/YSCALE))>5 THEN PRINT " WARNING- THE RELATIVE RANGE OF X
AND Y VALUES IS VERY LARGE"
```

```
5070 IF 0<XSML OR 0>XLRG THEN GO TO 5080
5075 X=(0-XSML)/XSCALE:Y=0:GOSUB 5400:Y=YGRAPHICS:GOSUB 5300:REM YAXIS
5080 IF 0<YSML OR 0>YLRG THEN GO TO 5100
5085 Y=(0-YSML)/YSCALE:X=0:GOSUB 5400:X=XGRAPHICS:GOSUB 5300:REM X AXIS
5090 REM *** STEP 6 *** Scale each (X,Y) pair for plotting
5100 FOR I=1 TO COUNT:REM Now we are ready to plot
5110 X=(X(I)-XSML)/XSCALE
5120 Y=(Y(I)-YSML)/YSCALE
5130 GOSUB 5400
5140 NEXT I
5150 REM *** STEP 7 *** ADD AXES AND LABEL THEM.  TO PUT IN Y AXIS LABELS YOU MU
ST USE PEEK AND POKE
5153 PRINT :PRINT
5154 PRINT "|    |    |    |    |    |    |    |"
5158 LEGEND$="                                    "
5159 REM a point x on the text scale would be at z=x*(xgraphics+1)/40 in the gra
phics window.
5160 FOR X=2 TO 32 STEP 10:LABEL$=STR$(INT(XSML+(10*X*XSCALE*(XGRAPHICS+1)/40))/
10):LEGEND$(X-1)=LABEL$:NEXT X
5161 FOR X=2 TO 32 STEP 10:LABEL$=STR$(INT(10*(XSML+X*(XGRAPHICS+1)/40*XSCALE))/
10):LEGEND$(X-1)=LABEL$:NEXT X
5162 IF LEN(LEGEND$)>38 THEN PRINT LEGEND$(1,38)
5164 IF LEN(LEGEND$)<=38 THEN PRINT LEGEND$
5168 IF BAD>0 THEN PRINT BAD;" POINTS OUT OF RANGE ";
5200 REM . Check X and Y to see if in range
5210 IF X<0 THEN X=0:BAD=BAD+1
5220 IF X>XGRAPHICS THEN X=XGRAPHICS:BAD=BAD+1
5230 IF Y<0 THEN Y=0:BAD=BAD+1
5240 IF Y>YGRAPHICS THEN Y=YGRAPHICS:BAD=BAD+1
5245 RETURN
5300 GOSUB 5200:DRAWTO X,YGRAPHICS-Y:RETURN
5400 REM *** STEP 6 *** This statement does the actual plotting, note how the Y
value is plotted
5401 GOSUB 5200:PLOT X,YGRAPHICS-Y:RETURN
```

Mixing Atari Graphics Modes 0 And 8

Douglas Crockford
Irvine, CA

Graphic mode 0 is the ATARI text mode. It supports upper case, lower case, inverse video, and has a position function for placing text anywhere within a 40 by 24 display field. Graphic mode 8 is the ATARI high resolution plot mode. It supports the plotting of points and lines in a 320 by 160 (or 192) display field. It would be very nice to use both modes at the same time. The text window is some help, but it confines the plot to the top and the text to the bottom. Modifying the display list provides a partial solution, but it is awkward and doesn't permit the mixing of text and plot on the same line.

A better solution is to use graphic mode 8 and plot the dots that make up the text characters. This can be done very quickly by a 6502 subroutine, which does things in software which are very similar to what the display hardware does 60 times a second.

The subroutine is called with the USR function. It has four arguments:

- the horizontal cursor position
- the vertical cursor position
- the address of the string to be displayed
- the length of the string to be displayed.

So, the code

```
GR.0
POSITION X,Y
PRINT STRING$;
```

will produce similar results to

```
GR.8
A=USR(ADR(PRINT$),X,Y,ADR(STRING$),LEN
(STRING$))
```

PRINT$ is a string containing the subroutine. The STRING$ should not extend past the last column on a row. Any imbedded function codes (cursor movement, insert, etc.) will be displayed literally. The position of the PLOT/DRAWTO pointer is not changed, nor is the current COLOR.

An interesting bonus is that adding 40*R to the horizontal argument causes the text to be displayed R plot rows lower than usual. This permits the display of subscripts, mathematical expressions, 1½ line spacing, underlining, and so on.

The program was prepared with the ATARI Assembler Editor. Following is an explanation of the program.

Lines	Explanations
1110.	SAVMSC contains a pointer to the first byte of display memory. (Not the display list, but the display itself.)
1120.	CHBAS contains the high order byte of the address of the character generator. Normally, this points to the character generator in the OS ROM. However, if a user defined character generator is in effect, then it will be used. Note that by changing the character generator between calls, several character sets could be on the screen at the same time.
1130–1220.	The temporary variables are put in the page 0 space reserved for the floating point package.
1270.	Pull the number of arguments off the stack.
1280–1360.	If there aren't exactly 4 arguments, then pull all the arguments off and return. This section of code isn't completely necessary, but without it, accidentally forgetting an argument could kill the system. It's a small price to pay.
1400–1430.	Pull off the horizontal position and store it in DISP.
1440–1460.	Pull off the vertical position and put it in Y for now.
1470–1500.	Pull off the string address and store it in STR.
1510–1540.	Pull off the string length. Return if it is 0. Otherwise, store it in LEN.
1580–1640.	Add SAVMSC to DISP.
1650–1740.	Add (Y * 320) to DISP by adding 320, Y times. 320 = 40 * 8 = (1 * 256) + 64
1780.	Y contains 0, so store it in SCAN. There are 8 plot lines in a character, and we will use SCAN to remember which one we're on.
1820–1830.	CHAR will remind us of which character of STRING$ we're on.
1880–1930.	Get the next character and save it in X. INV gets set to $FF if the MSB of the character is set, and 0 otherwise. INV is used to do inverse video.
1940–2100.	ATASCII codes don't go directly into display memory. Bits 5 and 6 get shuffled around first. This is done so that lower case in GR.1 would display as upper case of another color, instead of as digits and special characters.
2150–2220.	GENP is set to point to the beginning of the 8 byte section of the character generator for this generator for this character. The multiplication by 8 is done by shifting left 3 times.
2350–2380.	Select the proper byte from the character generator. If INV is $FF, then flip its bits. Put the byte in the display.
2350–2380·	Increment CHAR. If there are more characters in the string, then repeat.
2420–2470.	Advance DISP to the next row.
2480–2520.	Increment SCAN. If there are more scans to do, then repeat. Otherwise, return. The number returned to BASIC is meaningless.

The subroutine is position independent because it contains no JPs, JSRs, or data references to itself. It can run anywhere in memory. It is also under 256 bytes, so it can also run in page 6. It can be put into your program using the techniques in the BASIC Reference Manual and the Assembler Editor User's Manual. It can also be loaded from disk by a BASIC program. The DEMO program shows the subroutine being loaded into a string called PRINT$, and shows a few of the things it can do.

```
1 REM DEMO
2 REM THIS PROGRAM SHOWS HOW TO LOAD
3 REM A MACHINE LANGUAGE SUBROUTINE
4 REM AND EXECUTE IT.
5 REM DOUGLAS CROCKFORD
10 OPEN #5,4,0,"D:PRINT.OBJ":GET #5,L:GET #5,H
20 GET #5,L:GET #5,H:BEGIN=H*256+L
30 GET #5,L:GET #5,H:LET END=H*256+L-BEGIN+1:DIM PRINT$(END)
40 FOR I=1 TO END:GET #5,L:LET PRINT$(I)=CHR$(L):NEXT I
60 CLOSE #5
90 GRAPHICS 8
1000 A=USR(ADR(PRINT$),10,10,ADR("[1+(3n-2)]TR = RRRRR"),20)
1010 A=USR(ADR(PRINT$),101,9,ADR("n"),1)
1020 A=USR(ADR(PRINT$),105,9,ADR("3n -n"),5)
1030 A=USR(ADR(PRINT$),267,8,ADR("2"),1)
1040 A=USR(ADR(PRINT$),221,10,ADR("2"),1)
1050 A=USR(ADR(PRINT$),227,10,ADR("2"),1)
2000 DIM TEXT$(40)
2010 TEXT$="Mixing ATARI graphics modes 0 and 8"
2020 A=USR(ADR(PRINT$),3,0,ADR(TEXT$),LEN(TEXT$))
2030 COLOR 1:PLOT 24,9:DRAWTO 304,9
2100 PLOT 64,60:DRAWTO 260,60:DRAWTO 260,100:DRAWTO 64,100:DRAWTO 64,60
2110 PLOT 64,100:DRAWTO 74,110:DRAWTO 270,110:DRAWTO 270,70:DRAWTO 260,60
2120 PLOT 260,100:DRAWTO 270,110
```

```
            1020 ; A=USR(1536,X,Y,ADR(STRING$),LEN(STRING$))
            1040 ; THIS PROGRAM GIVES A TEXT DISPLAY CAPABILITY IN BASIC
            1050 ; GRAPHIC MODE 8.  IT RECEIVES X AND Y CHARACTER
            1060 ; COORDINATES FOR THE GRAPHIC WINDOW, AND A STRING OF
            1070 ; 40 OR FEWER CHARACTERS.
            1090 ; DOUGLAS CROCKFORD   02/28/81
0058        1110 SAVMSC =    $58       ; POINTER TO DISPLAY MEMORY
02F4        1120 CHBAS  =    $02F4     ; PAGE POINTER TO THE CHARACTER GENERATOR
00D4        1130 LEN    =    $D4       ; THE LENGTH OF THE STRING TO DISPLAY
00D5        1140 INV    =    $D5       ; INVERSE VIDEO CHARACTER FLAG
00D6        1150 DISPL  =    $D6       ; DISPLAY TEXT POINTER
00D7        1160 DISPH  =    $D7
00D8        1170 STRL   =    $D8       ; SOURCE STRING POINTER
00D9        1180 STRH   =    $D9
00DA        1190 GENPL  =    $DA       ; CHARACTER GENERATOR POINTER
00DB        1200 GENPH  =    $DB
00DC        1210 CHAR   =    $DC       ; CURRENT CHARACTER INDEX
00DD        1220 SCAN   =    $DD       ; CURRENT SCAN LINE
0000        1230        *=   $600
            1250 ; VERIFY THAT THE NUMBER OF ARGUMENTS IS CORRECT.
0600 68     1270        PLA
0601 C904   1280        CMP  #4
0603 F009   1290        BEQ  FI3
0605 AA     1300        TAX
0606 F005   1310        BEQ  FI2
0608 68     1320 DO1    PLA
0609 68     1330        PLA
060A CA     1340        DEX
060B D0FB   1350        BNE  DO1
060D 60     1360 FI2    RTS
            1380 ; RECEIVE THE ARGUMENTS
060E 68     1400 FI3    PLA
060F 85D7   1410        STA  DISPH
0611 68     1420        PLA
0612 85D6   1430        STA  DISPL
0614 68     1440        PLA
0615 68     1450        PLA
0616 A8     1460        TAY
0617 68     1470        PLA
0618 85D9   1480        STA  STRH
061A 68     1490        PLA
061B 85D8   1500        STA  STRL
```

```
061D 68      1510           PLA
061E 68      1520           PLA
061F F0EC    1530           BEQ  FI2
0621 85D4    1540           STA  LEN
             1550 ;
             1560 ; SET THE DISPLAY POINTER
             1570 ;
0623 18      1580           CLC                ; DISP := SAVMSC + X + (Y * 320)
0624 A5D6    1590           LDA  DISPL
0626 6558    1600           ADC  SAVMSC
0628 85D6    1610           STA  DISPL
062A A559    1620           LDA  SAVMSC+1
062C 65D7    1630           ADC  DISPH
062E 85D7    1640           STA  DISPH
0630 98      1650           TYA
0631 F00F    1660           BEQ  FI5
0633 A5D6    1670 DO4       LDA  DISPL
0635 6940    1680           ADC  #64
0637 85D6    1690           STA  DISPL
0639 A5D7    1700           LDA  DISPH
063B 6901    1710           ADC  #1
063D 85D7    1720           STA  DISPH
063F 88      1730           DEY
0640 D0F1    1740           BNE  DO4
             1750 ;
             1760 ; LOOP FOR THE EIGHT PLOT LINES IN A CHARACTER
             1770 ;
0642 84DD    1780 FI5       STY  SCAN
             1790 ;
             1800 ; LOOP FOR EACH CHARACTER IN THE STRING
             1810 ;
0644 A000    1820 DO6       LDY  #0
0646 84DC    1830           STY  CHAR
             1840 ;
             1850 ; FETCH A CHARACTER.  SET  INV  IF THE MSB IS SET.  CONVERT
             1860 ; IT FROM ATASCII TO THE DISPLAY CHARACTER SET.
             1870 ;
0648 B1D8    1880 DO7       LDA  (STRL),Y
064A A000    1890           LDY  #0
064C AA      1900           TAX
064D 1001    1910           BPL  FI8
064F 88      1920           DEY
0650 84D5    1930 FI8       STY  INV
0652 8A      1940           TXA
0653 2960    1950           AND  #96
0655 D004    1960           BNE  CA9
0657 A940    1970           LDA  #64
0659 100E    1980           BPL  FIB
065B C920    1990 CA9       CMP  #32
065D D004    2000           BNE  CAA
065F A900    2010           LDA  #0
0661 1006    2020           BPL  FIB
0663 C940    2030 CAA       CMP  #64
0665 D002    2040           BNE  FIB
0667 A920    2050           LDA  #32
0669 85DA    2060 FIB       STA  GENPL
066B 8A      2070           TXA
066C 291F    2080           AND  #$1F
066E 05DA    2090           ORA  GENPL
0670 85DA    2100           STA  GENPL
             2110 ;
             2120 ; MAKE A POINTER TO A CHARACTER IN THE CHARACTER
             2130 ; GENERATOR.
             2140 ;
0672 A900    2150           LDA  #0            ; GENP := (TRANSLATED CHARACTER * 8)
0674 A203    2160           LDX  #3            ;    + (CHBAS * 256)
0676 06DA    2170 DOC       ASL  GENPL
0678 2A      2180           ROL  A
```

```
0679 CA              DEX
067A D0FA            BNE   D0C
067C 6DF402          ADC   CHBAS
067F 85DB            STA   GENPH
                ;
                ; FETCH A CHARACTER SEGMENT FROM THE CHARACTER GENERATOR.
                ; COMPLEMENT IT IF INV IS SET.  STORE IT IN DISPLAY MEMORY.
                ;
0681 A4DD            LDY   SCAN
0683 B1DA            LDA   (GENPL),Y
0685 45D5            EOR   INV
0687 A4DC            LDY   CHAR
0689 91D6            STA   (DISPL),Y
                ;
                ; TEST FOR END OF STRING
                ;
068B C8              INY
068C 84DC            STY   CHAR
068E C4D4            CPY   LEN
0690 D0B6            BNE   D07
                ;
                ; ADVANCE THE DISPLAY POINTER.  TEST FOR LAST SCAN.
                ;
0692 18              CLC
0693 A5D6            LDA   DISPL
0695 6928            ADC   #40
0697 85D6            STA   DISPL
0699 9002            BCC   FID
069B E6D7            INC   DISPH
069D E6DD       FID  INC   SCAN
069F A908            LDA   #8
06A1 C5DD            CMP   SCAN
06A3 D09F            BNE   D06
06A5 60              RTS
06A6                 .END
```

©

An Atari Disassembler And Memory Lister

Charles Fortner
Lawrenceville, GA

Thomas Gordon's "A 6502 Disassembler" was published in the January, 1981, issue of **COMPUTE!**. The following program is a conversion for use on an Atari, and also includes an option to sequentially list the contents of memory.

The major obstacle in converting this program is caused by the lack of string arrays in Atari Basic. However, the conversion is not as difficult as might be suspected due to Atari's ability to dimension a string variable to any length. R$ is dimensioned to handle the 255 different opcodes of four digits each in line 5. R$ is then cleared in line 12, and the opcodes inserted by the subroutine at line 250. String manipulations are then used to check for the different opcodes.

The user may press "SELECT" during disassembly/listing to choose a new starting address or press "START" to choose a different mode of operation.

The disassembler as listed will request a new starting address if an invalid opcode is encountered. One possible program change is to convert line 75 to increment the current memory address (S) by one, and then branch to line 40 instead of 20 when an invalid opcode is encountered. This will allow a continuous listing to occur which can help in locating the next segment of legitimate code.

Many mysteries of the Atari can be uncovered by using the disassembler on the operating system and the basic cartridge. How many can you find?

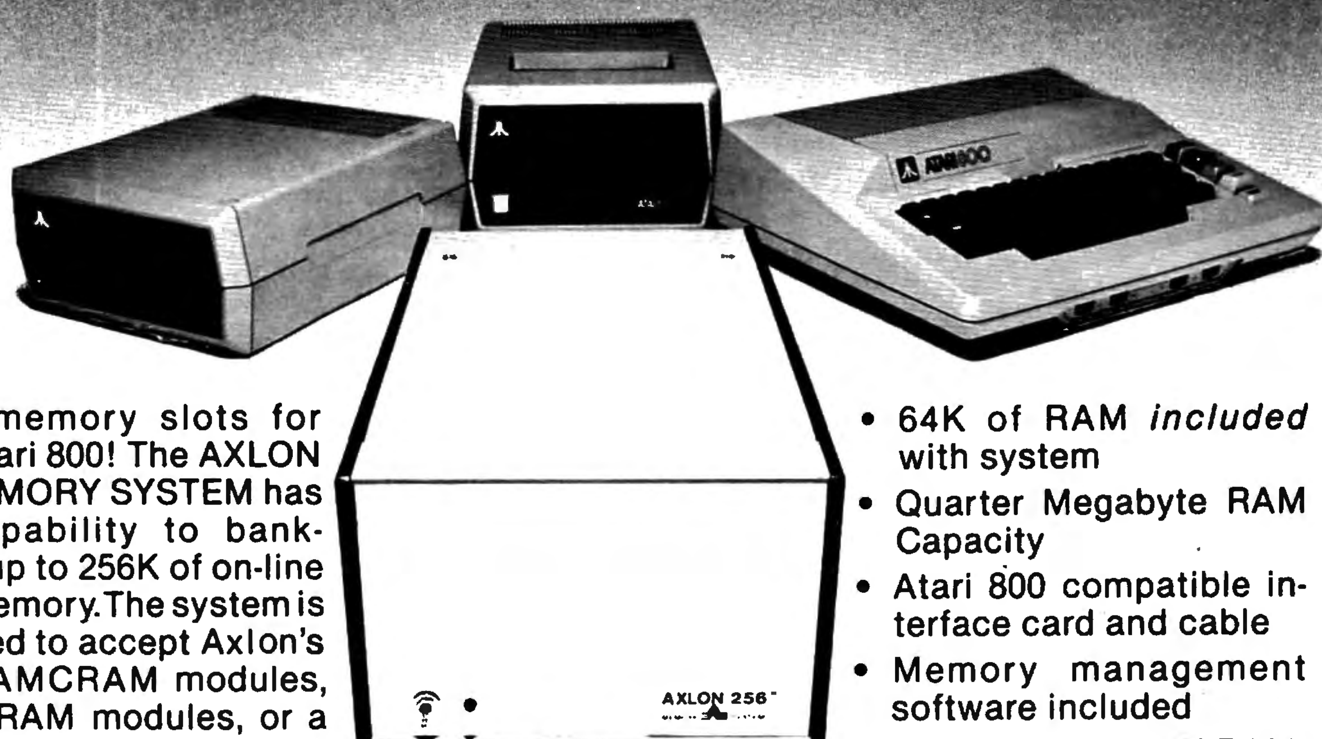
```
1 REM DISASSEMBLER ORIGINALLY BY THOMAS G. GORDAN-COMPUTE MAGAZINE 1/81
2 REM CONVERTED TO ATARI BY CHARLES G.FORTNER 1/81
5 DIM R$(1032),M$(4),A$(4),B$(1),C$(1),D$(1),E$(1),F$(4),FR$(1),TH$(1),TW$(1),OE
$(1),U$(1),ANS$(3),DIS$(3)
10 ? "6502 DISASSEMBLER AND MEMORY LISTER ":? :?
11 ? "PLEASE WAIT WHILE I SET UP"
12 FOR X=1 TO 1032:R$(X,X)=" ":NEXT X
15 GOSUB 250:REM FILL IN ALL OPCODES
16 ? :? "DO YOU WANT DISASSEMBLER":INPUT DIS$
20 PRINT "ENTER STARTING ADDRESS IN 4 DIGIT HEX":? :? "ADDRESS";
21 INPUT A$:IF DIS$="NO" THEN 1505:REM GOTO MEMORY LISTER ROUTINE
35 GOSUB 900:REM CONVERT HEX TO DECIMAL
40 Z=PEEK(S):A=S:IF PEEK(53279)=5 THEN ? :GOTO 20:REM IF 'SELECT' PRESSED THEN G
ET NEW ADDRESS FOR DISASSEMBLER
41 IF PEEK(53279)=6 THEN 16:REM IF 'START' PRESSED THEN GIVE CHOICE OF DISASSEMB
LER OR MEMORY LISTER
55 GOSUB 1000:REM GET HEX ADDRESS
60 PRINT ,;FR$;TH$;TW$;OE$;" ";
70 A=Z:GOSUB 1000
75 ? ;TW$;OE$;" ";:IF R$(Z*4+1,Z*4+1)=" " THEN ? "IS AN INVALID OPCODE":GOTO 20
76 ? R$(Z*4+1,Z*4+3);" ";
80 U$=R$(Z*4+4)
90 IF U$=" " THEN PRINT ;U$:S=S+1:GOTO 40
95 IF U$="N" THEN PRINT ;"A@";:GOTO 600
100 IF U$="A" THEN PRINT ;U$:S=S+1:GOTO 40
105 IF U$="Z" THEN PRINT ;"A@";:GOTO 625
110 IF U$="@" THEN PRINT ;"@$";:GOTO 625
115 IF U$="X" THEN PRINT ;"A@";:GOTO 645
120 IF U$="Y" THEN PRINT ;"A@";:GOTO 665
125 IF U$="B" THEN PRINT ;" (";:GOTO 685
130 IF U$="C" THEN PRINT ;" (";:GOTO 700
135 IF U$="U" THEN PRINT ;"A@";:GOTO 715
140 IF U$="R" THEN PRINT ;"TO ";:GOTO 765
145 IF U$="J" THEN PRINT ;" (";:GOTO 735
150 IF U$="V" THEN PRINT ;"A@";:GOTO 755
250 REM SUBROUTINE TO FILL IN ALL OPCODES
251 FOR X=0 TO 255:READ M$:R$((X*4)+1,(X*4)+4)=M$:NEXT X
255 DATA BRK ,ORAB,,,,ORAZ,ASLZ,,PHP ,ORA@,ASLA,,,ORAN,ASLN,,BPLR,ORAC,,,,ORAU,A
SLU,,CLC ,ORAY,,,,ORAX
260 DATA ASLX,,JSRN,ANDB,,,BITZ,ANDZ,ROLZ,,PLP ,AND@,ROLA,,BITN,ANDN,ROLN,,BMIR,
ANDC,,,,ANDU,ROLU,,SEC ,ANDY,,,
```

ATARI* 800* OWNERS!

NOW expand to

256K

RAM MEMORY CAPACITY!



Eight memory slots for your Atari 800! The AXLON 256 MEMORY SYSTEM has the capability to bank-select up to 256K of on-line RAM memory. The system is designed to accept Axlon's 32K RAMCRAM modules, Atari's RAM modules, or a combination of the two. Our memory system comes complete with hardware, 64K of RAM, memory management software, interface card and cable. Drop by your local computer store or contact Axlon for more information.

AXLON 256™
MEMORY SYSTEM

- 64K of RAM *included* with system
- Quarter Megabyte RAM Capacity
- Atari 800 compatible interface card and cable
- Memory management software included
- Accepts Axlon 32K RAMCRAM modules, Atari RAM modules, or a combination of both
- Dedicated power supply
- 90 Day Warranty

170 Wolfe Road
Sunnyvale, CA 94086
(408) 730-0216

AXLON

*Indicates trademarks of Atari, Inc.

```
265 DATA ANDX,ROLX,,RTI ,EORB,,,,EORZ,LSRZ,,PHA ,EOR@,LSRA,,JMPN,EORN,LSRN,,BVCR
,EORC,,,,EORU,LSRU,,CLI ,EORY,,,
270 DATA EORX,LSRX,,RTS ,ADCB,,,,ADCZ,RORZ,,PLA ,ADC@,RORA,,JMPJ,ADCN,RORN,,BVSR
,ADCC,,,,ADCU,RORU,,SEI ,ADCY,,,
275 DATA ADCX,,,,STAB,,,STYZ,STAZ,STXZ,,DEY ,,TXA ,,STYN,STAN,STXN,,BCCR,STAC,,,
STYU,STAU,STXV,,TYA ,STAY,TXS ,,
280 DATA STAX,,,LDY@,LDAB,LDX@,,LDYZ,LDAZ,LDXZ,,TAY ,LDA@,TAX ,,LDYN,LDAN,LDXN,,
BCSR,LDAC,,,LDYU,LDAU,LDXV,
285 DATA CLV ,LDAY,TSX ,,LDYX,LDAX,LDXY,,CPY@,CMPB,,,CPYZ,CMPZ,DECZ,,INY ,CMP@,D
EX ,,CPYN,CMPN,DECN,,BNER
290 DATA CMPC,,,,CMPU,DECU,,CLD ,CMPY,,,,CMPX,DECX,,CPX@,SBCB,,,CPXZ,SBCZ,INCZ,,
INX ,SBC@,NOP ,,CPXN,SBCN,INCN,
300 DATA BEQR,SBCC,,,,SBCU,INCU,,SED ,SBCY,,,,SBCX,INCX,
310 RETURN
600 A=PEEK(S+2):GOSUB 1000
605 PRINT ;TW$;DE$;
610 A=PEEK(S+1):GOSUB 1000
615 PRINT ;TW$;DE$
620 S=S+3:GOTO 40
625 A=PEEK(S+1):GOSUB 1000
630 PRINT ;TW$;DE$
632 S=S+2:GOTO 40
645 A=PEEK(S+2):GOSUB 1000
650 PRINT ;TW$;DE$;
655 A=PEEK(S+1):GOSUB 1000
660 PRINT ;TW$;DE$;",X":S=S+3:GOTO 40
665 A=PEEK(S+2):GOSUB 1000
670 PRINT ;TW$;DE$;
675 A=PEEK(S+1):GOSUB 1000
680 PRINT ;TW$;DE$;",Y":S=S+3:GOTO 40
685 A=PEEK(S+1):GOSUB 1000
690 PRINT ;TW$;DE$;",X)":S=S+3:GOTO 632
700 A=PEEK(S+1):GOSUB 1000
705 PRINT ;TW$;DE$;"),"
715 A=PEEK(S+1):GOSUB 1000
720 PRINT ;TW$;DE$;",X":GOTO 632
735 A=PEEK(S+2):GOSUB 1000
740 PRINT ;TW$;DE$;
745 A=PEEK(S+1):GOSUB 1000
750 PRINT ;TW$;DE$;")":S=S+3:GOTO 40
755 A=PEEK(S+1):GOSUB 1000
760 PRINT ;TW$;DE$;"Y":GOTO 632
765 A=PEEK(S+1):IF A<128 THEN 790
770 A=255-A
775 A=S+1-A:GOSUB 1000
780 PRINT ;FR$;TH$;TW$;DE$:GOTO 632
790 A=S+A+2:GOSUB 1000
795 GOTO 780
900 REM SUBROUTINE TO CONVERT HEX TO
    DECIMAL-S=DECIMAL VALUE
901 B$=A$(1):C$=A$(2,2):D$=A$(3,3):
    E$=A$(4,4):F$=B$
925 FOR X=1 TO 4
930 IF F$="A" THEN A=10:GOTO 965
935 IF F$="B" THEN A=11:GOTO 965
940 IF F$="C" THEN A=12:GOTO 965
945 IF F$="D" THEN A=13:GOTO 965
950 IF F$="E" THEN A=14:GOTO 965
955 IF F$="F" THEN A=15:GOTO 965
960 A=VAL(F$)
965 IF X=1 THEN S=A*4096:F$=C$
970 IF X=2 THEN S=S+A*256:F$=D$
975 IF X=3 THEN S=S+A*16:F$=E$
980 IF X=4 THEN S=S+A
985 NEXT X
990 RETURN
```

```
1000 REM SUBROUTINE TO CONVERT DECIMAL TO HEX
1001 F=INT(A/4096)
1005 R=A-F*4096
1010 TH=INT(R/256)
1015 R=R-TH*256
1020 TW=INT(R/16)
1025 OE=R-TW*16:H=F
1030 FOR X=1 TO 4
1035 IF H=10 THEN F$="A":GOTO 1070
1040 IF H=11 THEN F$="B":GOTO 1070
1046 IF H=12 THEN F$="C":GOTO 1070
1050 IF H=13 THEN F$="D":GOTO 1070
1055 IF H=14 THEN F$="E":GOTO 1070
1060 IF H=15 THEN F$="F":GOTO 1070
1065 F$=STR$(H)
1070 IF X=1 THEN FR$=F$:H=TH
1075 IF X=2 THEN TH$=F$:H=TW
1080 IF X=3 THEN TW$=F$:H=OE
1085 IF X=4 THEN OE$=F$
1090 NEXT X
1095 RETURN
1500 REM START MEMORY LISTER ROUTINE
1501 ? "ENTER STARTING ADDRESS IN 4 DIGIT HEX":? :? "ADDRESS";
1502 INPUT A$
1505 GOSUB 900
1510 Z=PEEK(S):A=S:IF PEEK(53279)=6 THEN ? :GOTO 16:REM IF 'START' PRESSED THEN
GIVE CHOICES AGAIN
1520 GOSUB 1000:REM GET HEX ADDRESS
1530 ? ,;FR$;TH$;TW$;OE$;" ";S;" ";
1540 A=Z:GOSUB 1000
1550 ? ;TW$;OE$
1560 S=S+1:IF PEEK(53279)=5 THEN 1585:REM IF 'SELECT' PRESSED THEN GET NEW ADDRE
SS FOR MEMORY LISTER
1570 GOTO 1510
1585 ? :GOTO 1500
```

©

Computerized Greeting Cards

For The Atari Computer

John Victor
Greenwich, CT

The idea of computerized greeting cards is not new — versions have been printed in other computer publications for the TRS-80 and the Apple computers. However, the Atari computer allows the programmer to go beyond simply designing and printing graphics Christmas trees. Instead, the programmer can produce really spectacular personalized computer shows that can be sent to friends on cassette (if the friends are fortunate enough to own an Atari computer). The cassette can include graphics, sound effects, music and a spoken message.

The program that I designed for this article was sent to my nephews Christopher and Eric who were getting an Atari 400 for Christmas. I realize that Christmas is now over, but the reader may want to use this one as a model for other holiday greetings.

Line 10 in the program indicates the locations of the subroutines used in this program. TREE starting at line 1000 draws a Christmas tree using regular keyboard characters (colored green). Every other * is actually printed in reverse type, but my printer will not print in reverse.

When the Christmas tree is printed on the screen, it appears all green. However, when the FLASH subroutine at line 1080 is activated, the reverse characters suddenly change color and flash on and off.

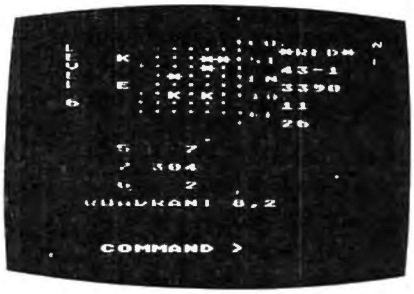
There are two music subroutines in this program, each of which plays a different type of music. The first (called MUSIC at line 1500) uses one voice and plays a short Christmas theme. The sound is "shaped" using the variable called LOUD. When the sound register is first turned on, LOUD is set to 15. The routine then rapidly reduces the value in LOUD, which creates a "plunking" sound like a piano. Each note is "plunked" as it is played. MUSIC2 plays "Joy To The World" in three part harmony.

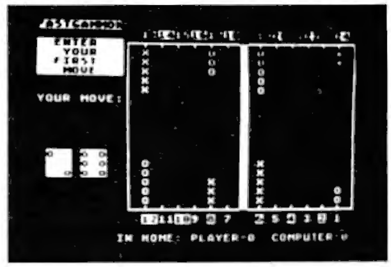
At three points in the program the cassette player is turned on with a POKE 54018,52 instruction. The program then goes to KEYPRESS which times the cassette player. This allows a spoken message to be delivered from cassette for a given

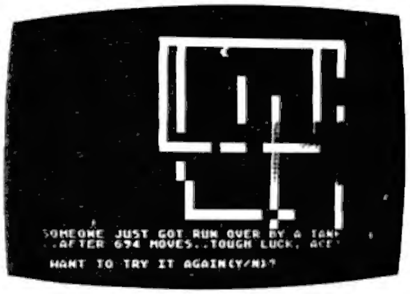
```
5 REM COMPUTERIZED CHRISTMAS CARD
6 REM JOHN VICTOR, PROGRAM DESIGN, INC.
10 MUSIC=1500:TREE=1000:DIM NAME$(20):KE
YPRESS=1600:FLASH=1080:MUSIC2=1550
100 GRAPHICS 2+16:POSITION 0,1:PRINT #6;
"____________________":POSITION 3,2:SETC
OLOR 4,1,0:PRINT #6;"MERRY"
110 POSITION 5,4:PRINT #6;"CHRISTMAS":PR
INT #6;"____________________"
115 FOR X=1 TO 20:SOUND 0,INT(RND(1)*255
),10,8:FOR Y=1 TO 50:NEXT Y:NEXT X
116 FOR Y=1 TO 50:NEXT Y
120 NAME$="christopher AND eric":FOR X=0
 TO 19:POSITION X,7:PRINT #6;"*":FOR Y=1
 TO 10:NEXT Y
121 SOUND 0,13*X,8,8
125 POSITION X,7:PRINT #6;NAME$(X+1,X+1)

126 FOR Y=1 TO 10:NEXT Y:NEXT X:SOUND 0,
0,0,0:FOR X=1 TO 600:NEXT X:GOSUB MUSIC
150 POKE 54018,52:TALKTIME=10:GOSUB KEYP
RESS
200 GOSUB TREE:POKE 54018,52:TALKTIME=5:
GOSUB KEYPRESS:GOSUB FLASH:GOSUB MUSIC
220 NAME$="christopher and eric":FOR X=0
 TO 19:POSITION X,10:PRINT #6;"*":FOR Y=
1 TO 10:NEXT Y
221 SOUND 0,13*X,8,8
225 POSITION X,10:PRINT #6;NAME$(X+1,X+1
)
226 FOR Y=1 TO 10:NEXT Y:NEXT X:SOUND 0,
0,0,0:POKE 54018,52:TALKTIME=8:GOSUB KEY
PRESS
230 GOSUB MUSIC2:END
1000 GRAPHICS 2+16
1001 SOUND 0,PT,10,LOUD
1010 PRINT #6:PRINT #6:PRINT #6
1020 PRINT #6;"         *"
1030 PRINT #6;"        ***"
1040 PRINT #6;"       *****"
1050 PRINT #6;"      *******"
1060 PRINT #6;"     *********"
1070 PRINT #6;"         i"
1071 PRINT #6;"MMMMMMMMMMMMMMMMMMMM"
1075 SETCOLOR 0,12,6:SETCOLOR 2,12,6:RET
URN
1080 FOR COUNT=1 TO 10:SETCOLOR 2,4,15:F
OR DELAY=1 TO 50:NEXT DELAY:SETCOLOR 2,4
,4:FOR DELAY=1 TO 50:NEXT DELAY
1090 NEXT COUNT:POSITION 9,2:PRINT #6;"+
":SETCOLOR 2,1,15:SETCOLOR 4,8,0
1100 RETURN
1500 READ PITCH,TIME
1510 POKE 20,0:IF PITCH=0 AND TIME=0 THE
N RESTORE :RETURN
1520 LOUD=15
1521 SOUND 0,PITCH,10,LOUD:LOUD=LOUD-LOU
D/5:C=PEEK(20):IF C<TIME THEN GOTO 1521
```

SOFTWARE FOR THE ATARI 800* AND THE ATARI 400*







TARI TREK™
By Fabio Ehrengruber

Get ready for an exciting trek through space. Your mission is to rid the galaxy of Klingon warships, and to accomplish this you must use strategy to guide the starship Enterprise around stars, through space storms, and amidst enemy fire. Sound and color enliven this action-packed version of the traditional trek game. Nine levels of play allow the player to make the mission as easy or as challenging as he wishes. At the highest level you are also playing against time. Damage to your ship can be repaired in space at a cost of time and resources if you can't make it back to base. TARI TREK gives you a lot of trek at a low price. This program is written entirely in BASIC and requires at least 24K of user memory. For the Atari 800 only.

Cassette - $11.95 Diskette - $14.95

FASTGAMMON™
By Bob Christiansen

Play backgammon against a talented computer opponent This is the latest and best version of the most popular backgammon-playing program for personal computers - FASTGAMMON. Roll your own dice or let the computer roll them for you. Adjust the display speed to be fast or slow. If you wish you can play a game using the same dice rolls as the previous game - a great aid in improving your skills at backgammon. Beginners find it easy to learn backgammon by playing against the computer, and even very good players find it a challenge to beat FASTGAMMON. The 12-page instruction booklet includes the rules of the game. Written in machine language. Requires only 8K of RAM and runs on both the Atari 400 and the Atari 800.

On cassette only - $19.95

TANK TRAP
By Don Ursem

A rampaging tank tries to run you down. You are a combat engineer, building concrete barriers in an effort to contain the tank. Use either the keyboard or an Atari joystick to move your man and build walls. If you trap the tank you will be awarded a rank based on the amount of time and concrete you used up. But they'll be playing taps for you if you get run over. There are four levels of play. Higher levels of play introduce slow curing concrete, citizens to protect, and the ability of the tank to shoot through any wall unless you stay close by. Music, color, and sound effects add to the excitement. Written in BASIC with machine language subroutines. Requires at least 16K of user memory Runs on the Atari 800 and on an Atari 400 with 16K RAM.

Cassette - $11.95 Diskette - $14.95

QS FORTH™ By James Albanese. Step into the world of the remarkable FORTH programming language. Writing programs in FORTH is much easier than writing them in assembly language, yet FORTH programs run almost as fast as machine code and many times faster than BASIC programs. QS FORTH is based on fig-FORTH, the popular model from the FORTH Interest Group that has become a standard for microcomputers. QS FORTH is a disk-based system that can be used with up to four disk drives. There are five modules included:

1. The FORTH KERNEL (The standard fig-FORTH model customized to run on the Atari computer).
2. An EXTENSION to the basic vocabulary that contains some handy additional words.
3. An EDITOR that allows editing source programs (screens) using Atari type editing.
4. An IOCB module that makes I/O operations easy to set up.
5. An ASSEMBLER that allows defining FORTH words as a series of 6502 assembly language instructions.

Modules 2-5 may not have to be loaded with the user's application program, allowing for some efficiencies in program overhead. Full error statements (not just numerical codes) are printed out, including most disk error statements. QS FORTH requires at least 24K of RAM and at least one disk drive. For the Atari 800 only.

On diskette only - $79.95

★ ★ ★ ★ ★ ★ ★ ★ ★ ★

ASSEMBLER by Gary Shannon. Write your own 6502 machine language programs with this inexpensive in-RAM editor/assembler. Use the editor to create and edit your assembler source code. Then use the assembler to translate the source code into machine language instructions and store the code in memory. Simple commands allow you to save and load the source code to and from cassette tape. You can also save any part of memory on tape and load it back into RAM at the same or at a different location. The assembler handles all 6502 mnemonics plus 12 pseudo-ops that include video and printer control. Commenting is allowed and error checking is performed. A very useful feature allows you to view and modify hexadecimal code anywhere in memory. Instructions on how to interface machine language subroutines to your BASIC programs are included. ASSEMBLER requires 16K of user memory and runs on both the Atari 800 and the Atari 400.

On cassette only - $24.95

★ ★ ★ ★ ★ ★ ★ ★ ★ ★

6502 DISASSEMBLER by Bob Pierce. This neat 8K BASIC program allows you to disassemble machine code, translating it and listing it in assembly language format on the video and on the printer if you have one. 6502 DISASSEMBLER can be used to disassemble the operating system ROM, the BASIC cartridge, and machine language programs located anywhere in RAM except where the DISASSEMBLER itself resides. (Most Atari cartridges are protected and cannot be disassembled using this disassembler.) Also works as an ASCII interpreter, translating machine code into ASCII characters. 6502 DISASSEMBLER requires only 8K of user memory and runs on both the Atari 800 and the Atari 400.

Cassette - $11.95 Diskette - $14.95

QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105, Reseda, CA 91335
(213) 344-6599

WHERE TO GET IT: Call us at (213) 344-6599 for the name of the Quality Software dealer nearest you If necessary you may order directly from us Mastercard and Visa cardholders may place orders by telephone Or mail your check or bankcard number to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335 California residents add 6% sales tax SHIPPING CHARGES Within North America orders must include $1 50 for first class shipping and handling. Outside North America the charge for airmail shipping and handling is $5.00 Pay in U.S. currency

*Indicates trademarks of Atari, Inc

```
1522 GOTO 1500
1550 RESTORE 3000
1551 READ PT1,PT2,PT3,TIME:IF TIME=0 THE
N RETURN
1552 SOUND 1,PT1,10,8:SOUND 2,PT2,10,8:S
OUND 3,PT3,10,8
1553 POKE 20,0
1554 C=PEEK(20):IF C<TIME THEN 1554
1555 FOR X=1 TO 3:SOUND X,0,0,0:NEXT X:G
OTO 1551
1600 POKE 19,0:POKE 20,0
1601 IF (PEEK(19)*256+PEEK(20))/60<TALKT
IME THEN 1601
1606 POKE 54018,60
1610 IF PEEK(764)<>255 THEN POKE 764,255
:RETURN
1611 GOTO 1610
2000 DATA 121,10,121,10,121,20,91,10,91,
10,91,20,96,10,91,10,81,10,72,10,68,10,8
1,10,72,30
2010 DATA 68,10,60,20,53,10,68,10,72,10,
91,10,81,20,91,60,0,0
3000 DATA 53,108,172,60,57,108,144,45,64
,108,162,15,72,108,144,90,81,96,128,30
3010 DATA 85,108,144,60,96,114,144,60,10
8,172,217,90,72,108,172,30,64,81,108,90,
64,81,162,30
3020 DATA 57,96,144,90,57,81,144,30,53,8
5,144,210,53,85,144,30,53,85,144,30,57,7
2,0,30
3030 DATA 64,81,108,30,72,85,0,30,72,85,
108,45,81,96,0,15,85,108,144,30,53,85,14
4,30
3040 DATA 53,85,144,30,57,72,0,30,64,81,
108,30,72,85,0,30,72,85,108,45,81,96,0,1
5,85,108,144,30
3050 DATA 85,108,217,30,85,108,144,30,85
,108,144,30,85,108,128,30,85,108,128,15,.
81,128,0,15,72,108,144,90
3060 DATA 81,108,128,15,85,0,0,15,96,114
,144,30,96,114,144,30,96,114,162,30,96,1
14,162,15,85,172,0,15
3070 DATA 81,114,193,90,85,114,144,15,96
,162,0,15,108,172,217,30,53,85,144,60,64
,108,162,30,72,108,144,45
3080 DATA 81,114,193,15,85,108,217,30,81
,96,128,30,85,108,144,60,96,114,144,60,1
08,172,217,120,0,0,0,0
```

interval. The programmer times this interval by setting the TALKTIME variable to a number given in seconds. When the time is up, the cassette player is shut off, and the program user must press a key to go on. (The prerecorded message must instruct the user to press the key.) LET TALKTIME = 10 will set the subroutine at 1600 to run for 10 seconds before it shuts off, and the greeting card recipient will have to press a key to keep the message coming.

The program can be recorded on the beginning of the cassette with the voice part recorded after the program. The voice will have to be timed carefully with a stopwatch so that it fits within the time allowed when the program is run. ©

Color Burst For Atari

Robert Blacka
Wyndmoor, PA

If you are looking for a way to display the Atari's excellent color graphics capabilities, here is a simple program that is guaranteed to amuse those friends which don't appreciate the finer points of calculating compound interest or the circumference of bicycle wheels.

The program starts out by selecting at random a point on the screen and a color after which the color appears to burst out in all directions. The program then selects a second point and color and the process repeats itself. Eventually, the colors begin to intermix creating complex intricate patterns. Just when the screen appears to be saturated with color, the display goes blank and the process starts all over.

Because the colors and points are selected completely at random, every design is unique and I have yet to see the same design repeated twice.

The Program works like this:

Line 110 sets the setcolor register to 0.
Line 112 limits the number of color bursts to 10.
Line 115 selects the color register corresponding to the setcolor statement.
Line 120, 130, 140 select the random start points and color.
Line 200 sets the number of color rays to be generated.
Line 210-220 select the end points of each color ray.
Line 250 delays the next color burst and makes the display more effective.
Line 260 selects a new color register.
Line 270 initiates another color burst.
Line 280 clears the screen and starts the process over again.

Some interesting variations can be created by changing Line 112 so that more or less color bursts are generated. Likewise, try experimenting with Line 200. As many as 100 color rays have been tried; however, these tend to saturate the screen rather quickly.

```
100 GRAPHICS 7+16
110 N=0
112 FOR R=1 TO 10
115 M=N+1
120 COLR=INT(16*RND(0))
130 X1=INT(159*RND(0))
140 Y1=INT(95*RND(0))
150 SETCOLOR N,COLR,10
160 COLOR M
200 FOR I=1 TO 12
210 X2=INT(159*RND(0))
220 Y2=INT(95*RND(0))
230 PLOT X1,Y1:DRAWTO X2,Y2
240 NEXT I
250 FOR Z=1 TO 200:
    NEXT Z
260 N=N+1:
    IF N>2 THEN N=0
270 NEXT R
280 GOTO 100
```

©

Binary/ Decimal Conversions For Atari

Jerry White
Levittown, NY

Sooner or later, just about every computer programmer will be faced with the task of converting a number from decimal to binary or from binary to decimal. I recently found myself in this situation. I got a scrap of paper and started calculating when I realized that my Atari could do it much faster than I can. It didn't take long to write the program since the logic is fairly simple and I've written dozens of programs in Atari Basic. Since a program like this may not be so easy for a less experienced programmer or a beginner, I dressed up the display and sent it to **COMPUTE!**

I used some special characters that will not print on the listing of the program. When you key in the program, you can leave out the REM statements but be sure to read them. For example, read lines 97 thru 99. After the first set of quotes in line 100, there should be 8 spaces in inverse video. Then we need two normal spaces and two up arrows. When you type that line, after the first set of quotes, press the Atari key, the space bar 8 times, the Atari key, the space bar twice, the ESC key, then hold the CTRL key and press the UP ARROW key, repeat the ESC-CTRL-UP ARROW procedure once more, then type the closing quotes.

When the user of this program is going to type a binary number, we want to remind the person that the program expects 8 digits. As the binary number is typed, it will replace the line of 8 inverse video spaces or cursors. This is a good way to display what is expected at an input instruction. There is also a drawback to this method. Suppose you typed in 11 then hit return. The input, in this case B$, would be two ones and six ATASCI character 160's. You may have to check for CHR$(160) in the string before accepting it. In this program, it doesn't matter. B$ is our binary number. We will be checking this string one position at a time to see if that position is a zero. If not, we will assume it is a one. We will also check the first position for the letters E and R. In any case, if we don't find what we are looking for, we will assume that we found the number one.

The rest of the program is straight forward. The variables used are B$ for binary number, D$ for decimal number in string form, D for decimal number in numeric form, D2 is that value divided by two, and GC is the character typed in the GET character routine (line 34).

When you have to convert from OR to binary numbers, run this program and let your computer do the work.

```
0 REM BINARY TO DECIMAL AND DECIMAL TO BINARY CONVERSION PROGRAM BY JERRY WHITE.

20 DIM D$(3),B$(8):B$="00000000"
30 GRAPHICS 0:SETCOLOR 2,0,0:POKE 752,1:POSITION 5,5
32 ? "TYPE B TO CONVERT FROM BINARY":? :? "   TYPE D TO CONVERT FROM DECIMAL"
34 OPEN #1,4,0,"K:":GET #1,GC:CLOSE #1
36 IF GC=66 THEN 50
38 IF GC=68 THEN 500
40 GOTO 34
49 REM LINE 50 BEGINS WITH A PRINT ESC-CTRL-CLEAR (CLEAR SCREEN) BETWEEN QUOTES
50 ? "}":SETCOLOR 2,0,0:DV=0:POKE 752,1:? :? "BINARY TO DECIMAL CONVERSION PROGR
AM:":GOTO 355
97 REM LINE 100 HAS 8 INVERSE VIDEO SPACES FOLLOWED BY
98 REM TWO NORMAL VIDEO SPACES FOLLOWED BY TWO UP ARROWS
99 REM WITHIN THE QUOTES
100 ? :? ,"          "
110 ? :? ,;:INPUT B$:IF B$(1,1)="E" THEN GRAPHICS 0:END
112 IF B$(1,1)="R" THEN RUN
120 DV=0:TRAP 360
200 IF B$(1,1)="0" THEN 220
210 DV=DV+128
220 IF B$(2,2)="0" THEN 240
230 DV=DV+64
240 IF B$(3,3)="0" THEN 260
250 DV=DV+32
260 IF B$(4,4)="0" THEN 280
270 DV=DV+16
```

```
280 IF B$(5,5)="0" THEN 300
290 DV=DV+8
300 IF B$(6,6)="0" THEN 320
310 DV=DV+4
320 IF B$(7,7)="0" THEN 340
330 DV=DV+2
340 IF B$(8,8)="0" THEN 355
350 DV=DV+1
355 POSITION 2,3:? :? ,"BINARY ";B$;"="
356 POSITION 26,6:? "   ":POSITION 2,5
360 ? :? ,"DECIMAL VALUE=";DV
370 ? :? ,"TYPE E TO END":? ,"TYPE R TO RERUN":? ,"OR TYPE A BINARY NUMBER."
400 TRAP 40000:GOTO 100
499 REM LINE 500 BEGINS WITH A PRINT ESC-CTRL-CLEAR (CLEAR SCREEN)
500 ? "}":SETCOLOR 2,0,0:POKE 752,1:? :? "DECIMAL TO BINARY CONVERSION PROGRAM:"
:GOTO 800
509 REM LINE 510 HAS 5 SPACES THEN TWO UP ARROWS WITHIN THE QUOTES
510 ? :? ,"       ";
520 ? :? :? ,;:INPUT D$:IF D$(1,1)="E" THEN GRAPHICS 0:END
530 IF D$(1,1)="R" THEN RUN
540 TRAP 500:IF VAL(D$)>255 THEN 500
545 DN=VAL(D$):B$=""
550 FOR DIGIT=1 TO 8
560 D2=VAL(D$)/2:D=INT(D2)
570 IF D2=INT(D2) THEN B$(9-DIGIT,9-DIGIT)="0":GOTO 590
580 B$(9-DIGIT,9-DIGIT)="1"
590 D$=STR$(D)
600 NEXT DIGIT
800 POSITION 2,3:? :? ,"DECIMAL ";DN;"=    "
810 ? :? ,"BINARY VALUE=";B$
820 ? :? ,"TYPE E TO END":? ,"TYPE R TO RERUN":? ,"OR TYPE A DECIMAL NUMBER."
830 TRAP 40000:GOTO 510
```
©

Autoloader For OSI

Charles Stewart
Adrian, MI

Almost anyone who has worked with an OSI computer in machine language has asked the question — "How can I save machine language programs?" There are various ways, most are in machine language themselves and load thru the monitor. But if the routines are to be interfaced with BASIC, the simplest way is to utilize the READ from DATA and POKE into memory.

The following routine is a BASIC program that will read a machine language program in memory and produce a BASIC load and go program with the machine language data in basic data statements (already decoded for you from the HEX the monitor requires to decimal).

To use, place the machine language program in memory via the monitor, assembler/editor etc. Unused memory locations in page two $0222 to $02FF in the standard OSI or locations above $0800 may be used. When you have the routine working as you want it, hit BREAK, COLD START, and answer 2048 to the prompt MEMORY SIZE. You should have 1297 Bytes free which is the minimum requirement to run the autoloader routine. You will still have your machine language routine in memory since an answer to MEMORY SIZE by a decimal number eliminates the memory check done by prom on a cold start.

How It Works

LINE 155 Requests the starting and ending addresses of the machine language program you wish to save. Respond with the decimal equivalent of the routine. The computer stores these in variables A and B. Next the program requests line number start and increment. Respond with the line number you wish to start the generated basic poke program with, followed by the line number to increment factor: i.e. a response of 100,10 will generate a BASIC program starting with line 100 followed by 110 and so on. The program stores these variables in D and E.
LINE 175 sets the maximum line length to 255 characters and places your OSI in the save mode.
LINE 180 prints to tape and screen the beginning line number and the statement For X = (decimal number entered as the start of the ML routine) TO (number entered as the end of the machine routine):READ Y:POKEX,Y:NEXT
LINES 210 to 275 are the meat of this program, where we look at the memory locations specified in variables A and B, and strip off the space always returned by basic in case the number may be negative. Then we print line number, the statement DATA followed by the actual data in the specified memory locations.

The Routine Works As Follows:

LINE 210 Sets two FOR NEXT loops from the address set as the start of the ML routine, the step 23 increments the memory addresses for the variable J
LINE 220 Reads the data in memory locations specified by line 210.
LINE 230 Strips off the leading space of the decimal number returned by basic. Not really necessary but saves considerable memory.
LINE 240 Prints line number and the statement DATA.
LINE 250 Prepares for print of the data and tests for the end of the routine.
LINE 251 Prints the machine code in decimal followed by a comma (CHR$(44))
LINE 270 Performs the same function as line 250.
LINE 271 Same as 251
LINE 280 Optional — used to automatically start the generated program when loaded into the computer.
LINE 290 Turns save flag off.

I have utilized this routine for the past 6 months and have found it to be quite a useful utility. It should function as described on most any computer utilizing microsoft basic with minor changes. Routine written on an OSI C1P.

OSI AARDVARK OSI
NOW MEANS BUSINESS!

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (0S65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.
MAXI-PROS — $39.95

THE EDSON PACK ALL MACHINE CODE GAMES FOR THE 8K C1P

INTERCEPTOR —You man a fast interceptor protecting your cities from Hordes of Yukky Invaders. A pair of automatic cannon help out, but the action speeds up with each incoming wave. It's action, action everywhere. Lots of excitement! $14.95

MONSTER MAZE — An Arcade style action game where you run a maze devouring monsters as you go. If one sees you first, you become lunch meat. Easy enough for the kids to learn, and challenging enough to keep daddy happy. $12.95

COLLIDE — Fast-paced lane-switching excitement as you pick up points avoiding the jam car. If you succeed, we'll add more cars. The assembler code provides fast graphics and smooth action. $9.95

SPECIAL DEAL—THE ENTIRE EDSON PACK— ALL THREE GAMES FOR $29.95

THE AARDVARK JOURNAL

FOR OSI USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.
2) Complete listings of two word processors for BASIC IN ROM machines.
3) Moving the Directory off track 12.
4) Listings for 20 game programs for the OSI.
5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues - $9.00
Vol. 2 (1981) 2 back issues and subscription for 4 additional issues - $9.00.

ACCOUNTS RECEIVABLE — This program will handle up to 420 open accounts. It will age accounts, print invoices (including payment reminders) and give account totals. It can add automatic interest charges and warnings on late accounts, and can automatically provide and calculate volume discounts.

24K and 0S65D required, dual disks recommended. Specify system.
Accounts Receivable. $99.95

*** * * SPECIAL DEAL — NO LESS! * * ***

A complete business package for OSI small systems — (C1, C2, C4 or C8). Includes MAXI-PROS, GENERAL LEDGER, INVENTORY, PAYROLL AND ACCOUNTS RECEIVABLE — ALL THE PROGRAMS THE SMALL BUSINESS MAN NEEDS. $299.95

P.S. We're so confident of the quality of these programs that the documentation contains the programmer's home phone number!

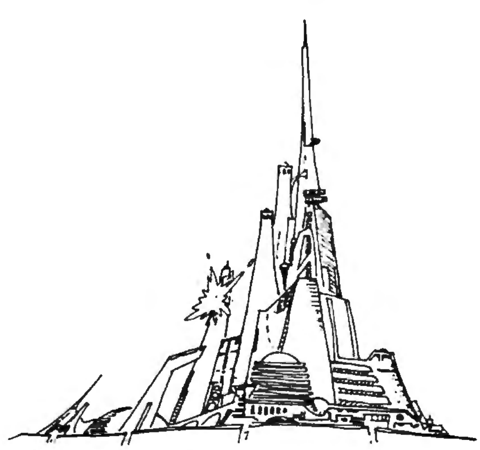
SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for — SUPERDISK II $29.95 (5 1/4") $34.95 (8").

AND FUN, TOO!



BOOKKEEPING THE EASY WAY — WITH BUSINESS I

Our business package 1 is a set of programs designed for the small businessman who does not have and does not need a full time accountant on his payroll.

This package is built around a **GENERAL LEDGER** program which records all transactions and which provides monthly, quarterly, annual, and year-to-date PROFIT AND LOSS statements. GENERAL LEDGER also provides for cash account balancing, provides a BALANCE SHEET and has modules for DEPRECIATION and LOAN ACCOUNT computation.
GENERAL LEDGER (and MODULES) $129.95.

PAYROLL is designed to interface with the GENERAL LEDGER. It will handle annual records on 30 employees with as many as 6 deductions per employee.
PAYROLL - $49.95.

INVENTORY is also designed to interface with the general ledger. This one will provide instant information on suppliers, initial cost and current value of your inventory. It also keeps track of the order points and date of last shipment.
INVENTORY - $59.95.

GAMES FOR ALL SYSTEMS

GALAXIAN - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95

MINOS - 8K - — Features amazing 3D graphics. You see a maze from the top, the screen blanks, and when it clears, you are in the maze at ground level finding your way through on foot. Realistic enough to cause claustrophobia. — $12.95

NEW — NEW — NEW

LABYRINTH - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95.

TIME TREK - 8K - Real Time and Real graphics Trek. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! — $9.95

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95
C1E/C2E similar to above but with extended machine code monitor. — $59.95

Please specify system on all orders

This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.

VISA

OSI

AARDVARK TECHNICAL SERVICES, LTD.
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

OSI

```
 10 REM CHARLES A STEWART
 20 REM 3033 MARVIN DR.
 30 REM ADRIAN MI 49221
 40 REM 517-265-4798
 50 REM AUTOLOAD PROGRAM FOR OSI C1P
 80 POKE133,0:POKE134,8
 90 DIMA$(24),B$(24):POKE15,0
 100 FORX=0TO40:PRINT:NEXT:PRINT" AUTO L
OAD OF MACHINE LANG PROG"
 110 PRINT" IN PAGE 2 OR MEMORY LOC ABOV
E $0800
 120 PRINT"PROGRAM REQUIRES 2047 BYTES T
O OPERATE
 150 FORX=1TO10:PRINT:NEXT
 155 INPUT"START,END ADDRESS IN DECIMAL"
;A,B
 170 PRINT:PRINT:INPUT"SOURCE PROGRAM LI
NE # START & INC";D,E
 175 POKE15,255:SAVE
 180 PRINTD;"FORX="A"TO"B":READY:POKEX,Y
:NEXT"
 210 FORI=ATOBSTEP23:FORJ=0TO22
 220 A$(J)=STR$(PEEK(I+J))
 230 A$(J)=RIGHT$(A$(J),LEN(A$(J))-1):NE
XTJ
 240 D=D+E:PRINTD;"DATA";
 250 FORJ=0TO11:IFI+J>BTHEN280
 251 PRINTA$(J);CHR$(44);:NEXT:PRINTA$(J
):
 260 D=D+E:PRINTD;"DATA";
 270 FORJ=13TO21:IFI+J>BTHEN280
 271 PRINTA$(J);CHR$(44);:NEXT:PRINTA$(J
):
 275 NEXTI
 280 PRINT:PRINT"POKE515,1:RUN"
 290 POKE517,0
```

Program Listing

```
START,END ADDRESS IN DECIMAL? 0,222

SOURCE PROGRAM LINE # START & INC? 100,1
0
 100 FORX= 0 TO 222 :READY:POKEX,Y:NEXT
 110 DATA76,116,162,76,195,168,5,174,193
,175,76,136,174
 120 DATA0,0,255,56,17,0,49,48,48,44
 130 DATA49,48,0,0,69,0,49,55,44,48,32,0
,78
 140 DATA34,0,75,0,53,49,53,44,49,58
 150 DATA82,85,78,34,0,82,73,78,84,65,36
,40,74
 160 DATA41,58,32,0,84,34,32,0,82,84
 170 DATA32,38,32,73,78,67,34,59,68,44,6
9,32,0
 180 DATA71,34,0,177,128,128,11,96,171,3
4
 190 DATA58,0,0,0,0,0,0,0,0,104,101,0,1
 200 DATA249,6,165,143,174,225,141,32,8,
6
 210 DATA247,1,32,25,0,251,1,3,226,5,12,
6,226
 220 DATA6,115,7,106,7,0,8,220,0,155
 230 DATA0,236,4,164,237,0,3,25,0,74,0,8
,6
 240 DATA71,6,255,164,0,83,0,104,0,4
 250 DATA76,30,180,19,6,227,5,0,0,6,6,13
6,0
 260 DATA0,175,33,0,0,136,161,0,0,33
 270 DATA56,0,8,0,230,195,208,2,230,196,
173,255,4
 280 DATA201,58,176,10,201,32,240,239,56
,233
 290 DATA48,56,233,208,96,128,79,199,82,
47,140,164,171
 300 DATA5,229,231,
POKE515,1:RUN
```

©

Example 1

Part One Of Two

OSI C1P Newspaper Route Listing Program

Charles L. Stanford
Cinnaminson, NJ

This program, like most, started out as a very simple task to fulfill a stated need. And like too many, it got very, very complicated. My son, John, has a paper route. In a big city suburb, newspaper routes are very volatile; the customer list changes as the promotions of the various papers attract readers, and as the residents move on with their corporations. So the route list is hard to keep current. Each day off requires a new hand-written list for the sub (too often Dad). A Paper Route program seemed like a natural. And the program was easy to write. It started out in much the same form as listed here. The data save method is similar to the one in **COMPUTE!**, Issue 2, "Home Accounting" article, with the exception that I added $trings for the customer's names. All seemed to be fine. But then the bug showed up. The program wouldn't save $trings to data statements when new customers were added! Everybody ended up with the same name.

A week (and a lot of POKEing around in RAM) later, I knew one heck of a lot more about my C1P's method of storing variable arrays, and the program ran. I think that a quick review of what I learned, and how the computer can be "fooled" by some $tring manipulation tricks, will be useful to many readers.

Microsoft BASIC Source Code Storage

Much has been written on the method Microsoft

BASIC uses to store programs. I think one of the best explanations if found in Edward H. Carlson's book "OSI BASIC In ROM". To simplify, the source code is stored in RAM starting at Hex address $0300. The first byte is 00. The next two hold the address of the next line, in the standard notation of lo byte first, hi byte second. To convert to decimal, multiply the decimal value of the second byte by #256 and add the value of the first. The next two bytes are the line number, in the same form. For example, line 100 would read 64 00 (the Hex value of Dec 100 followed by 0 * 256). Now comes the line itself, with the BASIC commands in their token form and all other information represented by its ASCII value. See Table 1 for the representation of a typical line.

Each successive line is ended by a 00, and each new line starts as above. The last line is followed by three bytes of 00. Next comes the variable table, with the simple variables stored first. The numeric variables are stored in four-byte floating point binary. I won't go into that here, except to say that a decimal number is represented in a manner similar to a logorithm, with the characteristic (exponent) first and the mantissa (base value) next. The $tring variables are stored in a much different manner. The second byte of a $tring variable is the ASCII value of the second character of the variable plus $80 (Dec 128). Where the next four bytes of a numeric variable are the value for the numeric, they are, for a $tring, the length of the $tring; the address of the location of the actual $tring elsewhere in memory; and 00 to end the variable.

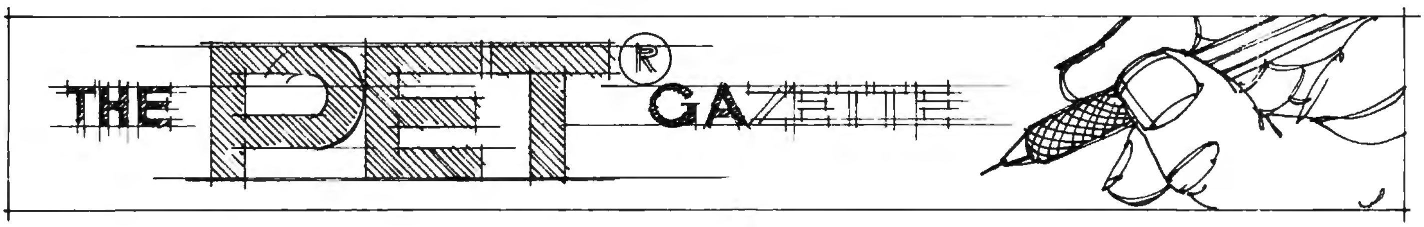
This latter characteristic is what brought me to not inconsiderable grief. The same difference exists for the storage of numeric and $tring arrays. Arrays start a bit differently, but the idea is the same. The first seven bytes define the array. For a string array, they are as shown in Table 1. The array used is dimensioned at two, which will give it three elements (remember that "0" is a place for a computer). In addition, you must remember that non dimensioned variables default to ten. Thus they have eleven elements, counting the 0th to the tenth. The third byte of each array is a pointer so the program can easily find the next array without searching through every element of each. It represents 7 + (No. of elements) * 4, which is the number of bytes to the next array. The fourth through seventh bytes contain $00, $01, $00, and (No. of elements + 1), respectively.

Next is the elements of the array, with four bytes each. They are, in order, the length of the $tring for that element; the address of the $tring elsewhere in memory; and 00 to end the element. If the $tring's value is established in the source code, whether in a DATA statement or as a $tring constant, its location stays with the source code. If the $tring is etablished during the run of the program, by keyboard input or through $tring manipulation, it is placed in high memory, working from the top of RAM down. However, you can fool the program. By concatenating a $tring with a zero length string, the BASIC routine thinks a new $tring has been established, and puts it at top of memory as well as in source code. A$ = A$ + "" does it. The disadvantage is that the $tring is now in two places, with attendant use of extra memory.

But why would you want to do this? One reason came to light during the creation of the Paper Route program. When a new customer is added, the routine at Line 525 of Listing 2 opens a space, and readdresses all of the Name $trings from the insertion location up. This means that the $tring in source code which used to be N$(X) is now N$(X + 1), and so on. Everything works fine, as the new N$(X) is INPUT and placed at top of memory. The problem arises when you try to save all the $trings, old and new, to DATA by the routines between Lines 800 and 995. I at first tried to save from N$(1) to N$(75) in all cases. It worked whenever customers were deleted. But if customers were added, everyone from the new one to the end had the new person's name. As it turned out, the program was trying to pick itself up by its own bootstraps!

If a customer is added at number 3, then old customer 3 becomes number 4. But the name is still stored in the third DATA statement. Now you start to rePOKE the DATA statements, from 1 to N. What happens? Old 3 is replaced by new 3. But now you try to read new 4 (which was old 3), and you, instead, get new 3. Sounds simple. But it sure was perplexing until I reached a fairly complete understanding of the $tring variable storage system discussed above.

Several solutions appeared possible, with the easiest to just concatenate each string. But that turned out to use up more than my 8K of RAM with 75 customers. The best answer seemed to be to reverse the order for adds and deletes. The disadvantage here is time; it takes about 35 seconds for the save routine, and it must be done once each for adds and deletes. On the other hand, more efficient solutions would involve machine language routines or complicated $tring manipulations. ©



Editor's Note: I've been intending to do this for several months, but Liz's closing comments, and this excellent summary, helped provide the final motivation. Following this article you'll find the full texts of her references (2, 3, 4). These are unavailable since these issues are out of print. The other references are all still available, either at your local dealer or from **COMPUTE!** *Robert Lock*

Relocation Of Basic Programs On The Pet

Elizabeth Deal
Malvern, PA

This article shows an easy way to relocate any Basic program by use of the APPEND command of the TOOLKIT(tm). It is written for the upgrade ROM Pet but the idea is transferable to all Pets. Palo Alto ICx TOOLKIT is used. The method should, however, work with the Skyles chip, as well as with any append or merge machine code program.

Background

In the past two years, several writers have shown how to partition the Pet into little Pets. If you are unfamiliar with this exciting development, read all of the references listed at the end. All of the necessary programming tricks have been described thoroughly in those articles.

Herman and Brannon described one of the reasons why one may want to partition the Pet's memory — quick access to different programs. The key reason why I value the idea of partitions is that I like to use several utilities during the program development stage without having to save or delete those utilities each time I save a program I am working on. The kind of utilities that come to mind are tape indexing systems, base conversions, logical bit operations and whatever other debugging aids are needed at the time. Though I have converted some to machine code, and can place them out of the way of a Basic program, others are just too tough to convert. Hence the value of at least two partitions in my Pet.

Whatever the reason, the procedure involves dividing the Pet's memory into smaller units. Herman showed a way to do it in Basic (2). Brannon did it in machine code (4). Hudson demonstrated a way of allocating space for machine code where Basic programs usually live, while parking a Basic program upward in memory.

Once the partitions are established, the desired Basic programs can be written into and saved from the partitions. This method, however, does not permit us to load an *existing* program into a partition other than at 1024 or $0400 hex. Old Pet owners got some relief when Young provided a relocation program (7). New Pet owners were out of luck. The Retyping of old programs (being prohibitively expensive activity) implied something better had to be found. The missing link was the forward pointer chaining which Harvey Herman explained in his "Hooray for SYS" article. And this has pointed me into the simplest solution of all.

How To Relocate The Lazy Way

It is clear from reading the references that the necessary conditions for program relocation are to:

1. set up one or more partitions
2. adjust all Basic pointers
3. change the tape header information
4. correct the forward pointers in the relocated program.

It is equally clear to me that the easiest way to take care of these tasks within my existing system is to use the APPEND command of the TOOLKIT. This is not an exotic solution, since I understand that a large proportion of Pet owners have some form of the TOOLKIT. Here is how to do it:

1. partition the Pet via the Brannon, Herman or Hudson method, paying careful attention to all required pointer adjustments, else face an unbelievable mess,
2. type NEW — don't ever forget this,
3. type APPEND "name" and wait for the load to complete.

That's all there is to it. The four conditions necessary for relocation have been met and the Basic program now resides in a place of your choice. The TOOLKIT didn't know, or care, that it appended a program to a nonexistent program.

Assorted Comments

There are several limitations that, at the moment, do not cause me any trouble, but you may have to face them. The APPEND command of my TOOL-

80 COLUMN GRAPHICS

The Integrated Visible Memory for the PET has now been redesigned for the new 12" screen 80 column and forthcoming 40 column PET computers from Commodore. Like earlier MTU units, the new K-1008-43 package mounts inside the PET case for total protection. To make the power and flexibility of the 320 by 200 bit mapped pixel graphics display easily accessible, we have designed the Keyword Graphic Program. This adds 45 graphics commands to Commodore BASIC. If you have been waiting for easy to use, high resolution graphics for your PET, isn't it time you called MTU?

The image on the screen was created by the program below.

```
10 VISMEM: CLEAR
20 P=160: Q=100
30 XP=144: XR=1.5*3.1415927
40 YP=56: YR=1: ZP=64
50 XF=XR/XP: YF=YP/YR: ZF=XR/ZP
60 FOR ZI=-Q TO Q-1
70 IF ZI<-ZP OR ZI>ZP GOTO 150
80 ZT=ZI*XP/ZP: ZZ=ZI
90 XL=INT(.5+SQR(XP*XP-ZT*ZT))
100 FOR XI=-XL TO XL
110 XT=SQR(XI*XI+ZT*ZT)*XF: XX=XI
120 YY=(SIN(XT)+.4*SIN(3*XT))*YF
130 GOSUB 170
140 NEXT XI
150 NEXT ZI
160 STOP
170 X1=XX+ZZ+P
180 Y1=YY-ZZ+Q
190 GMODE 1: MOVE X1,Y1: WRPIX
200 IF Y1=0 GOTO 220
210 GMODE 2: LINE X1,Y1-1,X1,0
220 RETURN
```

K-1008-43M Manual only $10 (credited toward purchase)
k-1008-43 Complete ready to install package $495

Mastercharge and Visa accepted

Write or call today for our full line catalog describing all MTU 6502 products, including our high speed 8" Floppy Disk Controller for up to 4 megabytes of PET storage.

MTU
Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605 U.S.A.
(919) 833-1458

NOW 80 COLUMN PETS CAN HAVE MTU HIGH RESOLUTION GRAPHICS

KIT works only with tape systems. It also works only in direct mode. Thus if you do not have a tape system or if you want to relocate Basic programs during a program execution (insanity!) another *machine code* appending program will have to be used. In both cases the original or somewhat modified Wollenberg (3) procedure should do the job. Please, write it up for **COMPUTE!**.

Do not forget that as the code is moved from one place to another, all saving must be done via the Machine Language Monitor, and not by the normal SAVE command.

A lot of complicated system designs can be done by use of partitions. However, the price we pay is the need for immaculate housekeeping. Forgetting even one little detail can cause a lot of grief. You may literally lose your program, you may crash and you may have a very confused Pet. Special attention needs to be paid in relocating programs which change Basic pointers during execution. Such changes must be transparent or must be communicated in some fashion to the "partition supervisor". In the Quadrapet, for example, pointer values will have to be reflected, at correct times, in the table of pointers in the tape buffer.

Relocation by appending permits us to lift one of Hudson's restrictions. If needed, we *may* now increase the size of the machine language partition in mid-stream by saving the Basic program via the Monitor and relocating it by the procedure described here.

As a result of this work I think I found one typo in the Quadrapet — line 1080 needs "32" where "2" now exists.

Thanks To All

One of the features of **COMPUTE!** I treasure is that it provides a forum for *exchange* and growth of better and better ideas. Robert Lock should be congratulated for *not* taking a "we already published it once" stance, for such a position usually stifles any possibility of improvements. Without the sources listed below relocation of Basic programs would still be just a wish.

References

(1) Butterfield, *Watching A Cassette Load*, Pet User Notes, vol. 2, #1

(2) Herman, *Memory Partition Of Basic Workspace*, **COMPUTE!**, Jan.-Feb. 1980

(3) Wollenberg, *Machine Language Code For Appending Disk Files*, **COMPUTE!**, July-Aug., 1980

(4) Brannon, *Quadrapet*, **COMPUTE!**, Sept.-Oct., 1980

(5) Herman, *Hooray For SYS*, **COMPUTE!**, Jan. 1981

(6) Hudson, *An 'Ideal' Machine Language Save For The Pet*, **COMPUTE!**, Jan. 1981.

(7) Young, *Relocate*, **COMPUTE!**, Feb. 1981 ©

Shortcuts to more Commodore capability

4 amazing new devices to make your Commodore friendlier, faster and more productive than ever

If you have a Commodore computer, life at the keyboard just got easier, thanks to four new products from Kansas City Computers, Inc.

THE SILENT SWITCH

If you use a switching device on your ROM based software, you may have experienced —select line transient problems.

Introducing our **Triple Flip** firmware. It's a remarkably simple switch that can be electronically *mechanically* switched. With the external switch running to internal switching, there are no address lines outside the computer. That nips select line transient problems right at the source.

Triple Flip lets you place three individual ROMS in a single ROM socket. A single switch can control up to seven **Triple Flips.** That gives you 21 ROM capability.

Versatile? **Triple Flip** is one-socket ROM selectable for Commodore computers 2716, 2532 and 2732. It also fits Pet 2001B, N and later models.

And like all KCCI products, it's backed by our full guarantee and 90-day warranty.

Don't change chips everytime you switch to another program. And don't risk select line transient problems. Put **Triple Flip** to work for you for just $39.95.

30 NEW COMMANDS

Until now, you've had to punch the keys 24 times or more to put some commands into action. Commands like transferring programs to disk or paper, or appending program modules.

Order our **UtiliRom** and start using your computer's memory instead of your own. **UtiliRom** includes an enhanced DOS universal wedge that lets you make most of your present commands in two keystrokes. Imagine having all this at your command:

- Wild card directory (DOS)
- Rename a file (DOS)
- Wild card file scratch (DOS)
- Copy files on disk (DOS)
- Reset disk system (DOS)
- Initialize 2000 series drives (DOS)
- Check error channel (DOS)
- New a disk (DOS)
- Validate a disk (DOS)
- Scroll down
- System cold start
- One key command to load a program (DOS)
- Send program listing to printer (with* or without* form feed at end)
- Send screen contents to printer (normal mode* or squeezed*)
- Send screen contents to disk file by any name*
- Disk program append*
- Repeat key function*
- Kill to turn off repeat*
- Escape to turn off ROM*
- Convert hex to decimal or
- Convert decimal to hex (with error detection)
- Fast jump to monitor
- Fast shift to upper or lower case
- Fast jump to cold start
- One key command to save a program
- Beep (programmable)*

**Asterisk indicates routines which can be called in basic as subroutines for increased computer power.*

Warning! Extensive testing has proven that even casual users of **UtiliRom** become hooked on its efficiency and speed. Serious programmers and custom software designers appreciate the fact that routines are included for fast disk I/O — up to 3 times faster than the system routines. In addition, **UtiliRom** can help protect software designs from theft.

UtiliRom firmware is available for any socket on any Commodore system (including the Address 9000). When used with **Chipmate** firmware (below), you can use UtiliRom in the same socket simultaneously with one other 2K ROM such as Toolkit™.

Order **UtiliRom** now at just $49.95.

Order **Chipmate firmware with UtiliRom** for same socket simultaneous use with most other 2K ROMs. **Chipmate with UtiliRom** is only $61.90. **Chipmate alone** is $14.95.

COMPUTERIZED SCRATCH PAD

Dream up anything you want and put it on your screen. Punch a couple of keys and put it on disk in less than two seconds. Retrieve it anytime, almost as fast as you can visualize it. Hit two more keys and it's generated on your printer in page-perfect format.

You're not dreaming. You're using **Screen Pro** software, the new disk based system that adds exciting new powers to your computer — and a faster response than ever before.

From compiling grocery lists to editing catalogs, from diagramming football plays to animating them, **Screen Pro** makes text and diagram work incredibly simple.

Screen Pro's entire 2000 character screen is displayed in a fraction of a second. You can easily place this subroutine into any software you write, for quick access later. In fact, an eight line program gives you access to any one of 10,000 pages of information — all within four screens!

Clear, understandable documentation shows you how to tailor **Screen Pro** to your exact needs. And it opens your eyes to uses limited only by your imagination — for only $39.95.

MONEY BACK GUARANTEE

All four products come to you with complete, easy-to-follow documentation. Installation instructions that are a snap. Plus step-by-step examples that walk you through the process.

You get more than documentation with your KCCI order. You get dependability Dependability backed by our Money Back Guarantee. Plus our full 90-day Service Warranty (that's three times as long as most of our competitors' warranties).

Start investing less time in your computer — and start getting more out of it. Fill out and mail this coupon with your check today.

Or see your local Commodore dealer.

Toolkit™ is a trademark of Palo Alto ICS Inc.
™Commodore Business Machines

KCCI

Kansas City Computers, Inc.
5214 Blue Ridge Boulevard
Kansas City, Missouri 64133 • Phone 816/356-6502

Name____________________
Address____________________
City__________ State______ Zip______
Computer make__________ Model______
Disk system make__________ Model______

YES! Please send me these shortcuts for more computer capability. My check is enclosed.

☐ TRIPLE FLIP	$39.95	☐ CHIPMATE WITHOUT UTILIROM	$14.95
☐ UTILIROM	$49.95	☐ SCREEN PRO	$39.95
☐ CHIPMATE WITH UTILIROM	$61.90	☐ TOTAL ORDER	

(Add $1.50 shipping & insurance for U.S. orders. Add $3.10 shipping for foreign orders — payment must be in U.S. funds. In Missouri, add 4% sales tax.)

Memory Partition of BASIC Workspace

Harvey B. Herman
Chemistry Department
University of North Carolina
at Greensboro

A 6502 microprocessor can address a total of 65K bytes of memory (RAM plus ROM). The address space for BASIC programs (RAM) is necessarily restricted to less than that without resorting to hardware tricks. However, most BASIC programs do not take up anywhere near the maximum amount of reserved memory (32K bytes for the PET). Occasionally it would be useful to have several short noninteracting BASIC programs in memory at the same time. For example, we use short programs to check student laboratory calculations (J. Chem. Ed., Vol. 55, p. 654 (1978)). When multiple laboratories are in process it would be simpler to LOAD a tape containing a number of programs and have each student run the program appropriate for his experiment.

One way to combine programs is to renumber and merge individual programs with a subsequent re-save of the combination. There are several disadvantages to this approach. It is important to keep line numbers separate in each program to be merged else you may not be able to delete or LIST parts of the program (unnerving at first). An ordinary LIST of the program will show frequently unrelated parts as one program (not esthetically pleasing). The student user must remember to RUN with a line number specified for his chosen segment (or risk being hopelessly confused. Finally this approach will not allow placing utility programs (written in BASIC) in reserved areas of memory unless they are merged with every program (a formidable task).

Since I frequently use a number of short programs and have unused memory I thought it would be helpful to partition the BASIC workspace for storage of individual programs. For example, an 8K PET (7167 bytes free) could have three 2K partitions under control of a 1K master program. It is possible to make other configurations as long as the total does not overrun the free memory available. If the partitioning is done properly the stored programs would not interact with each other. Each program would "think" it was in a 2K PET. (I actually owned a 2K PET once when I had a memory failure.) The master program would be in charge of adjusting the necessary pointers so a given program could be accessed when requested by the user.

Microsoft BASIC (for the PET and other microcomputers) uses pointers to subdivide free memory. The table summarizes important pointers (at least for this discussion) for both old and new PETs. The following material is for the old ROMs. It is not necessary to do any hex arithmetic to use the method I will describe. However, it does help to understand a little about pointers. If BASIC program text is stored beginning at location hex 401 (it is assumed location hex 400 contains a zero) the pointers to start of text (location 122/123) would read 1 and 4 for low and high byte respectively. That example was not too difficult but it must be remembered that the value returned is in decimal. If start of text was changed to, say hex 1001, location 123 would now read 16 corresponding to the decimal representation of the most significant half of that number (hex 10). To activate a new partition it is only necessary to set pointers to start of BASIC text (122/123), end of BASIC text (124/125) and top of memory (134/135). Subsequently executing CLR will set all the other pointers automatically (e.g., bottom of strings, etc.) and after END we find ourself in the new partition.

As an exercise I wrote a short master program (1K workspace) controlling three short donothing BASIC programs (each in a 2K workspace). They are shown in the figure. The master program asks the user for a program number and automatically sets the pointers to activate that program. At this point the user is in a 2K workspace with one program active which can be RUN or modified as desired. The last statement in each of the short programs returns the user to the master program. Each program is completely independent of the others, snug and protected in its own private world.

Setting up the example or one like it is not difficult. Each program could be typed in after the partition is activated by the master program (NEW first). Keep track of the size of each program by PEEKing at locations 124 and 125. This information should be stored in the master program so one can enter and leave the partition without destroying the BASIC text (c.f., line 210 in master program). The size of the master program should also be recorded and restored before returning to it (c.f., line 40 in program 1).

Relatively long programs are a nuisance to type into each partition. If the program is on cassette tape it can be relocated to any partition using the procedure described in my article "MOVE IT" (MICRO 16:17 and 17:18). Normally tapes load starting at hex 400. By reading in the tape header first and changing the load parameters in the tape buffer information on cassette tape can be stored elsewhere in memory. Keep two points in mind. One, before using the relocated programs for the first time the BASIC line links (see p. A-9 in PET

WORD-CHECK

WORDCHECK is a poor spellers dream come true. Designed to interact with WORDPRO, it has 2100 root words and suffixes. In addition for the business and scientific user it has the capacity for 900 industrial or scientific terms which you load in yourself. You have a total vocabulary of approximately 7500 words at your fingertips. It simply goes through the text and flags any words that it doesn't recognize.

WORDCHECK is the ideal program to proof your spelling, whether it is one paragraph or a 100 page manual. The dictionary is versatile, allowing the user to add or delete words. You can design the program with the technical terms your profession uses, even duplicating the table and tailoring it for each person in your office. Let WORDCHECK do the work for you quickly and accurately.

CREATE-A-BASE

CREATE-A-BASE is a data base file management system that enables the user to choose the number of fields needed in a file, and add or delete fields without disturbing any of the existing data. Once a file is created you can perform any of 30 functions. Such as:

- Interact with WORDPRO 4, and 4+
- Do mathematic functions on any 2 or more fields
- Sort 650 files in only 19 seconds
- Merge any sequential file into a **CREATE-A-BASE** file, and output a sequential file from a **CREATE-A-BASE** file.
- The report generator has the feature of user defined fields and field width.
- Printouts can be generated by values such as, greater than, less than, equal to or in alpha or numeric codes.

You don't have to be a programmer to operate **CREATE-A-BASE** on your COMMODORE computer. It's menu driven and asks you questions at each step as you perform any of its many functions.

AVAILABLE

at your local COMMODORE dealer or
distributed exclusively in CANADA by
B.P.I. Micro Systems, Ltd.
80 Barbados Blvd. #14
Scarborough, Ontario M1J1K9
Special Dealer Introductory Package Available

Micro Computer Industries Ltd.

1520 E. Mulberry, Suite 170 Fort Collins, CO 80524

1-303-221-1955

User Manual) must be corrected. The easiest way to do this is to type any line number *not* in the program and return. Two, record the length of the program by PEEKing at locations 124 and 125 after an *ordinary* tape load. In my example program 1 showed 4 and 74 respectively. Since I intended to relocate the program to a partition beginning at hex 800, it was necessary to use the values 8 and 74 in line 210 in the master program.

The partition idea described above should be applicable, with only minor changes, to any microcomputer using Microsoft BASIC. In fact I used a partition for the first time on my SYM to store an initialization program which was used infrequently. In this case the partitions were of unequal length, 4K and 8K. Readers might be interested in storing their short BASIC utility programs in an out of the way partition and activate the programs when necessary as I did with the SYM initialization program. Maybe others could share their ideas on the subject with me care of this magazine. We could publish the best ones in a future article. (Anyone for time sharing?)

Important PET Pointers (Low/High Bytes)

	1107 Old ROM	Upgrade ROM
Start of Text	122/123	40/41
End of Text	124/125	42/43
Top of Memory	134/135	52/53

```
10 REM MEMORY PARTITION-MASTER PROGRAM
20 REM  EXAMPLE:
30 REM THREE PROGRAM WORKSPACES
35 REM CREATED AT:
40 REM HEX 0800-0FFF   PROGRAM 1
50 REM HEX 1000-17FF   PROGRAM 2
60 REM HEX 1800-1FFF   PROGRAM 3
65 POKE 123,04:POKE 122,01
66 POKE 125,06:POKE124,57:CLR
67 POKE2048,0:POKE4096,0:POKE6144,0
70 :
80 REM HARVEY B. HERMAN
90 :
95 PRINT "WHICH PROGRAM DO YOU WANT";
100 INPUT"(1-3)";N
110 ON N GOTO 200,300,400
200 POKE 123,08:POKE122,01
205 POKE135,24:POKE134,0
210 POKE 125,08:POKE 124,74:CLR:END
300 POKE 123,16:POKE122,01
305 POKE135,24:POKE134,0
310 POKE 125,16:POKE 124,74:CLR:END
400 POKE 123,24:POKE122,01
405 POKE135,32:POKE134,0
410 POKE 125,24:POKE 124,74:CLR:END
```

```
10 REM PROGRAM 1
20 A=1
30 PRINT A
40 POKE123,4
45 POKE124,57
50 POKE125,06
55 POKE135,8
60 CLR:END
```

```
10 REM PROGRAM 2
20 B=2
30 PRINT B
40 POKE123,4
45 POKE124,57
50 POKE125,06
55 POKE135,8
60 CLR:END
```

```
10 REM PROGRAM 3
20 C=3
30 PRINT C
40 POKE123,4
45 POKE124,57
50 POKE125,06
55 POKE135,8
60 CLR:END
```

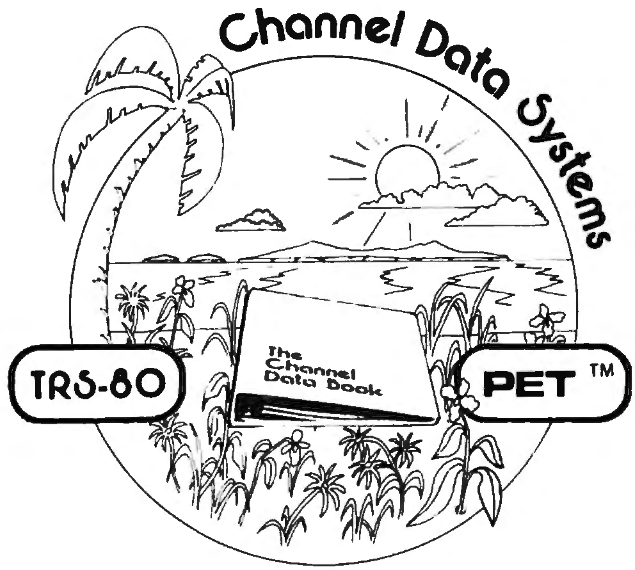
©

Machine Language Code For Appending Disk Files

Robert H. Wollenberg
Department of Chemistry
Stanford University

The attached machine language routine provides a very useful tool for those who have been frustrated (as I have) at the inability of the current disk operating system (DOS) of the Commodore 2040 Dual Drive Disk to append programs. Although firmware recently introduced by Palo Alto IC's (The Programmer's Toolkit) provides some relief to those who require a convenient appending procedure, it suffers a serious drawback. The system operates only by appending a tape file to a program already in memory. Thus, it becomes necessary to first save program pieces to tape and then reload these in proper sequence using the append command. Since I was reluctant to use this slow tape file procedure, I searched for alternatives involving disk files.

A little investigation of the DOS command, Copy, reveals that this instruction goes a long way toward solving the problem. Basic is stored in memory starting at location $0401. The first two bytes are forward



OMNIFILE — CBM or TRS-80 — $30

Omnifile is a versatile, in-memory database program with sorting, formatting, and computational features. Records can be entered, edited, and processed with a single letter command. **Omnifile** applications include inventory records, mailing lists, sales journals and collection lists. Records can be stored on the Commodre floppy disks or on the tape cassette. **Omnifile** uses approximately 6k of RAM memory. Up to 500 records can be contained in memory in a 32k CBM at any time. Multiple files are easily accessed from disk or tape. Items can be sorted, moved, inserted and reformatted. Calculations can be made and totals can be printed. The **Omnifile** package includes the program with sample data, listing and manual, and will operate on the large keyboard Commodore PET or CBM computers with at least 16k memory. Also available on diskette for $36. An abbreviated version, **Data Logger,** requiring only 1k of RAM is available on cassette for $15.

GENERAL LEDGER/PERSONAL LEDGER — CBM or TRS-80 — $30

General Ledger is a complete double entry bookkeeping system with provisions for budgeting and keeping records of income, deductible and non-deductible expenses, assets and liabilities. Simple interactive features allow entering transactions, adding or editing accounts, and printing of a detailed income statement and balance sheets. Data can be stored on the Commodore floppy disks or cassette. **General Ledger** occupies about 6.2k of RAM memory, allowing approximately 200 accounts on a 16k machine. Transaction files can be accessed by our **Omnifile** database program for complete analysis, sorting by date, account number, etc. The **General Ledger** program will operate on the new Commodore PET or CBM microcomputer systems and comes with sample data, listing, and manual. Also available on diskette for $36. An abbreviated version allowing about 35 accounts on a 1.0 or 2.0 BASIC 8k PET is available on cassette for $20.

EXPLORE — CBM — $15

Inspired by the computerized fantasy simulation "Adventure," **Explore** is a conversational program which operates on the Commodore PET with only 8k bytes of memory. **Explore** contains four adventures in which you operate a computerized tank, hunt treasure in a magic cave, explore the mall in Washington D.C., and survive in a haunted castle. **Explore** package includes introduction, five data files, and complete manual. Available from Channel Data Systems on cassette for $15. Indication of old or new ROMs is requested.

CHANNEL DATA BOOK for PET — CBM — $20

A complete hardware and software reference service listing descriptions for over 1400 software programs and over 200 peripheral devices for PET. Also includes an information sources section, and addresses for Commodore Dealers in the USA and Canada, Commodore Vendors, and PET user groups. Designed to organize documentation, newsletters, listings, and other user selected information in an attractive 3-ring binder.

ACCESSORIES

PORT NOISE COMPLETE	Speaker-amplifier for Commodore PET or CBM with connector (M & N pin) all ready to plug in—$20.
IEEE/USER PORT CONNECTOR	24 pin connector. With backshell—$7. Without backshell—$4.
C-10 TAPE CASSETTES	High quality AGFA tape with screw type case. $10/10 or $75/100.

Cash, Checks, Money Orders, VISA, Mastercharge
Add $2 shipping for each order—COD-$3.50
For foreign orders please inquire for pricing

CHANNEL DATA SYSTEMS

4141 State Street Santa Barbara, CA 93110 805-964-6695 pointers to the next line of code (stored by the usual 6502 convention of low byte/high byte). Locations $0403 and $0404 store the line number for this first line of basic. The ASCII Code for this line is stored beginning at location $0405 and ends with the delimiter zero. The next byte begins the second line of basic and is stored at the location pointed to by the forward pointers at locations $0401 - $0402 described above. By following the forward pointers from line to line, one eventually reaches the end of the basic code. This event is signaled when the forward pointers are zero.

When the DOS Copy command concatenates two disk files, the zero page values indicate that both programs were combined; however, listing the concatenated program reveals only the first program. The second program for all purposes remains invisible to the basic interpreter. This is because the forward pointers of the first program eventually point to the two delimiters, zero, and at this point the end of basic is signaled. The next two bytes are pointers to the start of basic text for the second program as originally saved to disk. If the first line of the second program is moved forward in memory by four bytes, then the last line of the first program will point to the first line of the second program and the linking process will be nearly complete. In order to finish the linking procedure, the forward pointers of the second program must be

```
0000                    * = $033A

033A  A9 FF    START    LDA #$FF       ;SETUP TO FIND PROGRAM DELIMITER
033C  85 11             STA $11
033E  85 12             STA $12
0340  20 2C C5          JSR $C52C      ;ROM FIND LINE # ROUTINE
0343  20 6C 03          JSR ENDBAS     ;CHECK FOR END OF BASIC
0346  F0 21             BEQ DONE       ;YES, EXIT AND PRINT READY
0348  A5 5C             LDA $5C        ;CURRENT LINE
034A  D0 02             BNE SKIP
034C  C6 5D             DEC $5D        ;MOVE BACK ONE BYTE
034E  C6 5C    SKIP     DEC $5C
0350  A9 00             LDA #$00
0352  A0 04             LDY #$04       ;SETUP FOR 4 LOOPS
0354  91 5C             STA ($5C),Y    ;STORE NEW PROGRAM DELIMITER
0356  A9 20             LDA #$20       ;LOAD ASCII BLANK
0358  88                DEY
0359  91 5C    AGAIN    STA ($5C),Y    ;STORE BLANKS
035B  88                DEY
035C  10 FB             BPL AGAIN
035E  20 72 C5          JSR $C572      ;ROM CLR ROUTINE
0361  20 42 C4          JSR $C442      ;ROM FIX CHAINING ROUTINE
0364  20 6C 03          JSR ENDBAS     ;CHECK FOR END OF BASIC
0367  D0 D1             BNE START      ;DO NEXT PROGRAM IF MORE
0369  4C 8B C3 DONE     JMP $C38B      ;EXIT TO BASIC
036C  38       ENDBAS   SEC            ;CHECK FOR END OF BASIC ROUTINE
036D  A5 2A             LDA $2A        ;TOP OF VARIABLES
036F  E5 5C             SBC $5C        ;CURRENT LOCATION
0371  C9 02             CMP #$02       ;DIFFERENCE = 2 IF END
0373  D0 04             BNE NOTEND     ;RETURN IF NOT DONE
0375  A5 2B             LDA $2B        ;CHECK HIGH ADDRESS
0377  C5 5D             CMP $5D
0379  60       NOTEND   RTS
```

```
0 REM ROBERT H WOLLENBERG
10 REM PROGRAM TO APPEND USING DISK CONCATENATE COMMAND
15 DEF FNA(X)=PEEK(X)+256*PEEK(X+1):DEF FNB(X)=FNA(X)-X
20 I=FNA(40):E=FNA(42):X=FNB(I)
25 IF I+X=E-2 THEN PRINT"END OF MEMORY-NO LINK":END
30 I=I+X:IF FNA(I)<>0 THEN X=FNB(I):GOTO 25
35 FOR J=I TO I+3:POKEJ,PEEK(J+4):NEXT
40 FORJ=I+4 TO I+7:POKEJ,32:NEXT: Y=I-1025+4 :REM OFFSET
45 X=FNB(I)+Y:POKEI+1,INT((I+X)/256):POKEI,I+X-256*INT((I+X)/256)
50 I=I+X:IF FNA(I)<>0 AND I<E-2 THEN 45
55 IF FNA(I)=0 AND I<E-2 THEN 35
60 PRINT"LINK-COMPLETE":END
READY.
```

recalculated to compensate for the relocation of code. This is done conveniently using the ROM chaining routine at $C442.

To use the attached machine code, first use the disk Copy command to concatenate up to four program pieces. For example, to append parts A, B and C to form program D, the following command is executed (after loading the DOS support):

►C0:D = 1:A,1:B,1:C

Next load the machine code and the concatenated program D into memory and type:

SYS826

The programs are linked and the message: "READY." appears.

For comparison, I have written a basic program to link concatenated programs. In this case the linking program must be the first program concatenated. Once concatenated, the new program is loaded into memory and then linked by typing:

RUN

The linking program is then deleted, leaving only the desired appended program. Comparison of these two procedures revealed that the machine language code ran nearly a thousand times faster than the basic code. ©

Multitasking On Your PET?

QUADRA-PET

Charles Brannon
Greensboro, NC

QUADRA-PET is a machine language program that lets you partition the memory of an upgrade ROM PET or CBM into four 8K blocks. Each block is an independent program workspace. Programs existing in each 8K partition can be selected and then used and modified without affecting any of the other programs. You can jump to any other of the programs at any time.

After initialization with SYS 926, PET displays the question:

WHICH PET? [1-4]

Perhaps Mary, an avid computer-games buff, types in "1" and loads STARTREK. She plays it for a while and then leaves to eat lunch. Meanwhile, Bob goes to the PET, sees that someone is using PET #1, and switches to PET #2 to write a business program. After nearly "perfecting" it, he leaves to see what Mary is up to. Now the kids come in, and after arguing for a half-hour agree to share the PET, one using PET #3 and the other PET #4. Fortunately for Bob and Mary, nothing the kids do can harm their programs.

How To Use QUADRA-PET

1. Load or type in one of the versions of QUADRA-

PET. (Basic or hex)
2. Enter **NEW**
3. **SYS 926** to initialize.
4. PET will respond with **WHICH PET? (1-4)**
5. Select the one you wish to use.
6. Before loading or typing in a program for the first time, type in **NEW.**
7. To select another PET, **SYS 826** and follow instructions 4-7.

Now comes the fun part -- how does it work? Many memory locations in zero-page (0-256) are *pointers* QUADRA-PET works with three of those pointers.

On power-up, PET determines the end of memory by writing a character to every memory location and then reading it back. PET then increments a memory location until a failure in reading that character occurs. This indicates that the end of available memory has been reached. Physically, this pointer is at location 52 decimal. ($34). The second pointer is at the start of memory, stored in location 41. Originally, this points to the actual start of user memory, 1024. The last pointer is the end of text pointer. As you write your program it changes.

QUADRA-PET partitions the memory by changing these pointers to point to successively higher memory locations, depending on which PET is in use. Since the end of text pointer changes, it must be saved before we move to a new PET and restored on return. QUADRA-PET, as it is in machine language, does all these things seemingly instantaneously.

HOW TO SAVE A PROGRAM PRODUCED WITH QUADRA-PET:

1. SYS 1024 to go to the Monitor.
2. Enter: .M 0028 002B and type RETURN.
3. You will get a display something like:
   .: 0028 01 04 3E 04 .. ..
4. We will use only the first four bytes. Write down the first pair in reverse order on paper, for example:
   0401
   Do the same with the second pair. (e.g. 043E)
5. Enter: .S "PROG NAME",01,XXXX,YYYY
   where "PROG NAME" is the name of your program, XXXX is the first number you wrote down, and YYYY is the second. For example, to save the example program which we will call "PET #1, you would enter: .S "PET #1", 01,0401,043E
6. Press RETURN and press play and record to save your program.
7. To load *this* saved program into a space prepared by QUADRA-PET, just SYS 1024 and enter .L "PROG NAME" where "PROG NAME" is the name of your program.

A little imagination will create many uses for QUADRA-PET.

For education, it is the perfect way to keep four students' programs in the PET at the same time. Each program can be worked on and modified in any way without affecting any other of the programs.

In business, four different business programs can exist simultaneously in PET's memory, ready to use. For the small penalty of loading the programs into the program workspaces at the start of the day, all four are within reach of a carriage return -- faster than any disk drive.

HOW TO LOAD A PRE-EXISTING PROGRAM INTO A SPACE PREPARED BY QUADRA-PET:

I could tell you how to do this on the old ROM PET but quite frankly, I can't find the memory locations for this procedure in the new PET. All you PET experts -- **HELP!**
If you can figure it out, please send in the procedure to **COMPUTE.**

Machine language programmers can fill partitions with useful routines, leaving one or more partitions for BASIC. QUADRA-PET itself is short and easily relocatable.

I would be interested to find out what novel and useful applications for QUADRA-PET *you* can think up!
Happy QUADRA-PETing!

References

CBM User Manual 2001-32, First Edition. Commodore Business Machines, Inc., Palo Alto, CA (1979)

Havery B. Herman, "Memory Partition of BASIC Workspace", **COMPUTE,** pp. 18-20 (Jan., Feb. 1980)

Jim Butterfield, "PET in Transition (memory map) **COMPUTE,** pp. 68-70 (Fall, 1979)

```
0 REM***********************************
1 REM           QUADRA PET
2 REM***********************************
3 REM:BY CHARLES BRANNON 06/07/80
10 FOR I =  826 TO 941
20 READ A
30 POKE I, A
40 NEXT
50 SYS926
60 END
1000 DATA174,126,3,165,42,157,131,3,165
1010 DATA43,157,135,3,169,143,160,3,32
1020 DATA28,202,32,228,255,41,15,240,249
1030 DATA201,5,176,245,170,202,142,126,3
1040 DATA169,1,133,40,189,127,3,133,41
1050 DATA189,131,3,133,42,189,135,3,133
1060 DATA43,169,0,133,52,189,139,3,133
1070 DATA53,32,119,197,96,0,4,32,64
1080 DATA96,3,3,3,3,4,2,64,96
1090 DATA32,64,96,128,87,72,73,67,72
1100 DATA32,80,69,84,63,32,40,49,45
1110 DATA52,41,0,169,0,141,0,32,141
1120 DATA0,64,141,0,96,76,58,3
READY.
```

PET/CBM Disk Formats

Robert W. Baker
Atco, NJ

Commodore is now including a new manual with all 5¼" floppy disk systems that are currently being shipped. If you have an older disk, whether or not you've upgraded the disk ROMs, you might be interested in trying to get a copy of the new manual listed as Commodore part number 320899. The manual covers the 2040, 3040, 4040, and 8050 drives in great detail. It includes general information on each, along with instructions for using the various programs included on the TEST/DEMO diskette supplied with the drive. The manual also discusses DOS support and the various DOS commands (both new and old). For more advanced programmers, there are sections on using random access files and the special disk utility commands. There's also a chart showing the corresponding disk commands for BASIC 3.0, the DOS Wedge, and for BASIC 4.0. This also provides a handy, quick reference that you might want to make a copy of and keep close by.

Of particular interest, are several tables and charts near the back of the book that describe the format of the disk header, BAM, directory, and disk files. They're provided for all three known formats of the Commodore disks. I thought it might be helpful to present some of the information here in a somewhat condensed form.

Remember that you can read the BAM, header, and directory blocks as a single sequential data file opened with a filename of $0 or $1. To read the data contained in these blocks you must use the GET# command since there are no carriage return delimiters. Also, you will only see 254 bytes per block, since the first two bytes used for the track/sector links are not sent to the PET/CBM when reading a data file.

If you look at the header formats, you'll see that the first byte received will always be an unique value for each disk format type. It identifies the disk format that the disk was originally created with, not the current DOS revision being used. This can be used as a program flag telling how the remainder of the header and directory should be read.

If you open a program file as a data file, the first two bytes read will indicate the load address for the program in standard 6502 address format (low byte, high byte). The remainder of the file will then be the tokenized BASIC program just as normally stored in memory.

2040/3040 BAM & HEADER
Track 18, Sector 0

Byte	Contents	Definition
0-1	18,1	Track & sector of first directory block.
2	1	Indicates DOS 1 format
3	0	Null flag for future use
4-143	...	BAM - bit map of available blocks for tracks 1-35
144-161	...	Disk name padded with shifted spaces
162-163	...	Disk ID
164-170	160	Shifted spaces
171-255	0	Nulls, not used

4040 BAM & Header
Track 18, Sector 0

Byte	Contents	Definition
0-1	18,1	Track & sector of first directory block
2	65	ASCII "A" for 4040 format
3	0	Null flag for future use
4-143	...	BAM - bit map of available blocks for tracks 1-35
144-161	...	Disk name padded with shifted spaces
162-163	...	Disk ID
164	160	Shifted space
165-166	50,65	ASCII "2A" for DOS version & format type
166-170	160	Shifted spaces
171-255	0	Nulls, not used

Note: Some 2040, 3040, 4040 disks may have ASCII characters in locations 180-191.

8050 Directory Header
Track 39, Sector 0

Byte	Contents	Definition
0-1	38,0	Track & sector of first BAM block
2	67	ASCII "C" for 8050 format
3	0	null flag for future use
4-5	0	unused
6-21	...	Disk name padded with shifted spaces
22-23	160	Shifted spaces
24-25	...	Disk ID
26	160	Shifted space
27-28	50,67	ASCII "2C" for DOS version and format type
29-32	160	Shifted spaces
33-255	0	Nulls, not used

8050 BAM Format
First BAM Block = Track 38, Sector 0

Byte	Contents	Definition
0,1	38,3	Track & sector of second BAM block
2	67	ASCII "C" for 8050 format
3	0	Null flag for future use
4	1	Lowest track# in this BAM block
5	51	Highest track# + 1 in this BAM block
6	...	Number of unused blocks on track 1
7-10	...	Bit map of available blocks on track 1
11-255	...	BAM for tracks 2-50, 5 bytes each

Second BAM Block = Track 38, Sector 3

Byte	Contents	Definition
0,1	39,1	Track & sector of first directory block
2	67	ASCII "C" for 8050 format
3	0	Null flag for future use
4	51	Lowest track# in this BAM block
5	78	Highest track# + 1 in this BAM block
6	...	Number of unused blocks on track 51
7-10	...	Bit map of available blocks on track 51
11-140	...	BAM for tracks 52-77, 5 bytes each
141-255	...	Unused

Common Directory Format
First block on 2040/3040/4040 = Track 18, Sector 1
First directory block on 8050 = Track 39, Sector 1

Byte	Definition
0-1	Track & sector of next directory block (last block has a track of 0)
2-31	File entry 1
34-63	File entry 2
66-95	File entry 3
98-127	File entry 4
130-159	File entry 5
162-191	File entry 6
194-223	File entry 7
226-255	File entry 8

Structure Of Single Directory Entries

Byte	Definition
0	File type flag OR'ed with $80 Types: 0 = Deleted 1 = Sequential 2 = Program 3 = Relative
1-2	Track & sector of 1st data block
3-18	File name padded with shifted spaces
19-20	Track & sector of first side sector block for relative files only
21	Record size for relative files
22-25	Unused
26-27	Track & sector of replacement file when OPEN@ is in effect
28-29	Number of blocks in file (low byte, high byte)

Standard File Formats

Byte	Definition
0-1	Track & sector of next sequential block. The last block is indicated by a 0 followed by a value that indicates the length of the last partial block.
2-255	Sequential data files— 254 bytes of data with carriage returns as record separators. Program files— 254 bytes of program information stored in PET/CBM memory format. (key words stored as BASIC tokens) End of file is marked by three zero bytes just as in memory. The first two bytes of the first block indicate the load address of the program. Relative files— 254 bytes of data. Empty records contain $FF (all binary ones) in the first byte followed by $00 (binary zeroes) to the end of the record. Partially filled records are padded with nulls ($00).

Structure Of BAM Entried For Any Track

Byte	Definition
0	number of available sectors for track
1	bit map sectors 0-7
2	bit map sectors 8-15
3	bit map sectors 16-23
4	bit map sectors 24-31 (only on 8050)
	For the bit maps, 1 = available 0 = not available

Relative File Side Sector Blocks

Byte	Definition
0-1	Track & sector of next side sector block
2	Side sector number (0-5)
3	Record length
4-5	Track & sector of 1st side sector (number 0)
6-7	Track & sector of 2nd side sector (number 1)
8-9	Track & sector of 3rd side sector (number 2)
10-11	Track & sector of 4th side sector (number 3)
12-13	Track & sector of 5th side sector (number 4)
14-15	Track & sector of 6th side sector (number 5)
16-255	Track & sector pointers to 120 data blocks

Interfacing With The User Of Your Pet Programs

Don Cassel

As programmers of microcomputers we seldom write programs that are intended solely for our own use. Rather the programs we develop are directed to a student audience, to business users or even for publication or sale to the general microcomputer public. It is therefore crucial that we develop the program in such a way that it interacts with the user at a suitable intellectual level and with an effectiveness that permits ease and simplicity of operation.

The intellectual level considers both the language and technical ability of the program's user and employs prompting and dialogue that is consistent with this level. Effectiveness of operation complements the intellectual by considering the methods which are used to communicate with the user. These methods may include queries, prompting, codes, menus, form filling and so on. Suitable choices by the program designer often determines a person's success in using the program and a willingness to use it again.

User Level

When we develop a program it is usually intended for a specific audience. In general, we should assess the age, education, training, intelligence and motivation of the user. After a general evaluation of our audience it is useful to place the user in one of the following three categories.

1. Casual
This is a user who uses a micro infrequently and generally has no training in computers. All first time users fall into this category. So do many students and educators.

2. Trained
A trained user is a person who has been given formal (sometimes informal) training on the use of computers. This training might be limited to the use of a particular program or be as broad as a computer literacy course.

3. Programming Skills
This is the most sophisticated user. He or she will have done some programming and be familiar with programming terminology and language syntax.

In addition to these three categories a user may operate the program on two different levels. These are defined as an active or passive operator. An active operator is one who initiates program action by entering commands. This level is typical of games where the user enters values that control the direction and speed of a ball on the screen.

A passive operator is one who takes action based on the program's initiative. For instance, when a program asks for the user's name. Many programs will use both active and passive interaction.

User Dialogues

Prompting

In a sense any information that requires a user response represents prompting. However, our interest here is when the program displays a statement or question and then waits for the user to type a response. Prompting is appropriate for all levels of users but the language of the prompt should be directed to the specific user level. In any case courtesy should be used. A prompt such as

```
PLEASE ENTER YOUR NAME?
```

is infinitely better than a curt

```
NAME?
```

This kind of prompt is usually implemented as follows:

```
100 PRINT "PLEASE ENTER YOUR NAME";
110 INPUT N$
```

or more directly and efficiently as

```
100 INPUT "PLEASE ENTER YOUR NAME";N$
```

With this type of prompt you may get a variety of responses, such as:

```
JOHN
JOHN SMITH
SMITH
```

In some programs this doesn't matter but if it does make a difference the prompt should be more specific.

```
PLEASE ENTER YOUR SURNAME?
```

Some prompts may have simple alternatives such as YES or NO, TRUE or FALSE, ADD or SUBTRACT. In these situations the prompt should indicate which responses are expected.

```
100 INPUT "DO YOU WANT MULTIPLE CHOICE
  (YES/NO)";A$
110 IF A$="YES" THEN 200
```

This prompt indicates clearly that a YES or NO answer is expected. Some programs also permit the user to respond with just the first letter of the response. This is done by extracting the first letter of A$ using the LEFT$ function and then testing for a "Y" or "N". Using this method single letter responses as well as the complete word are acceptable.

```
100 INPUT "WOULD YOU LIKE MULTIPLE
    CHOICE
  (YES/NO)";A$
110 IF LEFT$(A$,1)="Y" THEN 200
```

Some prompts use data from previous operations in the program. An example of this is a program that generates drill and practice questions for addition. Prior to the prompt the program generates two values (A and B) which the student then adds mentally or on paper before entering the answer. This prompt might then use both the print and input statements.

```
100 PRINT "[clr dn dn dn dn dn]";
110 PRINT "WHAT IS THE SUM OF";A;"+"B;
120 INPUT S
```

Note the use of line 100 which clears the screen and moves the cursor down 5 lines before printing the prompt. This action avoids any distraction from previous questions that would otherwise remain on the screen.

A more creative solution to this problem might be to print the prompt in the form of a traditional addition question as follows:

```
100 PRINT "[clr dn dn dn dn dn]";
110 PRINT "[rt rt rt rt]";A
120 PRINT "[rt rt rt]+";B
130 PRINT "[rt rt rt]_____"        (_ is a shift@)
140 INPUT "[rt rt rt]";S
```

If A is the value 10 and B is 15 this code displays the following:

```
  10
  15
 ----
  ?
```

Although this solution is a lot more work the results are far superior to previous methods and uses the capability of the PET in implementing an effective solution.

Default Responses

In many applications a choice is given to the user but the response can often be anticipated. A program which normally reads a file but has an option to create a file is an example of when a default would be useful. The prompt might be:

(READ) OR (CREATE) A FILE?

Here the user must type in either READ or CREATE as a response. Since we expect that READ is the most frequent response it may become the default. This is done by printing the default value with the prompt and then moving the cursor to the left past the default value.

```
100 INPUT "(READ) OR (CREATE) A FILE
  ↑ READ[lt lt lt lt lt lt]";A$
  |                  ↑
  |                  |
2 blanks         6 cursor lefts
```

This is what the display shows:

```
(READ) OR (CREATE) A FILE? READ
                          ↑↑
             _____________||
            |              |
Question mark created as a result     cursor position
of A$ in the INPUT statement.
```

The six cursor lefts are needed to move past the four letters of READ and the two spaces. The question mark appears in the first space and then the cursor automatically moves one position to the right leaving the second space for readability. Now if the user wants to use the READ default only the return key is pressed. If not CREATE may be typed to select the alternative.

A complete input module might be the following:

```
100 INPUT "(READ) OR (CREATE) A FILE
      READ[lt lt lt lt lt lt]";A$
110 IF A$="READ" THEN 200
120 IF A$="CREATE" THEN 400
130 PRINT "[clr]ENTER EITHER READ OR
      CREATE"
140 GOTO 100
```

Menus

A menu provides a list of alternatives from which the user selects one entry by typing a number or letter. The following code gives an example of this approach:

```
100 PRINT "[clr]SELECT ONE OF THESE
      SEARCHES"
110 PRINT "1 – BY AUTHOR"
120 PRINT "1 – BY SUBJECT"
130 PRINT "3 – BY KEYWORD"
140 PRINT "4 – BY TITLE"
150 INPUT A
160 ON A GOTO 1000,2000,3000,4000
170 GOTO 100
```

The codes 1,2,3,4 translate nicely for use in either an ON — GOTO or ON — GOSUB which greatly simplifies program analysis of the user's response.

A similar approach uses the first letter of each menu item as the response key. The letter may be highlighted by turning the reverse on before the character and off after it.

```
100 PRINT "[clr]SELECT ONE OF THESE
      SEARCHES"
110 PRINT "[rvs]A[rvs off]UTHOR"
120 PRINT "[rvs]S[rvs off]UBJECT"
130 PRINT "[rvs]K[rvs off]EYWORD"
140 PRINT "[rvs]T[rvs off]ITLE"
150 INPUT A$
160 FOR I=1 TO 4
170 IF A$=MID$("ASKT",I,1) THEN ON I GOSUB
      1000,2000,3000,4000: I=4
180 NEXT I
190 GOTO 100
```

This approach tends to cause the reversed characters to run together on the screen. A simple solution is to separate each line by placing a PRINT at lines 105,125,135 and 145 as follows:

```
105 PRINT
```

Lines 160 to 180 examine the input character against the string "ASKT" in line 170. The response character will be equal to position I of the string. In other words letter S will be found when I is 2 and the GOSUB will transfer to subroutine 2000. The I=4 at the end of line 170 forces the end of the FOR loop without the need for additional looping.

Most menus require only a single letter or number response as seen in the previous examples. When the INPUT statement is used two keystrokes are required: one for the character and one for the Return key. One advantage this has is the user may change the character anytime before return is pressed by moving the cursor to the left and retyping the character.

In other cases a quick response is preferred with a minimum of keying. Here the GET statement comes to our rescue. If the previous example replaces line 150 with

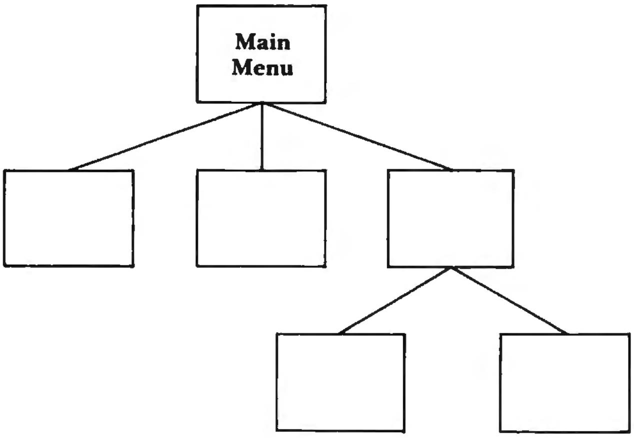
```
150 GET A$:IF A$="" THEN 150
```

the typing of a single letter will be sufficient. This method works only for a single character response but is unequalled for speed.

Multi-Level Menus

When we are using a screen as the program's medium for communication with the user there is a limit to the number of items a menu can contain. Usually it is preferable to limit menu items to less than ten if numeric responses are required. With a blank line between each item this requires at least 19 lines on the screen. However, many applications may require well in excess of ten items. The procedure for handling many entries is to create a multi-level menu.

The multi-level menu begins with a main menu that branches to several lower level menus. Lower level menus may themselves point to even lower levels in a complex application.



These lower level menus should provide a means of escape in the form of a menu item that points back to the main menu.

An example of this application is a budgeting program with a main menu that asks for annual, monthly or daily budget items. The program then branches to a lower level menu to itemize particular entries in one of these three categories.

```
100 PRINT "[clr]ENTER A BUDGET CATEGORY"
110 PRINT "[dn rvs]A[rvs off]NNUAL"
120 PRINT "[dn rvs]M[rvs off]ONTHLY"
130 PRINT "[dn rvs]D[rvs off]AILY"
140 GET A$:IF A$="" THEN 140
150 FOR I=1 TO 3
160 IF A$=MID$("AMD",I,1) THEN ON I GOSUB
    200,400,600
170 GOTO 100
200 REM ANNUAL BUDGET ITEMS
210 PRINT "[clr]SELECT AN ANNUAL ITEM"
220 PRINT "[dn]1 - HOUSE INSURANCE"
230 PRINT "[dn]2 - CAR INSURANCE"
240 PRINT "[dn]3 - INCOME TAX"
250 PRINT "[dn]4 - PROPERTY TAX"
260 PRINT "[dn]5 - RETURN TO MAIN MENU"
270 GET A$:IF A$="" THEN 270
280 N=VAL(A$)
290 IF N=5 THEN RETURN
300 ON N GOSUB 1000,2000,3000,4000
310 GOTO 200
400 REM MONTHLY BUDGET ITEMS
      :
    etc.
```

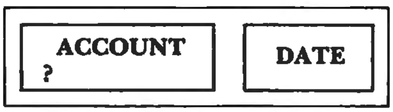
Form Filling

Some application areas such as accounting, CAI and testing can benefit from a technique called form filling. This method permits the program's user to input data in a predefined location on the line. Typically this location is inside a box, as in an accounting ledger, or in a blank area such as a fill in the blank type of test question. The following code shows the use of form filling to enter an account number and date in a precise location within a box.

In the following code the lower case "b" represents a blank character.

```
10 PRINT "[clr rvs]bbACCountbbbbbbDATEbbbbb
   [rvs off]"
20 PRINT "[rvs]b[rvs off]bbbbbbbbb[rvs]b[rvs off]bbb
   bbbbbbbbb[rvs]b[rvs off]"
30 PRINT "[rvs]24 space[rvs off]"
40 INPUT "[home dn rt]";A$
50 INPUT "[home dn 11 rt's]";D$
60 A$=LEFT$(A$,6)
70 D$=LEFT$(D$,9) sk 2
```

The above code displays a form on the screen something like this:



When the program asks for input the question mark appears in the box under the appropriate heading and is followed by the flashing cursor. This type of program presents a unique problem since the INPUT statement accepts all of the characters on the line following the question mark. This also includes graphic characters.

One solution is to use the GET statement in a loop but a far simpler method is to use the LEFT$ function which selects the number of characters desired from the input string. This of course means you need to know how many characters will be entered.

Command Language

Command languages are useful for applications such as word processing where simple prompting or the use of a menu is either impractical or too unwieldy to enter complex commands. For instance, a typical command is to change a string from one value to another to correct a spelling error or to change a word. To change the word "error" in the previous sentence to "mistake" a command like

C/error/mistake

is entered.

Using a command language requires considerably more experience than simply responding to a menu, but it is much faster than using a dialogue. Programming for a command language also tends to be more complex since the program needs to recognize the type of command, often identified by the first character in the command, and the operands which can legitimately accompany that command. Often there is no prompt since this type of application requires an active operator who initiates all action.

The following code shows how the preceding command might be analyzed in BASIC.

```
100 INPUT C$
110 IF LEFT$(C$,1)="C" THEN 500
 .
 .
500 REM DECODE CHANGE COMMAND
510 IF MID$(C$,2,1)<>"/" THEN PRINT "COMMAND
    ERROR": GOTO 100
520 L=LEN(C$)
530 FOR I=3 TO L
540 IF MID$(C$,I,1)="/" THEN 570
550 S1$=S1$+MID$(C$,L,1)
560 GOTO 580
570 S2$=RIGHT$(C$,L-I):I=L
580 NEXT I
590 IF LEN(S2$)=0 THEN PRINT "COMMAND
    ERROR": GOTO 100
```

Statements 510 and 590 ensure the command format is followed by checking for a slash separating the command (C) from the first string and that there is a second string. Statements 530 to 560 extract the first string by concatenating each character to S1$. When the end of the string is found by 540 the second string is extracted in 570 and stored in S2$.

The discussion has certainly not been an exhaustive study of user interaction but hopefully some insights have been given into ways of communicating effectively with the users of your programs.

Conventions

For ease of reading and interpreting the BASIC language examples in this report the following conventions have been used:

Code	Interpretation
b	blank or space
clr	clear screen
home	home cursor
dn	cursor down
lt	cursor left
rt	cursor right
up	cursor up
rvs	reverse
rvs off	reverse off

Bibliography

Drake, Bob. *Menu Interpretation Made Clearer.* ECOO Newsletter Volume 1, Number 4, October 1980, Page 49.

Martin, James. *Design of Man-Computer Dialogues.* Prentice-Hall Inc., Englewood Cliffs NJ, 1973

Nelson, Ted. *Interactive Systems and the Design of Virtuality.* Creative Computing, Nov. 1980, Page 56.

Poynor, Hugh. *Improving Your Console Input.* Interface Age, October 1980, Page 100.

Keeping TABs On The Printer

Jim Butterfield
Toronto, Canada

It seems to work, but it doesn't. This vexes a lot of people: after all, TAB seems ready made for the printer, and quick tests appear to show that it works. But it won't work when you get down to cases — with one exception we'll mention later.

It Can't Work

To understand the difficulty, we must first understand that a Basic TAB does not send a special "tab" signal out to the printer. Instead, it sends out what it thinks is the proper number of space characters. If you code PRINT#4, TAB(15) the computer will probably transmit 14 spaces to the printer.

The mechanisms to make TAB work properly to the printer are not in place. Consider this: that if TAB were built to work with the printer, it must also be able to work properly with two printers at the same time. A user can hook up two printers as devices 4 and 5, for example, and print invoices on number four and summaries on number five. If printer TAB were in there, the computer would need to keep separate records of where each printer was positioned on its own line. In principle, we could run ten printers at the same time, since we can have up to ten logical files; the computer would need to keep ten independent records of each printer's line position.

No such information is available in the PET/CBM. Thus, there's no way to calculate how many space characters would bring it to the desired column.

It Seems To Work

PET/CBM Basic included TAB strictly as a screen function. You can TAB quite effectively on a screen display. The mechanism is this: the computer subtracts the cursor position from the desired print position, and sends the appropriate number of spaces (or cursor-rights in the case of the screen). For example, if the cursor is at position 7 and the program gives TAB(16), nine spaces will be delivered. If we're printing to the screen, that produces exactly the right result; we'll end up at column 16 where we can print the next item.

However, if we're sending this information to the printer, we may produce a wrong result. Suppose, for example, we code PRINT#4, "THERE";TAB(10);"IT";TAB(20);"IS". The five characters of THERE go to the printer, leaving the printer positioned at column six. At this point we would hope that the computer would subtract six from the TAB value of ten and send four spaces; instead, it subtracts one (the screen cursor position) from ten and sends nine spaces. It prints in column 15 rather than the desired column 10. It gets worse, since we hoped that the computer would send 8 spaces to carry us to TAB(20); instead, it will send 19, and IS ends up starting in column 36.

The misleading part of all this is that tabbing seems to be taking place; spaces are certainly inserted in the print line. The columns don't seem to line up quite right, and the programmer/victim goes looking through his coding trying to find his mistake. It's not a coding mistake: the system can't work. The only mistake is trying to use TAB with the printer at all.

Finding Another Way

The healthiest way around the problem is to calculate your own print line spacings. Keep track of where the printer is working by counting characters with the LEN function, or making sure every printed item is always the same length. Here's a quick program using LEN:

```
100 OPEN 4,4
110 DATA FIVE,SIX,SEVEN,EIGHT,NINE,TEN,
    ELEVEN
120 FOR J=5 TO 11
130 READ A$
140 B$=STR$(J)
150 PRINT#4,A$;LEFT$("          ",10-LEN(A$));
160 PRINT#4,B$;LEFT$("     ",5-LEN)B$));
170 PRINT#4,"X"
180 NEXT J
190 CLOSE 4
```

Note how we change the numeric into a string (B$) so that we can measure its length. Here's another approach which forces all strings to the same length:

```
100 OPEN 4,4
110 DATA FIVE,SIX,SEVEN,EIGHT,NINE,TEN,
    ELEVEN
120 FOR J=5 TO 11
130 READ A$
140 B$=STR$(J)
150 PRINT#4,LEFT$(A$+"          ",10);
160 PRINT#4,LEFT$(B$+"     ,5);
170 PRINT#4,"X"
180 NEXT J
190 CLOSE 4
```

Can you see how to change lines 150 and 160 to right-justify the columns if desired? For example, line 160 becomes PRINT#4,RIGHT$("   "+B$, 5); and the numbers slide over to the right.

Since we're arranging our own columns, this type of program works equally well to printer or screen. Try changing line 100 to OPEN 4,3 in either program and the output will go neatly to the screen display.

Making It Work

You can force TAB to work with the printer by using a trick. It's probably too cute for serious

NEW - LOW COST FLOPPY DISK FOR PET*!

PEDISK II

WHY PEDISK?

FAST – The *Pedisk* system loads directly to main memory from the disk and loads – saves information at least 3 times faster than any other disk available.

SIMPLE – A simple command syntax makes its BASIC commands easy to use. At the same time, its sophisticated indexed, sequential, or relative file handling makes it powerful.

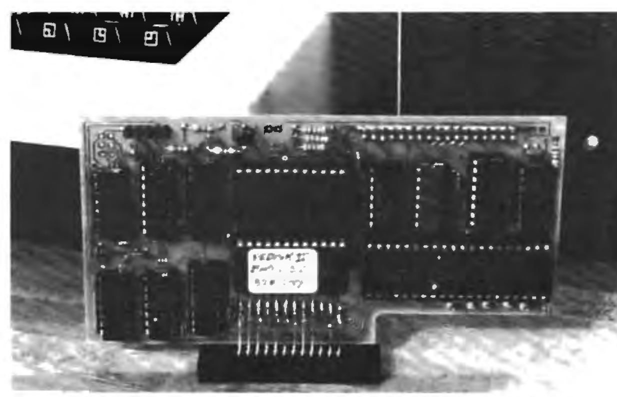
RELIABLE – The ultra-simple circuit design with its LSI disk controller chip provides maximum timing margins for error free operation.

5¼" system – 1 drive, double density (143K)	$595.00	2 drives, double density (286K)	$945.00
5¼" system – 1 drive, quad density (286K)	$795.00	2 drives, quad density (572K)	$1,195.00
8" system – 1 drive, IBM 3740 business (295K)	$1,295.00	2 drives, IBM 3740 business (590K)	$1,895.00

Introducing **PEDISK II**, a low cost high performance floppy disk system consisting of: 1) *PEDISK II* controller, 2) flat disk cable, 3) Disk Drive Assembly, and 4) PDOS II software. The *PEDISK II* is a full function peripheral that provides high speed program storage and a sophisticated file handling package.

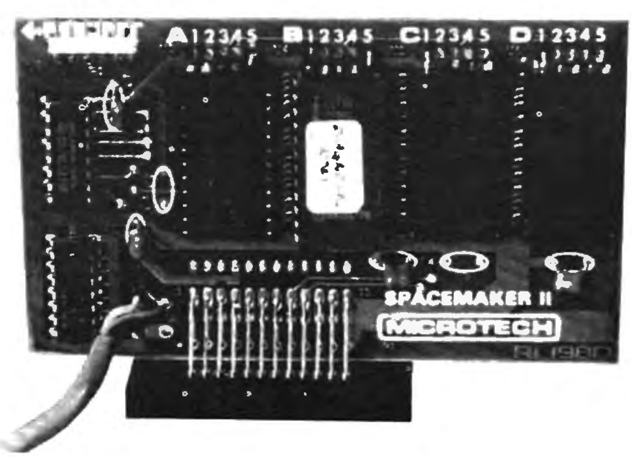
The small *PEDISK II* controller board mounts inside any 2000, 4000, or 8000 series machine. Standard systems are available with one or two disk drives. An optional third drive can also be added. The *PEDISK II* System can be operated simultaneously with any Commodore disk system for data exchange purposes.

The PDOS II software links BASIC by adding a new repertoire of disk commands. !OPEN, !CLOSE, !INPUT, and !PRINT provide the basis of the powerful file handling package. !LOAD, !SAVE and !RUN allow complete disk control. PDOS II also offers a full DOS-mode of operation for all disk diagnostic and utility functions. Diskette format, backup, diagnostic, and reorganization capabilities are provided.



PEDISK II CONTROLLER BOARD

NEW - SPACEMAKER II switch between one of FOUR ROMS



SPACEMAKER II is the new ROM switch from Microtech. It allows either manual or software controlled switching of up to four ROMs in a single ROM expansion socket. The switching is accomplished with a side-mounted slide switch or via ROMDRIVER, an accessory board which allows software controlled switching and keyboard controlled switching. ROM I/O is a special software package available on disk to implement full keyboard control of the ROMs. In addition, it adjusts for memory differences found in various utility ROMs.

SPACEMAKER II	$39.00
ROMDRIVER	$39.00
ROM I/O Commodore or Pedisk	$ 9.95

SEE YOUR LOCAL DEALER FOR ALL cgrs MICROTECH PRODUCTS

cgrs MICROTECH P.O. BOX 102, LANGHORNE, PA. 19047

*PET IS A TRADEMARK OF COMMODORE

coding, but it will perform correctly.

Here's the method: force the screen cursor to match the printer column exactly. Then a TAB, which references the cursor, will generate the proper number of spaces since the cursor and the printer are at the same column. The coding needs to be carefully worked out; the screen must print exactly the same as the printer, and output must always go to the printer first. Additionally, TAB references must be arranged so that they always begin a PRINT statement.

An example will show how it's done:

```
100 OPEN 4,4
110 DATA FIVE,SIX,SEVEN,EIGHT,NINE,TEN,
    ELEVEN
120 FOR J=5 TO 11
130 READ A$
140 REM
150 PRINT#4,A$;
155 PRINT A$;
160 PRINT#4,TAB(10),J;
165 PRINT TAB(10),J;
170 PRINT#4,TAB(15),"X"
175 PRINT TAB(15),"X"
180 NEXT J
190 CLOSE 4
```

It's not recommended as a general programming approach. But seeing this program work will give you an insight into how TAB functions — and malfunctions. ©

Assembler in BASIC for the PET

Eric Brandon

The most important tool the machine language programmer has is his assembler. If he has 32K, a disk, new ROMs, and $170, he can buy MAE or the Commodore assembler. Otherwise, his choice is limited. If the thing he lacks the most is the $170, or if he just wants to dabble in machine language, he will often end up with Newmon, Supermon, or a similar extended monitor that has a simple assembler. This is fine, but these assemblers are not symbolic, and he must calculate branches, jumps, and the like every time he modifies or relocates it.

Since I have an 8K PET with old ROMs, the only option open to me was to write my own symbolic assembler. I had to decide whether to write it in machine language or BASIC. Since I could not face the bleak prospect of writing a parser in machine language, I wrote it in BASIC. An added bonus is that it works on all ROMs (I have tested it on 2.0 and 3.0), and can easily be used by readers of **COMPUTE!** with other machines, since they all use the 6502.

The first thing you must do after typing it in, is to change line 1 to suit your memory size. MEM is the number of lines of machine code it can hold. When MEM is large, not only does it eat up memory, but it slows down the assembly process as well. M2 is the number of symbols it can keep track of. For an 8K PET, I suggest MEM=40 and M2=20; larger values are likely to run out of memory. For 16K or 32K PETs, MEM=200 and M2=100, unless you know you are going to use more lines or symbols.

When you RUN it, you will see the number 1, and a white cursor beside it. This means you are in line 1, and that it is waiting for input into the label field. If you type to the end of the field, hit SPACE, or hit RETURN, you will jump to the next field. The length of the label field is 6 characters, the operation field is 3, and the operand field is 10 characters long. A SPACE or RETURN in the operand field will put you at the beginning of the next line. When in the label field, there are two special commands you can type in. You type "FIX" when you have made a mistake. FIX returns you to the previous line so that if you type FIX on line 20, a 19 with a cursor beside it will appear underneath.

Typing "EXIT" will bring you to a menu. Type "I" to input some more code. You will be asked at what line you wish to begin inputting. Type "D" to delete. You will be asked for a starting and an endinng line number. The starting line and everything up to, but not including, the ending line will be deleted. To delete just one line, type its number as both starting and ending lines. Type "N" to insert. You will be asked what line to begin inserting at and how many lines to insert. All lines including and below the one you specified will be moved down the number of lines you said, leaving a gap of blank lines behind. Type "L" for list; you supply the beginning and ending line numbers. Type "A" to assemble your program. You must specify whether you want the output to go to the screen printer. Note that if you choose the screen, there will often be lines of more than 40 characters since the output was designed for the printer. If you plan to use the screen exclusively, I suggest you modify lines 1180–1210 to make the output less than 40 characters long. Note that your program has been POKEd into memory and may be executed with an SYS after assembly. Type "Q" to quit the program. If you hit Q accidentaly, a GOTO 300 will return you to the program with your work intact.

I have included a sample printout which you should consult as I explain the operation of the assembler. As you can see, a symbol table is generated before the actual code. An "=" as the operation will set the symbol on the left equal to the value of the expression on the right. The first line should be an asterisk equal to a value. This sets the origin of the program in memory. The origin may be set only once, and only in the first line; any other attempt will give unpredictable results. Immediate addressing must be indicated with a pound sign as in lines 8 and 16 of my sample program. Hexadecimal numbers must be preceded by a dollar sign, and must be 1 to 4 characters in length. Binary numbers must be preceded by a percentage sign, and may be of any length. Decimal is assumed by default. A symbol must begin with a letter, and contain only letters and numbers. In accumulator addressing, the operand must be the letter "A", therefore "A" is an illegal symbol, although symbols may contain and begin with the letter "A". Addition within the operand field is non-standard. Only

```
CHAR =      $03E0
SCRN =      $0020
AGAIN =     $0342
LOOP =      $0349
 1                        *       =    $33A
 2                        CHAR    =    $3E0
 3                        SCRN    =    $20
 4    033A A9 00                  LDA  #0
 5    033C A8                     TAY
 6    033D 8D E0 03               STA  CHAR
 7    0340 85 20                  STA  SCRN
 8    0342 A9 80          AGAIN   LDA  #$80
 9    0344 85 21                  STA  SCRN+
 10   0346 AD E0 03               LDA  CHAR
 11   0349 91 20          LOOP    STA  (SCRN),Y
 12   034B C8                     INY
 13   034C D0 FB                  BNE  LOOP
 14   034E E6 21                  INC  SCRN+
 15   0350 A6 21                  LDX  SCRN+
 16   0352 E0 84                  CPX  #$84
 17   0354 D0 F3                  BNE  LOOP
 18   0356 EE E0 03               INC  CHAR
 19   0359 D0 E7                  BNE  AGAIN
 20   035B 60                     RTS
```

symbols can be added to and addition is done by following the symbol with plus signs as in lines 9, 14, and 15 of the sample program. The number of plus signs is equal to the number to be added. If BUFFER = 30, LDA (BUFFER++,X) means LDA (32,X). I suggest that when writing self-modifying code, you put the code to be modified physically before the code that modifies it. Otherwise, you are almost sure to get an error message. By the way, my sample program fills the screen with every possible character, and is an excellent demonstration of the speed of machine language. It is written for the old ROM PET, but will work on the new ROMs if line 3 is changed to SCRN = B5.

Here is a quick summary of the part of my program for those who may wish to modify it:

- **Lines 100–200** : control the input, and use the input routine at line 4000.
- **Lines 300–600** : execute the command options other than ASSEMBLE.
- **Lines 660–770** : create the symbol table.
- **Lines 790–1220** : assemble the code.
- **Lines 4000–4160** : input routine
- **Lines 5000–5510** : are the op-code tables
- **Lines 6000–6100** : find the numerical value of the operand.
- **Lines 7000–7040** : Convert hexadecimal numbers to decimal.
- **Lines 8000–8020** : Convert binary numbers to decimal.
- **Lines 9000–9020** : Convert decimal numbers to hexadecimal.
- **Lines 1000–10100** : Separate the labels, operations, and operands from the packed array A$.

The program is quite compact because it had to be compressed to fit in an 8K PET. As it is, 8K can only hold about 40 lines of machine code along with the program before running out of memory.

Improvements that could be made are: compacting the code even more, putting in READ/WRITE to cassette or disk routines, putting a BYT pseudo-op, and much more. In the meantime, it is a better assembler than the non-symbolic ones, and hopefully will be of use to PET owners; especially those with old ROMs who have woefully few assemblers available for them. If you find any bugs, make any improvements or have any questions about my program, please write me at:

Eric Brandon
36 Hartfield Road
Islington, Ontario
Canada
M9A 3C9

If you don't want to type it in, send me a blank cassette and $5, and I'll make you a copy.

ASSEMBLER/EDITOR 1.5 – ERIC BRANDON

```
1 MEM=50:M2=20
5 PRINT"■"
10 DIMA$(MEM),S$(M2),V(M2),LI(3)
15 H$="0123456789ABCDEF"
100 LN=1
110 PRINTLN;:TB=5:LT=6:GOSUB4000:IFIN$="EXIT"THEN300
120 IFIN$="FIX"THENLN=LN-1:PRINTCHR$(-13*(ASC(GT$)<>13));:GOTO110
125 IF GT$=CHR$(13)THENPRINT"■";
130 A$(LN)=IN$+" ":TB=13:LT=3:GOSUB4000:A$(LN)=A$(LN)+IN$+" "
160 IF GT$=CHR$(13)THEN200
170 TB=18:LT=10:GOSUB4000:A$(LN)=A$(LN)+IN$
190 IF GT$<>CHR$(13)THENPRINT
200 LN=LN+1:GOTO110
300 PRINT"■■I■NPUT ■D■ELETE I■N■SERT ■L■IST ■A■SSEMBLE ■Q■UIT"
310 PRINT"COMMAND ?";
320 GETCM$:IFCM$=""THEN320
325 PRINTCM$:IFCM$<>"I"THEN410
340 INPUT"LINE ";LN:IFLN=0THEN300
350 GOTO110
410 IF CM$<>"D"THEN460
420 INPUT"■LINES - FROM,TO ";FL,LL:IFFL<>LLTHEN430
422 FORT=FLTOMEM-1:A$(T)=A$(T+1):NEXTT:GOTO300
430 FORT=LL TO MEM:A$(T-LL+FL)=A$(T):A$(T)="":NEXTT:GOTO300
460 IFCM$<>"N"THEN500
470 INPUT"FIRST LINE,NUMBER";FL,LL:FORT=MEM-LLTOFLSTEP-1:A$(T+LL)=A$(T):NEXTT
490 FORT=FLTOFL+LL-1:A$(T)="":NEXTT:GOTO300
500 IF CM$<>"L"THEN580
510 INPUT"LINES FIRST,LAST";FL,LL:FORT=FLTOLL
521 IF LEN(A$(T))=0THENPRINTT:GOTO565
525 LI(1)=0:LI(2)=0:LI(3)=0:LI=0:FORQ=1TOLEN(A$(T))
540 IF MID$(A$(T),Q,1)=" "THENLI=LI+1:LI(LI)=Q
545 NEXTQ:IFLI(3)=0THENLI(3)=Q-1
550 PRINTTTAB(5)LEFT$(A$(T),LI(1))TAB(13)MID$(A$(T),LI(1)+1,LI(2)-LI(1));
560 PRINTTAB(18)RIGHT$(A$(T),LI(3)-LI(2)+1)
565 NEXTT:GOTO300
580 IFCM$<>"Q"THEN600
590 PRINT"■GET BACK IN WITH ■GOTO 300■":END
600 IFCM$<>"A"THEN300
605 PRINT"■■S■CREEN OR ■P■RINTER ?";
610 GETDV$:IFDV$=""THEN610
620 PRINTDV$:IFDV$="S"THENDV=3:GOTO650
640 DV=4
650 CLOSE1:OPEN1,DV:SB=1
660 FORT=1TOMEM:GOSUB10000:IFLB$=""THEN710
670 IF OC$<>"="THEN700
680 GOSUB6000:IFLB$="*"THENPC=NU:OG=NU:GOTO770
690 S$(SB)=LB$:V(SB)=NU:SB=SB+1
692 N=V(SB-1):GOSUB9000
695 PRINT#1,S$(SB-1)" ="LEFT$("          ",8-LEN(S$(SB-1)))"$"R$:GOTO770
700 S$(SB)=LB$:V(SB)=PC:SB=SB+1
702 N=V(SB-1):GOSUB9000
705 PRINT#1,S$(SB-1)" ="LEFT$("          ",8-LEN(S$(SB-1)))"$"R$
```

```
710 IFOC$=""THEN770
715 IFOP$=""THENPC=PC+1:GOTO770
717 IFOP$="A"THENPC=PC+1:GOTO770
720 IFLEFT$(OC$,1)<>"B"OROC$="BIT"OROC$="BRK"THEN740
730 PC=PC+2:GOTO770
740 IFLEFT$(OC$,1)="J"THENPC=PC+3:GOTO770
750 GOSUB6000:IFNU<256THENPC=PC+2:GOTO770
760 PC=PC+3
770 NEXTT
790 PC=OG:ER=0
800 FORT=1TOMEM:GOSUB10000:IFOC$=""THEN1220
805 IFOC$="="THENO1$=OP$:MV$="  ":PC$="    ":IL=0:GOTO1160
810 IFOP$=""THENAM$="G":IL=1:GOTO1060
820 IFOP$="A"THENAM$="H":IL=1:GOTO1060
825 X=0:Y=0:I=0:M=0:Z=0
830 FORQ=1TOLEN(OP$):Q$=MID$(OP$,Q,1):IFQ$=")"THENI=1:GOTO865
840 IFQ$="#"THENM=1:GOTO865
865 NEXTQ
866 FORQ=1TOLEN(OP$)-1:Q$=MID$(OP$,Q,2)
867 IFQ$=",Y"THENY=1:GOTO870
868 IFQ$=",X"THENX=1
870 NEXTQ
875 O1$=OP$:GOSUB6000
876 IFNU<256THENZ=1
880 IFLEFT$(OC$,1)="B"ANDOC$<>"BRK"ANDOC$<>"BIT"THEN1000
890 IFZTHEN940
900 IFXTHENAM$="K":GOTO1030
910 IFYTHENAM$="L":GOTO1030
920 IFITHENAM$="M":GOTO1030
930 AM$="N":GOTO1030
940 IFMTHENAM$="I":GOTO1030
950 IFIANDYTHENAM$="O":GOTO1030
960 IFIANDXTHENAM$="P":GOTO1030
970 IFXTHENAM$="Q":GOTO1030
980 IFYTHENAM$="R":GOTO1030
990 AM$="S":GOTO1030
1000 AM$="J":IFNU>PC+1THENOS=NU-PC-2:IFOS>127THENER=1
1010 IFNUMBER<PC+1THENOS=254+NU-PC:IFOS<128THENER=1
1020 IFER=1THENPRINT"█TOO LONG CONDITIONAL BRANCH":GOTO300
1025 FO=OS:IL=2:GOTO1060
1030 IFZ=0THEN1050
1040 FO=NU:IL=2:GOTO1060
1050 SO=INT(NU/256):FO=(NU/256-SO)*256:IL=3
1060 RESTORE:FORW9=1TO56:READI$:IFLEFT$(I$,3)=OC$THENCD$=I$:W9=100
1070 NEXTW9:IFW9=57THENPRINT"█ILLEGAL MNEMONIC":GOTO300
1080 FORW9=4TOLEN(CD$)STEP3:IFMID$(CD$,W9,1)=AM$THENLW=W9:W9=100
1090 NEXTW9:IFW9<100THENPRINT"█ILLEGAL ADDRESSING MODE":GOTO300
1100 MV$=MID$(CD$,LW+1,2):N$=MV$:GOSUB7000
1110 POKEPC,V:IFIL=1THEN1140
1120 POKEPC+1,FO:IFIL=2THEN1140
1130 POKEPC+2,SO
1140 N=PC:GOSUB9000:PC$=R$:PC=PC+IL
1150 N=FO:GOSUB9000:FO$=R$:N=SO:GOSUB9000:SO$=R$
1160 IFIL<3THENSO$="  "
1170 IFIL<2THENFO$="  "
1175 IF AM$="H"THENO1$="A"
1180 PRINT#1,TLEFT$("    ",4-LEN(STR$(T)))PC$" ";
1190 PRINT#1,MV$" "RIGHT$(FO$,2)" "RIGHT$(SO$,2)"  ";
1200 PRINT#1,LB$LEFT$("        ",8-LEN(LB$))OC$LEFT$("     ",5-LEN(OC$));
1210 PRINT#1,O1$:O1$=""
1220 NEXTT:GOTO300
3999 END
4000 IN$="":NL=0:PRINTTAB(TB);
4020 PRINT"█ █▌";
4030 GETGT$:IFGT$=""THEN4030
```

```
4031 IF GT$>"Z"ORGT$<" "ANDGT$<>CHR$(13)
     ANDGT$<>CHR$(20)THEN4030
4035 NL=NL+1
4040 IF GT$=CHR$(20)ORGT$=CHR$(13)THEN4100
4045 IFGT$=" "THENPRINT" ";:RETURN
4050 PRINTGT$;:IN$=IN$+GT$
4060 IF NL=LTTHEN4100
4070 GOTO4020
4100 IF GT$<>CHR$(20)THEN4150
4105 IF LEN(IN$)<2THEN4120
4110 PRINT" ■■";:NL=NL-2:IN$=LEFT$(IN$,
     LEN(IN$)-1):GOTO4020
4120 IFLEN(IN$)=0THENNL=NL-1:GOTO4020
4130 PRINT" ■■";:NL=NL-2:IN$="":GOTO4020
4150 IFGT$=CHR$(13)THENPRINT" "
4160 RETURN
5000 DATAADCN6DS65I69K7DL79P61071Q75
5010 DATAANDN2DS25I29K3DL39P21031Q35
5020 DATAASLH0AN0ES06K1EQ16
5030 DATABCCJ90,BCSJB0,BEQJF0
5060 DATABITN2CS24
5070 DATABMIJ30,BNEJD0,BPLJ10,BRKG00
5110 DATABVCJ50,BVSJ70,CLCG18,CLDGD8
5150 DATACLIG58,CLVGB8
5170 DATACMPNCDSC5IC9KDDLD9PC10D1QD5
5180 DATACPXNECSE4IE0
5190 DATACPYNCCSC4IC0
5200 DATADECNCESC6KDEQD6
5210 DATADEXGCA,DEYG88
5230 DATAEORN4DS45I49K5DL59P41051Q55
5240 DATAINCNEESE6KFEQF6
5250 DATAINXGE8,INYGC8
5270 DATAJMPN4CM6C
5280 DATAJSRN20
5290 DATALDANADSA5IA9KBDLB9PA10B1QB5
5300 DATALDXNAESA6IA2LBERB6
5310 DATALDYNACSA4IA0KBCQB4
5320 DATALSRH4AN4ES46K5EQ56
5330 DATANOPGEA
5340 DATAORAN0DS05I09K1DL19P01011Q15
5350 DATAPHAG48,PHPG08,PLAG68,PLPG28
5390 DATAROLH2AN2ES26K3EQ36
5400 DATARORH6AN6ES66K7EQ76
5410 DATARTIG40,RTSG60
5430 DATASBCNEDSE5IE9KFDLF9PE10F1QF5
5440 DATASECG38,SEDGF8,SEIG78
5470 DATASTAN8DS85K9DL99P81091Q95
5480 DATASTXN8ES86R96
5490 DATASTYN8CS84Q94
5500 DATATAXGAA,TAYGA8,TSXGBA,TXAG8A
5510 DATATXSG9A,TYAG98
6000 AD=0
6005 Q$=LEFT$(OP$,1):IFQ$="$"ORQ$="%"OR
     (ASC(Q$)>64ANDASC(Q$)<91)THEN6030
6010 IFASC(Q$)>47ANDASC(Q$)<58THEN6030
6020 OP$=RIGHT$(OP$,LEN(OP$)-1):GOTO6000
6030 Q$=RIGHT$(OP$,1):Q1=ASC(Q$):IF(Q1
  >47ANDQ1<58)OR(Q1>64ANDQ1<91)THEN6050
6035 IFQ$="+"THEN6050
6040 OP$=LEFT$(OP$,LEN(OP$)-1):GOTO6030
6050 IFRIGHT$(OP$,2)=",X"THENOP$=
     LEFT$(OP$,LEN(OP$)-2)
6052 IFRIGHT$(OP$,2)=",Y"THENOP$=
     LEFT$(OP$,LEN(OP$)-2)
```

```
6053 IFRIGHT$(OP$,1)=")"THENOP$=LEFT$(OP$,LEN(OP$)-1)
6055 IFLEFT$(OP$,1)="$"THENN$=OP$:GOSUB7000:NUMBER=V:GOTO6100
6060 IFLEFT$(OP$,1)="%"THENN$=OP$:GOSUB8000:NUMBER=V:GOTO6100
6070 IFASC(LEFT$(OP$,1))<58THENNUMBER=VAL(OP$):GOTO6100
6075 IFRIGHT$(OP$,1)="+"THENAD=AD+1:OP$=LEFT$(OP$,LEN(OP$)-1):GOTO6075
6080 FORW1=1TOM2:IFS$(W1)=OP$THENNUMBER=V(W1):W1=999
6090 NEXTW1:IFW1=M2+1THENPRINT"■UNDEFINED SYMBOL ERROR":GOTO300
6100 NU=NU+AD:RETURN
7000 IFLEFT$(N$,1)="$"THENN$=RIGHT$(N$,LEN(N$)-1)
7010 V=0:IFLEN(N$)=4THEN7030
7020 N$=LEFT$("0000",4-LEN(N$))+N$
7030 FORR2=1TO4:D$=MID$(N$,R2,1):TV=ASC(D$)-48:IFTV>9THENTV=TV-7
7040 V=TV*16↑(4-R2)+V:NEXTR2:RETURN
8000 IFLEFT$(N$,1)="%"THENN$=RIGHT$(N$,LEN(N$)-1)
8010 V=0:FORZ=LEN(N$)TO1STEP-1:V=V+VAL(MID$(N$,Z,1))*2↑(LEN(N$)-Z):NEXTZ
8020 RETURN
9000 FD=INT(N/4096):N=(N/4096-FD)*4096:SD=INT(N/256):N=(N/256-SD)*256
9010 TD=INT(N/16):N=INT((N/16-TD)*16):R$=MID$(H$,FD+1,1)+MID$(H$,SD+1,1)
9020 R$=R$+MID$(H$,TD+1,1)+MID$(H$,N+1,1):RETURN
10000 IFA$(T)=""THENOC$="":LB$="":GOTO10100
10005 LI(1)=0:LI(2)=0:LI(3)=0:LI=0
10010 FORR2=1TOLEN(A$(T)):IFMID$(A$(T),R2,1)=" "THENLI=LI+1:LI(LI)=R2
10020 NEXTR2:IFLI(3)=0THENLI(3)=R2-1
10030 LB$=LEFT$(A$(T),LI(1)):OC$=MID$(A$(T),LI(1)+1,LI(2)-LI(1))
10040 OP$=RIGHT$(A$(T),LI(3)-LI(2)+1)
10050 IFLB$=" "THENLB$="":GOTO10070
10060 LB$=LEFT$(LB$,LEN(LB$)-1)
10070 OC$=LEFT$(OC$,LEN(OC$)-1)
10080 IFOP$=" "THENOP$="":GOTO10100
10090 OP$=RIGHT$(OP$,LEN(OP$)-1)
10100 RETURN
```

Machine Language: Uncrashing

Jim Butterfield
Toronto, Canada

Techniques for uncrashing a PET/CBM system were outlined in **COMPUTE!** issue 1. It's worth while going over it again.

The term "crash" sounds noisy; but in fact, a computer system that crashes goes utterly silent. It no longer responds to the keyboard, and can't be controlled. A crash can be corrected by turning the power off and on, but in doing so you will lose the contents of RAM memory. Not only will you lose your program, but it will be much harder to investigate why the crash happened.

Three Types Of Uncrash

There are three ways of getting your machine out of a crash: soft reset, NMI and Reset line. I recommend the Reset line method for reasons that will become visible later.

A soft reset system involves planting a program in the interrupt system which will check the RUN/STOP key, or a selected combination of keys, at periodic intervals. If it finds the RUN/STOP key is pressed, the interrupt routine automatically sets the main program to a fixed restart point.

An NMI reset has a hardware configuration that causes a signal received on the NMI (Non Maskable Interrupt) line to send the processor to a fixed restart point. It requires no program, just a setting of the NMI vector address to point at the desired restart location.

Both of the above methods depend upon the interrupt system being able to function properly. There's a class of crash that leaves the interrupts completely out of action; in this case neither the soft reset nor the NMI reset will work. Additionally, Original ROM PETs don't support NMI reset.

The Reset line method involves sending a signal on the processor's Reset line, of course. With many processors you are given a button or key to press to cause this; the processor will then ask you what you want to do: Machine language or Basic? In some cases the Reset key is too accessible, and users must learn how to get back to their program after they have pressed the button accidentally.

On the PET, you are supplied with no button for Reset; and even after you arrange to connect it yourself, you have a second problem to solve: how to stop Basic from going into cold start and wiping memory. This problem cannot be completely solved for Original ROM PETs, but can be done on more recent systems. We'll give details on this later.

Types of crash

If you lead a charmed life, your programs will run without fault and you'll never need uncrashing methods. If you're very lucky, your programs will give wrong answers but will run without crashing. Most mortal programmers, however, write machine language programs that crash once in a while. The crashes can be broadly divided into four categories: Break, Tight Loop, X2, and Battered Basic.

Break is not a true crash, since you don't lose control of the system. When your machine language program goes wild, it eventually stumbles into a location that contains zero — which is read as a Break instruction. The processor flips into the monitor, and you're saved. On early PET systems that have no built-in monitor, you're likely to see an ?INVALID QUANTITY error. Either way, you have miraculously escaped disaster.

Tight Loop crashes are situations where the machine language program gets into a loop from which there is no exit. This may be a coding error on your part; or the program might have branched to entirely the wrong place and have gotten itself locked up there. You can usually get out of this type of crash with a soft reset system or an NMI reset.

X2 crashes are the tough ones. The processor has encountered an Op Code which ends in binary 0010, or 2 in hexadecimal, and it's not A2 which is a legitimate LDX command. The microprocessor goes into an internal "race" condition: it will cease to execute code and will ignore all interrupts. Neither soft reset nor NMI will help here, since both are based on interrupts. The only thing that works is Reset; and in the PET, Reset causes Basic cold start; and Basic cold start causes memory to be wiped; and wiped memory causes programmers to gnash their teeth and tear their hair out. For all PETs except those fitted with Original ROMs, there is a better way; it's called the Reset line method.

Battered Basic may be a side issue, but can be annoying if you don't understand it. Basic cannot work unless a number of locations, mostly in zero page, contain the right values. Your program may have changed some of these locations — indeed, you may have deliberately swapped out parts of zero page to give yourself working space. If so, you can get back to the Machine Language Monitor but it would be unreasonable to expect Basic to work. That's often all right, since you'll be doing your debugging work largely in the Machine Language Monitor and won't need Basic. Just be sure you know about this potential problem, so that you won't be disturbed by Basic's inability to function.

For PET/CBM Upgrade and subsequent ROMs, the following locations are often needed to sustain Basic:

13-15	**String descriptor pointers**
28-29	**Start of Basic pointer**
34-35	**Limit of Basic memory pointer**
50	**Garbage yardstick**
51	**Jump op code (4C)**
65	**Floating accumulator overflow**
70-87	**CHRGET subroutine**
0400	**Basic start (00)**

On other machines, the equivalent locations can be found. You may wish to note their contents in case you ever need to fix up a sick Basic.

The Reset Line Method

This is good for Upgrade and subsequent ROM systems on PET/CBM. The basic method is: set the Diagnostic Sense line to ground, then place a momentary ground on the Reset line. The system will awake in the Machine Language Monitor, but the Stack Pointer is set at 01, which means trouble unless you use a special procedure. Type a semicolon and press the RETURN key. This seems to do nothing but give you an empty line, but it's important. Now go back and change the Stack Pointer (the value under SP) from 01 to FA and press RETURN. Your machine is now cleanly in the Machine Language Monitor. You may also return to Basic if Basic is still in working order.

There are two hardware connections you need to make for Reset Line uncrashing. If you're not technically oriented, you might ask your dealer to hook up the buttons for you. On the original PET hardware (edge connector on the right hand edge of the board), the Reset line is pin 27 of the memory expansion connector and the diagnostic sense pin is pin 5 of the parallel user port. On newer PET hardware (memory expansion pin connector at the right of the board), the Reset line is available on pin J4-22 of the front pin connector, and the Diagnostic line at both pin J9-20 of the rear pin connector as well as on pin 5 of the parallel user port.

Whether you hook it up yourself, have a friend or dealer do it for you, or buy a commercial product, you'll find a Reset line uncrasher to be well worth-while. If you never make mistakes, you won't need one, of course.

Summary

It's good to be able to re-awaken a dead machine. It gives you the chance to pick through memory to find where the fatal error might have occured; or even to restart your program, implanting Break instructions at crucial points to pin down the problem.

Original ROM PETs can't be revived in many cases; if you do much machine language work you'll be wise to upgrade your ROM set. Newer PETs need only a little hardware and a procedure which you can learn, and you'll be able to uncrash in almost all cases.

Whatever machine you use, remember that Basic may not survive a serious crash. Be prepared to use your Machine Language Monitor for most of your testing and debugging.

Beginning programmers often feel a sense of dread when their machine goes dead. There's no need. Crashes are part of the game. Learning how to come cleanly out of them — and pin them down — can be part of the fun of programming. ©

Notes On The Pet SAVE Command

Louis F. Sander
Pittsburgh, PA

You can SAVE several copies of a program by using a FOR ... NEXT loop. Giving the direct mode command:

```
FOR I=1 TO 3 : SAVE "SPACEWARS" : NEXT
```

will SAVE three consecutive copies of SPACEWARS without further action from you. The same method also works in program mode. If you're developing a program and want to make periodic backup copies, put these two lines into the program:

```
63998 END
63999 FOR I=1 TO 3 : SAVE "PROGRAM NAME" :
  NEXT
```

When you're ready to make backup copies, just get into direct mode and enter GOTO 63999, and you'll get three copies. Line 63999 has the highest line number allowed by BASIC, insuring that the automatic save routine keeps away from the rest of your program. Line 63998 keeps your program from inadvertently SAVEing itself.

You have great flexibility in naming a program you are about to SAVE. You can give it the name it had when it was LOADed, or you can give it a completely new name. Or you can SAVE it with no name at all. The name of a program to be SAVEd can appear within quotes, or as a string variable:

```
FOR I=1 TO 3 : A$ = "PROGRAM #" + STR$(I):
  SAVE A$ : NEXT
```

will SAVE your program three times, under the names PROGRAM #1, PROGRAM #2, and PROGRAM #3.

You can also use a FOR ... NEXT loop to VERIFY multiple copies of the same program, SAVEd as described above:

```
FOR I=1 TO 3 : VERIFY : NEXT
```

will make PET do the job without further instructions from you. If a VERIFY ERROR occurs, however, the loop will abort and PET will return to the READY state.

Be extremely careful about your RECORD button when you are SAVEing and VERIFYing, especially when these are done under program control. Trying to SAVE *without* the RECORD button will leave your tape exactly as it was before the SAVE, and trying to VERIFY *with* the RECORD button depressed will erase the tape. Remember — PET can't tell the difference between its cassette recorder buttons. It only knows whether one or more of them are down, and it can't tell which ones or how many you've pushed.

Like most other BASIC commands and statements, SAVE can be abbreviated. The proper abbreviation is 'Sa', which prints on the screen as 'S(spade)' if your PET is in graphics mode. Similarly, VERIFY can be abbreviated as 'Ve', and LOAD as 'Lo'. The abbreviated forms work exactly like the fully-spelled ones, but they are easier to type. ©

Product Review:

The Optimized Data Systems PH-001 2114 RAM Adapter

David C. Swaim
Marietta, GA

I turned on my PET 2001-8 one day, loaded a program and typed RUN. The computer went into never-never land leaving me with no option but turning it off and back on. When I did this I got the "*** COMMODORE BASIC ***" but only 2612 bytes free. I had a bad RAM chip. If you own an "original" PET 2001-8 and recently had a RAM chip go bad you already know the next part of the story. The 2001-8 PET uses the 6550 RAM chips. These chips are out of production and if you can locate one you will pay a premium price for it. Eventually you will not be able to get one at any price. Just when I was feeling like my computing days were over I discovered Optimized Data Systems (P.O. Box 595, Placentia California 92607). These fellows make a handy adapter which allows you to replace your 6550 RAMs with the more available (and cheaper) 2114 RAMs. Their PH-001 2114 RAM Adapter allows you to replace up to 4K of memory one chip at a time so you can keep using your good 6550 RAMs. This was obviously just what I needed.

Being impatient and knowing only that they made an adapter I decided to call them (area code 714 996-3201, don't call collect). The person I talked to was very helpful and explained what they made. You can get the RAM adapter in various stages of readiness. The PHB-001 is a bare double-sided printed circuit board and sells for $8.95 (add $1.50 postage and handling to all prices, California residents add sales tax). You supply all the other parts. The other parts needed are two 22 pin wire-wrap sockets, a 74LS139 dual decoder, two 0.1 uf capacitors and 2114s and sockets. If you think you might have trouble finding these parts the PHK-001 at $13.95 contains all the parts to make the adapter with two 2114 sockets. The PHK-001S at $16.95 is the complete kit including eight sockets for your 2114s. If you are not adept at soldering miniature PC boards (the lands on the adapter board are pretty close together) I recommend the assembled and tested PH-001S adapter at $22.95. The 2114 RAMs are extra on all of the above. I only needed one so that's all I got. I ordered the assembled version plus one 2114 RAM over the phone Monday afternoon (phone charge card orders only). On Thursday I received the adapter in the mail. I can only say that shipment was prompt. The adapter itself was in a sealed plastic bag. The 2114, which I almost threw away with the packing, was pressed onto a small piece of conductive foam and was loose in the packing material. They included a self addressed stamped post card requesting an honest response to the product on which I suggested they at least put the 2114s in a plastic bag in the future.

Installation of the adapter is simple. First you remove the last two 6550 RAMs from the main PC board on the PET. These are inserted into the 22 pin sockets on the adapter board. These two sockets are wire-wrap sockets and they plug directly into the now empty sockets on the PET. Then when any 6550 RAM in the top 4K goes bad simply unplug it and plug a 2114 into the corresponding socket on the adapter board (a diagram is provided). With two of these adapter boards the entire 8K of memory can be replaced with 2114s.

The user instructions that come with the PH-001 are pretty thorough with adequate warnings on avoiding static electricity while handling the RAMs. The only problem I had was figuring out which way the 2114 went onto the circuit board. After studying the parts layout diagram I was able to figure it out. The last page of the directions gives a very detailed description of how the adapter works and suggests the possibility of using the PH-001 as a 4K memory expansion (Hmm.......).

The materials and workmanship on the PH-001 are excellent and the peace of mind is well worth the small price. ©

Discovering Tape File Names

David R. Heise
Chapel Hill, NC

A PET BASIC program can open a data file on tape without knowing the file name. The statement:

OPEN 1,1

causes the PET to search the tape and open the first file it encounters.

The PET gets the file's name, but the information is not provided to the program. The file name is read from tape and stored in the tape buffer when the open statement is executed. It stays there until an INPUT or GET statement is executed for the file.

The program below demonstrates a procedure for retrieving the unknown file name and storing it in a BASIC string that can be used as desired.

The program itself also is of interest. It automatically transfers data files on a tape over to a disk, retaining the original file names. (Names are truncated to 16 characters if necessary since 16 is the maximum for disk file names.) The program continues transferring files until you press the STOP key.

```
100 REM MAKE A STRING THE FIRST VARIABLE IN THE PROGRAM
110 AA$=""
120 REM FIND OUT WERE IT IS DEFINED
130 X=PEEK(42)+256*PEEK(43)
140 REM REDEFINE IT SO IT REFERS
    TO CELLS IN FIRST TAPE BUFFER
150 POKE X+3,127:POKE X+4,2:POKE X+2,16
160 :
170 REM PAUSE FOR USER
180 PRINT
 "PRESS A KEY WHEN READY TO TRANSFER TAPE FILES TO DISK"
190 GET A$:IF A$="" THEN 190
200 :
210 REM GET THE NEXT TAPE FILE
220 OPEN 1,1
230 REM AA$ HAS ITS NAME PLUS
    BLANKS SPACING OUT TO 16 CHARACTERS
240 PRINT AA$
250 REM REMOVE THE BLANKS
260 FOR I=16 TO 1 STEP -1
270 IF MID$(AA$,I,1)=" " THEN NEXT I
280 C$=LEFT$(AA$,I)
290 I=1:NEXT I
300 REM OPEN THE DISK ERROR CHANNEL
310 CLOSE 15:OPEN 15,8,15
320 REM OPEN THE DISK DATA FILE
330 OPEN 2,8,5,"0:"+C$+",S,W"
340 REM CHECK FOR PROBLEMS
350 INPUT#15,A$,B$,C$,D$:IF A$<>"00" THEN PRINT B$:STOP
360 REM GET A DATUM FROM THE TAPE FILE
370 GET#1,A$
380 REM IF THE TAPE FILE IS DONE,
    CLOSE THE READ AND WRITE FILES AND REPEAT
390 IF (ST)=64 THEN CLOSE 1:CLOSE 2:GOTO 220
400 REM OTHERWISE BUILD THE DISK FILE
410 PRINT#2,A$;
420 GOTO 370
```

©

Petbug

John Blackburn
London, UK

There is a bug in the machine language tape SAVE routine which only applies to 32K machines. I first discovered this when saving a self-relocating program which automatically packs itself into the top of memory. When loaded, it didn't run. It turns out that the byte at $7FFF gets corrupted.

I spent a lot of time trying to find out what I was doing wrong. Finally, I decided that there's only one thing to do at a time like this. So I did it.

Jim replies, "The problem is at $FC08 to $FC0B in 4.0 systems; the high bit of the tape write address (in C7 and C8) is used as a flag to signal end-of-write ... with the result that LDA $C8, BPL WRITEBYTE drops through one byte early and the byte at $7FFF gets confused with the checksum byte."

So, if you're writing this sort of program, put in a dummy byte at the end, so avoiding $7FFF.

If you've got one and it gives trouble, set top-of-memory to $7FFF before you load it (instead of leaving it at $8000 as at power-up). Then save it again the easy way. All your SYS commands will be one byte lower, but doubtless the screen will inform you of the changes. ©

Review:

Machine Language Utility-Pac (PET ROM)

Competitive Software
21650 Maple Glen Drive
Edwardsburg, Mi 49112
$79.95

Harvey B. Herman
Greensboro, NC

PETs with version 2.0 (upgrade) and version 4.0 ROMs have a built-in machine language monitor. It, however, is fairly primitive and many people have felt the need for enhancement. User's groups have distributed a program by Jim Butterfield (among others) called Supermon. The program adds about seven more commands (e.g. disassemble and hunt) to the original monitor. It must be loaded from tape, a time consuming procedure, if one is sans disk. Furthermore, it takes up valuable RAM space. Plugging in the 4K ROM from Competitive Software (specify 2.0 or 4.0 ROM and $9000 or $A000 as a starting location) gives an additional 34 commands and no waiting for tape to load. I was impressed with all they were able to fit into a 4K ROM.

There was no trouble installing the ROM ($A000 in my version) as the instructions are quite explicit. A simple SYS call enables and jumps to the monitor. I tried a representative sample of the commands listed below and they all worked. A few act exactly like Supermon, but I said the ROM includes many additional commands. It really is convenient to have the routines always at the ready, especially if you don't have a disk. Even if you do there are extra disk support commands (like the wedge) which the company has fitted into the remaining free space on the ROM, after the monitor. I think the Utility-Pac ROM should be seriously considered by PET machine language hackers who have the need for a comprehensive monitor.

```
COMMODORE MONITOR COMMANDS

.G  GO RUN
.L  LOAD FROM TAPE
.M  MEMORY DISPLAY
.R  REGISTER DISPLAY
.S  SAVE TO TAPE OR DISK
.X  EXIT TO BASIC
```

```
MACHINE LANGUAGE UTILITY-PAC 1.2R COMMANDS

.A  SIMPLE ASSEMBLER
.B  BREAK SET
.C  COMPARE MEMORY
.D  DISASSEMBLER
.F  FAST TYPE
.H  HUNT MEMORY
.I  INTEGRATE MEMORY
.N  NEW LOCATER
.Q  QUICK TRACE
.T  TRANSFER MEMORY
.W  WALK CODE
.P  PRINTER MODE ON
.O  PRINTER MODE OFF
.V  VIDEO SCREEN DUMP TO PRINTER
.Z  RELOCATE
.$  HEX TO DECIMAL (4 DIGIT HEX)
.#  DECIMAL TO HEX (4 DIGIT HEX)
.>  DECIMAL TO HEX & ASCII (2 DIGIT HEX)
.<  HEX TO DECIMAL & ASCII (2 DIGIT HEX)
.↑  ASCII TO DECIMAL & HEX (2 DIGIT HEX)
.*  AUTO REPEAT ON OR OFF
.+  PLUS
.-  MINUS
.@ JUMPS TO USER PROGRAM LOCATION 1
.\ (BACKSLASH) JUMPS TO USER  PROGRAM LOCATION 2

USE SHIFTED CHARACTERS FOR THE FOLLOWING COMMANDS

.F  FILL MEMORY
.A  AND
.O  OR
.E  EXCLUSIVE OR
.C  1'S COMPLIMENT
.I  INVERSE +/-
.N  NO COMPARE TO HEX
.V  ENHANCED VIDEO SCREEN DUMP TO PRINTER
.M  MEMORY TEST
```

Additional Commands from BASIC

REV PRINT: Lists programs to printer in UPPER AND LOWER CASE
DOS: Allows easy WEDGE commands to operate disk system
SCREEN DUMP: Dumps screen to printer automatically
ENHANCED SCREEN DUMP: Dumps enhanced Video Image to printer!
RE-NEW: Restores program if you accidentally wipe it out
AUTO-CURSOR: Allows automatic repeating of all keys
DISK APPEND: Allows merging of disk files
REV SCREEN: Reverses display on screen
DISPLAY: Displays memory locations in operation ©

The Single Board 6502

Eric Rehnke

KIM Is Dead!!! Long Live KIM!!!

A few minutes ago, I confirmed the rumor that Commodore is no longer producing the KIM-1. The person I spoke with at Commodore didn't offer any reason for the decision so one must assume that other high priority projects like VIC and PET needed the production capacity and resources that were being allocated to support KIM.

While I can fully understand this move, that doesn't make it easier to accept. I cut my computer eye teeth on KIM and still use this remarkable system for all of my work (my Apple and PET are gone now). Of all the 6502 single board computers, KIM still is the favorite CPU in my disk based system.

If you feel about KIM as I do, please bow your head and observe a moment or two of silence.

New Chips From MOS Technology

In my last column I mentioned some of the recent developments in the 6500 family coming from Rockwell. Since they were of the single chip computer variety, they were probably of limited interest to most of you hobbyists and designers of small quantity custom systems.

Well, if you're done crying in your soymilk, I have some good news for you. It seems that the folks at MOS Technology have also been busy with some developments of their own to enhance the 6500 family. Three new members of the family, a new CPU, and two I/O ports, are being introduced (or will be introduced shortly). The new CPU is designated the 6508 and features a built-in 8 bit I/O port, 256 bytes of RAM as well as the 16 address and 8 data lines necessary for full 64K expansion. The 6508 has been designed for multiprocessing as evidenced by the tri-state address bus and the need for an external two-phase clock.

One of the new I/O ports is the 6523, which contains three 8-bit I/O ports. These ports operate in a similar fashion to those on the earlier 6530's. That is, they have no automatic handshaking capability and are used as simple I/O ports. Of course, each of the 24 I/O lines on the 6523 is individually programmable.

The 6525 is the other new I/O port. It is similar to the 6523 except that the 6525 has an additional operating mode available giving it a rather powerful capability. Depending on the mode selected, the 6525 can provide 24 individually programmable I/O lines *or* 16 I/O lines, 2 handshake lines and 5 *priority interrupt inputs!!!* How's that for capability? And the priority interrupt operation is fully automatic.

The only thing that disappointed me about these new chips was the lack of a hardware interval timer. I would be willing to give up some of the addressing space (who uses 65K of memory in a small controller anyway?) or even 50 or so bytes of RAM to have an interval timer in the CPU chip. (Maybe *I'd* better quit crying in my soymilk.)

Anyway, here are the pinouts for the new devices:

6508

RES	1	40	Ø2 IN
Ø1 IN	2	39	R/W
IRQ	3	38	DB_0
AEC	4	37	DB_1
VDD	5	36	DB_2
A_0	6	35	DB_3
A_1	7	34	DB_4
A_2	8	33	DB_5
A_3	9	32	DB_6
A_4	10	31	DB_7
A_5	11	30	P_0
A_6	12	29	P_1
A_7	13	28	P_2
A_8	14	27	P_3
A_9	15	26	P_4
A_{10}	16	25	P_5
A_{11}	17	24	P_6
A_{12}	18	23	P_7
A_{13}	19	22	A_{15}
VSS	20	21	A_{14}

If I get a chance to play with some of these devices in the near future, perhaps I'll present some simple circuits.

Another significant bit of information I'm sure you'll be interested in is the fact that MOS Technology is working on a CMOS version of the 6502 and is planning to introduce a complete family of CMOS microprocessor devices. The factory is fairly tight-lipped about just what sort of devices will be included in their CMOS line, but said it would be a *complete* family. (To me, that means CPU, I/O, RAM and ROM/EPROM.) Oh well ... time will tell.

Get more information on the 6508, 6523, and

6523/6525

VSS	1	40	DB7
PA0	2	39	DB6
PA1	3	38	DB5
PA2	4	37	DB4
PA3	5	36	DB3
PA4	6	35	DB2
PA5	7	34	DB1
PA6	8	33	DB0
PA7	9	32	PC7
PB0	10	31	PC6
PB1	11	30	PC5
PB2	12	29	PC4
PB3	13	29	PC3
PB4	14	28	PC2
PB5	15	26	PC1
PB6	16	25	PC0
PB7	17	24	RS0
CS	18	23	RS1
WRITE	19	22	RS2
VDD	20	21	RST

6525 from MOS Technology, 950 Rittenhouse Rd., Norristown, PA 19403 (215) 666-7950.

Here's a list of some application notes that are available from MOS Technology:

Applications Brief #	Title
1	Octal Bus Transceiver, Our Equivalent Asynchronous Two-Way Communicator, TTL-CMOS Compatible
2	MPS 6525 Versus MC6821/8255
3	Handling The 'RDY' Line In 6500 CPU's
4	MPS6520 Versus MC6820/ MC6821
5	Dual Processor Configuration With The MPS6508
6	6551 Versus 6850
7	Memory Expansion With The 6508
9	MPS 6508-An Active CPU During Block DMA
11	MPS6508-Handling Vectored Interrupts
12	MPS6525-A Two In One Deal

They can be obtained from MOS Technology (950 Rittenhouse Rd., Norristown, PA 19403)

Modem Update

Almost there ... but not quite. If you read my column in the June/July 1980 issue of **Compute II**, you are aware of my interest in modems. Quite frankly, I have been searching for the "perfect" modem. Here are my requirements.

1. Under $300.
2. Direct connection to phone line w/FCC approved interface.
3. Has capability for autodial/autoanswer under computer control.
4. Originate/Answer mode can be set by computer.
5. Communicates at 300 baud.
6. Interfaces easily with my computer (probably through an RS232 port).

While at the 6th West Coast Computer Faire, I did find one that met all the requirements except for number 4. This modem is called the Microconnection and is made by The Microperipheral Corp. (2643 151st Place N.E., Redmond, WA 98052 206-881-7544). The fact that the Originate/Answer mode is only manually settable from a switch on the front panel and not by the computer, takes it off my list. *(Editor's Note ... picky, picky. RCL)* But this doesn't mean you have to take it off your list. The reason I want this feature is for a possible automated store and forward system where another computer calls my computer for a message for yet another computer somewhere else. Without the ability of my computer to set the mode under software control, someone would have to manually set the switch to Answer while the system is waiting for a call, or to Originate when the system is ready to make a call to the other computer. If you don't need this feature, then the Microconnection could be considered for your application. *(Editor's Note ... Now I understand. RCL)*

Strangely enough, as I was writing this article, the latest issue of Electronic Design (April 16, 1981) reached my desk. In it was an article on how to build a 300 baud modem (with ALL my requirements) from scratch using some preassembled filter modules from Cermetek (1308 Borregas Ave., Sunnyvale, CA 94086). These modules are used in the D.C. Hayes Micromodem which is available for the Apple II or the S-100 bus.

If it turns out that I can't find a commercially available modem that meets all my requirements, I may have to build it from scratch. My rule of thumb on building things from scratch is — IF IT'S AVAILABLE DON'T BUILD IT. If I figure all the time involved in chasing around for parts and the time it takes to actually build and debug the project, I come out WAY ahead if I just spend the extra bucks and let someone else do all the hard work.

Come to think of it, I may be able to MODIFY the Microconnection to suit my needs. Hmmmm....

Real Time Clock Subroutine

I wrote this subroutine a few years ago just as an exercise and thought you might like to see it.

As opposed to the more conventional straight line coding of real time clocks, this one uses a loop to increment each value and check it against the limit. Five zero page locations are required to hold the time and they must be properly initialized prior to use. Keep in mind that this is just a SUBROUTINE. The complete real time operating system must be written which uses RTCLK as the clock update routine.

Can you write one that is shorter?? If so, send it in and it may be published. How about one that's shorter AND faster?

```
HDE ASSEMBLER REV 2.2

 LINE#    ADDR    OBJECT      LABEL  SOURCE            PAGE 0001

01-0005  2000                 ;6502 REALTIME CLOCK SUBROUTINE
01-0010  2000                 ;WRITTEN BY E REHNKE 12,31,77
01-0015  2000                 ;THE IDEA COMES FROM BYTE MAG
01-0020  2000                 ;NOV. 77 P. 50. IT WAS CONVERTED
01-0025  2000                 ;FROM A 6800 PROGRAM JUST TO SEE
01-0030  2000                 ;HOW EASY IT BE TO DO.
01-0035  2000                 ;
01-0040  2000                 ;THIS ROUTINE IS SETUP TO USE
01-0045  2000                 ;THE HARDWARE FROM BYTE, NOV.77
01-0050  2000                 ;P. 72 AND CONSISTS OF THE 60 HZ.
01-0055  2000                 ;LINE FREQ, DIVIDED DOWN AND
01-0060  2000                 ;PROPERLY CONDITIONED TO FORM A
01-0065  2000                 ;TTL COMPATIBLE 15 HZ. CLOCK WHICH
01-0070  2000                 ;IS APPLIED TO THE INTERRUPT INPUT
01-0075  2000                 ;ON KIM.
01-0080  2000                 ;
01-0085  2000                 ;THE FOLLOWING ZPAGE CLOCK REGS
01-0090  2000                 ;MUST BE INITIALIZED TO THE
01-0095  2000                 ;CORRECT TIME PRIOR TO STARTING
01-0100  2000                 ;THE REAL TIME CLOCK.
01-0105  2000                 ;THEY ARE AS FOLLOWS:
01-0110  2000                 ;         $00E0= DAYS COUNTER
01-0115  2000                 ;         $00E1= HOURS COUNTER
01-0120  2000                 ;         $00E2= MINUTES COUNTER
01-0125  2000                 ;         $00E3= SECONDS COUNTER
01-0130  2000                 ;         $00E4= FRACTIONAL COUNTER
01-0135  2000                 ;
01-0140  2000                        *=$E0
01-0145  00E0                 CLKREG *=*+5
01-0150  00E5                 ;
01-0155  00E5                        *=$2000
01-0160  2000  F8             CLOCK  SED               ;WORK IN THE DECIMAL MODE
01-0165  2001  A2 04                 LDX #$4           ;INITIAL OFFSET
01-0170  2003                 ;
01-0175  2003  18             CKLOOP CLC
01-0180  2004  B5 E0                 LDA CLKREG,X      ;GET THE TIME...
01-0185  2006  69 01                 ADC #$1           ;DO A DECIMAL INCREMENT...
01-0190  2008  95 E0                 STA CLKREG,X      ;AND PUT IT BACK.
01-0195  200A  DD 17 20              CMP TABLE,X       ;CHECK IT AGAINST THE LIMIT.
01-0200  200D  D0 07                 BNE RETURN        ;NOT YET? THEN LEAVE.
01-0205  200F  A9 00                 LDA #$0           ;IF SO, CLEAR THAT REGISTER.
01-0210  2011  95 E0                 STA CLKREG,X
01-0215  2013  CA                    DEX               ;IS IT TIME TO LEAVE?
01-0220  2014  10 ED                 BPL CKLOOP        ;NOT DONE, DO SOME MORE.
01-0225  2016  60             RETURN RTS               ;BACK TO MAIN LINE.
01-0230  2017                 ;
01-0235  2017  99             TABLE  .BYTE $99,$24,$60,$60,$15
01-0235  2018  24
01-0235  2019  60
01-0235  201A  60
01-0235  201B  15
01-0240  201C                 FINISH .END


ERRORS = 0000

END OF ASSEMBLY = 201B
:ASM
```

©

Aim User Input And Output

Joel Swank
Rockaway, OR

AIM 65 is Rockwell's entry into the SBC (Single Board Computer) market. It provides many advanced features not usually found in inexpensive systems. Among these is the capability for device independent I/O. Along with this is the capability for the user to add his own I/O device or devices. The AIM user manual devotes three pages to this feature. The information here is accurate but insufficient. Since I have recently interfaced a disk system to my AIM via the user I/O function, I have had the opportunity to investigate this feature thoroughly.

Whenever the AIM firmware receives a command requiring input or output, it calls a routine to determine which device to use. Two routines are used for this purpose: WHEREI for input and WHEREO for output. WHEREI and WHEREO prompt the user for the device to be used. The user enters a one character device code: 'P' for the printer, 'T' for tape, etc. The WHERE routine then sets a flag in memory (INFLG or OUTFLG). The selected device then becomes the active input device (AID) or the active output device (AOD). Subsequently the AIM input or output routine checks INFLG or OUTFLG and calls the appropriate input or output routine for the chosen device. When the user chooses the 'U' code an indirect jump is performed to a vector on page one. There is a vector for user input and one for user output. These vectors must be set to point to the I/O drivers for the user device. The user routine is called once from The WHERE subroutine to perform initialization for the device and once from the input or output routine for each character to be processed. So that the user routines can distinguish between a call for initialization and a call for I/O the carry flag of the processor status register is clear for a call for initialization and set for a call for I/O. Also the output routine must pull the character to be output from the stack. The user routines return to the AIM monitor via an RTS (ReTurn from Subroutine) instruction. The above is all explained in the AIM manual. There are a few other important considerations though.

The manual does not mention the fact that the user routines must preserve the contents of the registers. The input routine must preserve X and Y, and the output routine must preserve X, Y, and A. The AIM monitor provides two subroutines for saving the X and Y registers on the stack, and

```
937A           ;******************************************
937A           ;*
937A           ;*        ENTRY FOR USER OUTPUT
937A           ;*
937A           ;******************************************

937A B01D      USERO  BCS OUTCHR        ;CARRY SET FOR OUTCHAR
937C 209EEB           JSR PHXY          ;SAVE REGS
937F 20F797           JSR OPNCHK        ;IS FCB OPEN?
9382 D03B             BNE AOPN          ;YES, ERROR
9384 20E69A           JSR FILEO         ;GET FILE NAME
9387 A015             LDY #NUMSEC       ;CLEAR NUMBER OF SECTORS
9389 A900             LDA #0
938B 91F8             STA (FCBPTR),Y
938D 201298    OPENU  JSR OPEN          ;OPEN FILE
9390 B017      OPNBK  BCS ERROUT
9392 20619B    NOIERO JSR ROLIN         ;RESTORE ZERO PAGE
9395 20ACEB    UBACK  JSR PLXY          ;RESTORE REGS
9398 60        SKIPUO RTS

9399           ;   OUTPUT ONE CHARACTER TO DISK

9399 68        OUTCHR PLA               ;GET OUTPUT BYTE
939A C9FF             CMP #$FF          ;DELETE FFs
939C F0FA             BEQ SKIPUO
939E 209EEB           JSR PHXY          ;SAVE REGS
93A1 209B9B           JSR POINTO        ;INIT PTRS
93A4 20C799           JSR PUTCHR        ;SEND CHARACTER
93A7 90EC             BCC UBACK

93A9           ; ERROUT : PUT ERROR MESSAGE

93A9 48        ERROUT PHA               ;SAVE ERROR CODE
93AA 20FEE8           JSR LL            ;RESET I/O FLAGS
93AD 2094E3           JSR CKER00        ;DISPLAY ERROR
93B0 A924             LDA #'$'          ;DISPLAY ZAPDOS ERROR CODE
93B2 20BCE9           JSR OUTALL
93B5 68               PLA
93B6 2046EA           JSR NUMA          ;IN HEX
93B9 20619B    JRET   JSR ROLIN         ;RESTORE ZERO PAGE
93BC 4CA1E1           JMP COMIN         ;BACK TO MONITOR

93BF A90C      AOPN   LDA #OPFILE       ;OPEN FILE ERROR
93C1 D0E6             BNE ERROUT


93C3           ;******************************************
93C3           ;*
93C3           ;*        ENTRY FOR USER INPUT CALL
93C3           ;*
93C3           ;******************************************

93C3 B00F      USERI  BCS INCHR         ;CARRY SET FOR INPUT
93C5 209EEB           JSR PHXY          ;SAVE REGS
93C8 20279B           JSR INITB         ;INIT HI ZERO PAGE
93CB 208E9B           JSR POINTI        ;INIT PTRS
93CE 20E69A           JSR FILEO         ;GET FILE NAME
93D1 4C8D93           JMP OPENU         ;GO OPEN

93D4           ;   GET A CHARACTER FROM DISK

93D4 209EEB    INCHR  JSR PHXY          ;SAVE REGS
93D7 208E9B           JSR POINTI        ;INIT POINTERS
93DA BA               TSX
93DB BD0401           LDA $0104,X       ;WHERE CALLED FROM?
93DE C9D6             CMP #$D6          ;ASSEMBLER OPEN CALL?
93E0 F004             BEQ ASOP          ;YES, GO GET NEXT FILE
93E2 C9D4             CMP #$D4
93E4 D018             BNE GETIT         ;NO, JUST GET NEXT CHAR
93E6           ;  ASSEMBLER WANTS A NEW FILE
93E6 A920      ASOP   LDA #' '
93E8 8D3DA4           STA DIBUFF+5      ;MARK END OF FILE NAME
93EB           ;        ENTRY FOR PL65 .DFILE ROUTINE
93EB 20549B    PL65IN JSR ROLOUT        ;SAVE ZERO PAGE
93EE A900             LDA #0            ;CLEAR BUFFER INDEX
93F0 8522             STA CURCHR
93F2 A006             LDY #DRIVE
93F4 B1F8             LDA (FCBPTR),Y    ;RETRIEVE DRIVE# FROM FCB
93F6 8523             STA DRVSAV
93F8 201D98           JSR REOPEN        ;OPEN NEW FILE
93FB 4C9093           JMP OPNBK         ;FINISH

93FE 204899    GETIT  JSR GETCHR        ;A CHARACTER
9401 9092             BCC UBACK         ;NO ERROR


9403 C90E             CMP #ENDOFI       ;END OF FILE?
9405 F08E             BEQ UBACK         ;YES, CONTINUE
9407 D0A0             BNE ERROUT        ;NO, MUST BE ERROR
```

MTU

CODOS

Photo credit: GREAT GALAXY IN ANDROMEDA: Palomar Observatory, California Institute of Technology

THE MTU FLOPPY DISK CONTROLLER WITH 16K RAM GIVES YOUR AIM-65 ION DRIVE POWER!

HARDWARE

- 16K 2 PORT RAM ONBOARD WITH WRITE PROTECT
- USES THE NEC-765 DISK CONTROLLER CHIP
- ROM BOOTSTRAP LOADER SPEEDS LOADING
- DMA OPERATION ALLOWS INTERRUPTS
- SUPPORTS 8 INCH DRIVES 1 OR 2 SIDED
- MAXIMUM STORAGE IS 4 MEGABYTES
- ANALOG PLL DATA SEPERATOR

SYSTEM FEATURES

- FORMAT UTILITY LOGS OUT DEFECTIVE SECTORS
- DISK/FILE COPY WITH WILDCARD SELECTION
- SYSTEM CUSTOMIZATION UTILITY
- VISIBLE MEMORY TERMINAL DRIVER PROVIDED
- INTERCHANGE CODOS SOFTWARE AMONG KIM, SYM, AIM, PET SYSTEMS
- IN FIELD USE FOR OVER 6 MONTHS

CODOS SOFTWARE

- CODOS DISK OPERATING SOFTWARE
- 8K RAM RESIDENT ALLOWS UPGRADES
- FINDS AND LOADS 32K BYTES IN 3 SECONDS
- STARTUP FILE EXECUTES AT BOOT-UP
- COMMAND FILE EXECUTION FROM DISK
- DYNAMIC DISK STORAGE ALLOCATION
- DEVICE-INDEPENDENT I/O
- TRUE RANDOM ACCESS TO RECORD IN ONE ACCESS
- MONITOR WITH 29 BUILT-IN COMMANDS
- FULL ENGLISH ERROR MESSAGES
- FILE NAMES 12 CHARACTERS + EXTENSIONS
- FILE SIZE UP TO 1 MEGABYTE
- UP TO 247 FILES PER DISK DRIVE
- INDIVIDUAL WRITE PROTECT ON FILES
- WORKS WITH AIM EDITOR, ASSEMBLER, BASIC AND MONITOR ROMS
- SUPERVISOR CALLS AVAILABLE TO USER PROGRAM

K-1013M Hardware Manual-$10, K-1013-3M CODOS manual-$25, K-1013-3D RAM/Disk controller with CODOS-$595, Floppy drives, cables, power supply also available.

MASTERCARD & VISA accepted

WRITE OR CALL TODAY FOR OUR 48 PAGE FALL 1980 CATALOG DESCRIBING ALL MTU 6502 PRODUCTS, INCLUDING 320 BY 200 GRAPHICS, AIM GRAPHIC/TEXT PRINT SOFTWARE, BANK-SWITCHABLE RAM/ROM/I-O, AIM CARD FILE, POWER SUPPLY AND MORE!

Micro Technology Unlimited • 2806 Hillsborough St. • P.O. Box 12106 • Raleigh, N.C. 27605 • (919) 833-1458 restoring them without affecting the A register. These are PHXY and PLXY. I found need to investigate these two interesting subroutines when I tried to enter PLXY via a JMP (Jump) instruction instead of a JSR (Jump to Sub-Routine) instruction. Whenever a subroutine call is the last instruction of another subroutine I usually enter it with a JMP instruction instead of a JSR followed by a RTS. This saves one byte and normally works the same. This does not work for PHXY or PLXY because of the way they manipulate the stack. When a subroutine is entered via JSR, the return address is the last two bytes on the stack. Anything that the subroutine pushes on the stack must be pulled off before it can return properly. To get around this problem PHXY and PLXY use a third subroutine called SWSTAK (SWap the STAcK). SWSTAK swaps the 2 bytes that are 2 locations back on the stack with the 2 bytes that are 4 locations back on the stack. So, PHXY pushes the X and Y registers onto the stack and calls SWSTAK. SWSTAK swaps the X and Y bytes with the return address for PHXY and then returns to PHXY. PHXY then returns to its caller with the X and Y register values next on the stack. PLXY works just the opposite. It first calls SWSTAK to swap the saved X and Y registers with its return address. It then pulls X and Y off the stack and returns. If PLXY is entered via a JMP instead of a JSR the stack is not in the expected condition and PLXY ends up returning to the address contained in the saved X and Y registers giving unpredictable results. As long as they are used properly, PHXY and PLXY can be used by the user I/O routines to save and restore the X and Y registers.

A problem I had with the output routine is detecting the end of the output stream. Some devices such as tape and disk need to have a termination or 'close' routine that is executed after all output is complete. This routine must write the last buffer or, as in the case of disk, update the directory. The AIM output routine gives the user routine no indication of when output is complete. There is no consistant way to determine this from the data itself. I solved this problem by using one of the AIM user function keys to execute the routine to close the output file. This means that I must remember to push that key after each use of user output. This is inconvenient but the only feasible way to solve the problem.

```
0000              ; PL65 .DFILE INTERFACE
0000              ;
0000              ;   PL65 DATA AREAS
0000              ;
0000              PLDRIV =$0158         ;PL65 SAVEA DRIVE NUMBER HERE
0000              PLBUFF =$014B         ;PL65 SAVES FILE NAME HERE

0000              ;   AIM ADDRESSES

0000              DIBUFF =$A438         ;DISPLAY BUFFER
0000              PHXY   =$EB9E         ;SAVE X AND Y REGS

0000              ;   ZAPDOS ADDRESSES

0000              POINTI =$9B8E         ;POINT TO INPUT FCB
0000              PL65IN =$93EB         ;ENTRY FOR DFILE INTERFACE
0000              FCBPTR =$F8           ;POINTER TO FCB
0000              DRIVE  =6             ;OFFSET TO DRIVE # IN FCB
0000              ;
0000                     *=$112
0112              ;F3 VECTOR
0112 4CD00F              JMP DFILE      ;

0115                     *=$0FD0
0FD0              ;
0FD0 209EEB       DFILE  JSR PHXY       ;SAVE REGS
0FD3 A206                LDX #6
0FD5 BD4B01       DLUP   LDA PLBUFF,X   ;COPY FILE NAME TO DIBUFF
0FD8 9D38A4              STA DIBUFF,X
0FDB CA                  DEX
0FDC 10F7                BPL DLUP
0FDE A920                LDA #$20
0FE0 8D3EA4              STA DIBUFF+6   ;MARK END OF NAME
0FE3 208E9B              JSR POINTI     ;POINT TO INPUT FCB
0FE6 AD5801              LDA PLDRIV     ;GET PL65 DRIVE
0FE9 C9FE                CMP #$FE       ;ANY SPECIFIED?
0FEB F007                BEQ NODRV      ;NO, SKIP
0FED 6A                  ROR A
0FEE 6A                  ROR A          ;SHIFT TO 2 HI BITS
0FEF 6A                  ROR A
0FF0 A006                LDY #DRIVE
0FF2 91F8                STA (FCBPTR),Y ;AND SAVE IN FCB
0FF4 4CEB93       NODRV  JMP PL65IN     ;ENTER ZAPDOS
0FF7                     .END
0FF7                ERRORS= 0000
```

```
0000              ;  TIMER BUG VERIFICATION PROGRAM

0000              DI1024 =$A497          ;TIME X 1024
0000              RINT   =$A485          ;TIME OUT
0000              COMIN  =$E1A1          ;RETURN TO MONITOR
0000              CUREAD =$FE83          ;INPUT A CHARACTER
0000              RED2   =$E976          ;ECHO A CHAR

0000                     *=$200

0200 2083FE       READ   JSR CUREAD      ;READ A CHAR
0203 48                  PHA             ;SAVE IT
0204 A9FF         SET    LDA #$FF        ;255 X 1024
0206 8D97A4              STA DI1024      ;START TIMER
0209 2C85A4              BIT RINT        ;TIME UP ALREADY?
020C 1003                BPL LUP         ;NO, TRY AGAIN
020E 4CA1E1              JMP COMIN       ;EXIT ON TIMER ERROR
0211 2C85A4       LUP    BIT RINT        ;TIME UP?
0214 10FB                BPL LUP         ; NO, WAIT
0216 68                  PLA
0217 2076E9              JSR RED2        ;ECHO CHAR
021A 4C0002              JMP READ        ;REPEAT
021D                     .END
021D                ERRORS= 0000
```

An even greater problem is how to handle end of file on input. My disk routines detect end of file and return a condition code, but there is no way to tell the AIM routine that there is no more data. Each different AIM program detects the end from the data in its own way. The 'L' command uses a zero length record; the editor uses two successive CRs (carriage returns); the assembler uses a .END statement followed by two CRs; BASIC uses a CTL-Z. Another inconsistancy in the 'L' command causes it to try to read 5 or 6 more characters from the final zero length record than the 'D' command wrote. The user input routine must compensate for this and provide pad characters or the 'L' command will not terminate properly. When the KBD/TTY switch is in the TTY positionn, and OUTFLG is set to 'U', the AIM CRLF routine inserts an LF(Line Feed) and a null (AIM uses hex FF for a null) after each CR. On input AIM does not expect these characters to be included. The 'L' command will ignore the LFs and nulls when inputting a line of data. The editor will ignore the LFs but not the nulls. The null between the two successive CRs that end the file cause the editor to fail to recognize the end and continue to request data. To solve this problem, the nulls must be deleted from either the input stream or the output stream. I chose to delete nulls from the output stream because this saves disk storage space.

The assembler requires that the source file be read twice, once for each pass. It is designed to read the source file from tape. Before starting pass 1 it saves the name of the tape file at location $A7. Before starting pass 2 it moves the name of the tape file back to the name buffer ($A42E) and to the display buffer ($A438). It then calls the tape open routine. If the source is coming from the 'U' device it moves the file name and then makes an extra call to the user input routine. However it does not indicate to the user input routine that the call is to open a file and not to read another character. The only way I could find to detect the extra call is to test the stack to see what page the call was from. The assembler also makes an extra call to the user input routine when it encounters a .FILE statement. The .FILE statement is used to link source files together so that programs too long for the editor buffer may be assembled. When the assembler encounters a .FILE statement it moves the file name to the name buffer and display buffer and makes a call to the user input routine. Again the only way to distinguish this call from a normal input call is by checking the stack. While investigating the .FILE statement I found an undocumented feature of the assembler. The .END statement may also contain a file name. If it does, then that file is used to start pass 2 instead of the one saved at location $A7. This allows pass 2 to start with a different file than pass 1. Of what use is this feature? Sometimes it may be useful to have the first file of a program contain only label definitions (= directives). Since these statements only make entries in the symbol table and generate no code it is not necessary to read them again for pass 2. This feature can be used to save time and printer paper. You may have a file containing definitions for all the AIM subroutine addresses that you use for every program you write, but they still consume no extra space in the source file or the program listing. You can use the .END feature to omit the definition file from pass 2, but remember to set the program counter in the first file to be read in pass 2, instead of in the definitions file or assembly errors may result.

So, the user input must detect the special open calls by the assembler .FILE and .END statements, and the start of pass 2. This can only be done by checking the page of the calling routine. These calls are made from pages $D4 and $D6. It must also provide pad characters for the 'L' command. The user output routine must delete all null characters so that the editor can properly recognize the end of the file on input. As an example I have included listing 1 which is the user I/O interface for my disk system. The disk routines are not included. These routines work with all AIM commands that use the AID or AOD including 'L', 'D', the editor, the assembler, BASIC, and PL/65.

PL/65 also has a linked file feature. It uses the .TFILE statement to link tape files and the .DFILE statement to link disk files. The .TFILE statement causes the tape open routine to be executed. The .DFILE statement executes the disk open routine through the user F3 key vector. This makes the disk open interface much more straightforward than the way the assembler does it. PL/65 stores the file name at location $14B and the drive number at location $158 before calling the F3 key routine. Listing 2 is the routine needed to implement the .DFILE statement.

Another consideration is console communications. It may be necessary to request information from the user during user I/O. For example, my disk initialization routines prompt the user for the name of the file to be used. Care must be taken which AIM monitor subroutines are used at this time. Some communicate with the keyboard and display only, while others use the AID or AOD. Normally the keyboard and display are the AID and AOD so calling OUTPUT and OUTALL, for example, give the same results. After the WHERE subroutine is executed the AID or AOD has changed. If the user output initialization calls OUTALL instead of OUTPUT to display a character, it will end up calling itself. The results are unpredictable. There are two ways to solve this problem. Either be sure to call only keyboard/display routines or change the AID or AOD to the keyboard/display before attempting to communicate with the user, and restore the AID or AOD after communications is complete.

You can make your programs device independent also. To make a program device independent, you must call the WHERE subroutines before doing any I/O that is to be device independent. The data must be read or written with subroutines that use the AID or AOD. Here again care must be taken to use only AID and AOD subroutines and not the keyboard/display subroutines. Calling a wrong subroutine could cause part of the data to end up on the display or the program could hang up waiting for input from the keyboard. Subroutine LL can be used to return the keyboard and display to the AID and AOD. Even BASIC programs can change the AID and AOD by calling the WHERE subroutines with the USR function. This will cause BASIC input or output to be redirected. Of course when using some devices such as tape or disk there are close routines that must be executed to terminate output. Routine DU12 ($E511) does this for AIM tape output.

Another discovery that I made is that AIM's 6532 timer has the same bug as KIM's 6530 timer. KIM's 6530 has a bug that causes it to ignore a start command on the average of once every 256 starts. I chose to use the 6532 because it has a maximum interval of more than a quarter of a second, while the 67522 timer can only time up to 65 miliseconds. I was immediately suspicious of the 6532 when I saw that it works exactly like the 6530 and that the two chips have other similar features. So I wrote a small program to prove the prescence of this bug. Listing 3 is the result.

The 6530/32 timer, when not in use, is continuously counting down from $FF to 0 at the rate of the CPU clock signal. The bug occurs when the CPU trys to store a starting value in the counter register just as the count is passing zero. When this occurs the timer ignores the CPU. The result is an immediate time out the first time the program checks the timer. The routine in listing 3 proves that the 6532 is guilty. It first reads a character from the keyboard to get a random starting time for the timer. It then starts the timer for about a quarter second and checks to see if there is an immediate time out. If there is, it returns to the monitor. If there is not an immediate time out it enters a loop and waits for the timer to time out. Then it echos the character and repeats the sequence. If the timer is working correctly the program can never end, unless the escape key is used. It may take several hundred trys sometimes, but this program will always eventually catch the timer bug and return to the monitor.

This bug will cause occasional errors in my disk system if not circumvented. The way to circumvent this bug is to always use 2 successive stores to start the timer. If the first try fails, the second will work since the timer will not have counted back to zero by the time the second store is executed. If the first try is successful then the second will also be successful since the timer will not have counted back to zero yet. Does this bug cause any problems in the AIM? I am not sure. I looked at the AIM program listing manual and found that the AIM uses this timer in the printer routines and the tape routines. The best guess I could make is that it may cause an occasional 'PRINTER DOWN' when the printer is really up, or maybe a lost bit on tape. I would like to know if anyone has experienced any problems like these.

AIM 65 provides you with a flexible I/O system. The user I/O function gives an expandability not usually found in an SBC. If you follow the guidelines given here you should be able to implement any device via the AIM user I/O function. ©

Dungeons And Dragons Dice Simulator For The KIM-1

Myron A. Calhoun
Manhattan, KS

Last Christmas my older son received a "Dungeons and Dragons" game, but when the package was opened there were no dice included. (There were small numbered squares of cardboard for shaking and drawing out of a cup, but this seemed to slow the game considerably.) Even worse, the local hobby shop was completely sold out, so a state of near emergency existed.

(Trumpets!) Enter a KIM-1 to the rescue. The enclosed little program was quickly derived, my son was taught how to load it into the KIM-1, and the crisis was over. (Even though he has since located dice in another store, their relatively high cost and his small allowance have caused him to continue using this program!)

Summary Of Operation

Pressing the KIM-1's "0", "1", "2", "3", "4", or "5" key simulates rolling a 4-, 6-, 8-, 10-, 20-, or 100-sided die. The result is displayed as a random number in the range 1-4, 1-6, 1-8, 1-10, 1-20, or 1-100, respectively. Pressing any other key clears the display to zeroes. Holding any one of the operational keys down displays successive random numbers but too fast to read. "Random" numbers are derived from the free-running built-in timer in the KIM-1.

```
ASM6502:  6502 CROSS-ASSEMBLER USING PROPOSED I.E.E.E. STANDARD (DRAFT 11)
                    ; DUNGEONS AND DRAGONS DICE SIMULATOR FOR THE KIM-1

                    ; PRESSING THE KIM-1'S "0", "1", "2", "3", "4", OR "5" KEY
                    ; SIMULATES ROLLING A 4-, 6-, 8-, 10-, 20-, OR 100-SIDED
                    ; DIE.  THE RESULT IS DISPLAYED AS A RANDOM NUMBER IN THE
                    ; RANGE 1-4, 1-6, 1-8, 1-10, 1-20, OR 1-100, RESPECTIVELY.
                    ; HOLDING ONE OF THE ABOVE-NAMED KEYS DOWN WILL DISPLAY
                    ; SUCCESSIVE RANDOM NUMBERS BUT TOO FAST TO READ.  PRESSING
                    ; ANY OTHER KEY WILL CLEAR THE DISPLAY TO ZEROES.  "RANDOM"
                    ; NUMBERS ARE DERIVED FROM THE FREE-RUNNING BUILT-IN TIMER
                    ; IN THE KIM-1.

1704                RANDOM  EQU  H'1704    ; DEFINE MISCELLANEOUS ADDRESSES
00FB                LEFT    EQU  H'FB      ;   OF THIS-N-THAT IN THE KIM-1
00FA                MIDDLE  EQU  LEFT-1    ;   "OPERATING SYSTEM" RESERVED
00F9                RIGHT   EQU  MIDDLE-1  ;   MEMORY AREA
1F1F                SCANDS  EQU  H'1F1F
1F6A                GETKEY  EQU  H'1F6A

0000 A9 00          START   LD   .A,#0     ; CLEAR THE INITIAL DISPLAY
0002 85 F9                  ST   .A,RIGHT  ;   TO ALL ZEROES
0004 85 FA                  ST   .A,MIDDLE

0006 85 FB          NEWVALU ST   .A,LEFT   ; SET NEW VALUE (FOUND BELOW)

0008 20 1F 1F       DISPLAY CALL  SCANDS   ; "PUMP" THE DISPLAY AND ALSO
000B F0 FB                  BZ    DISPLAY  ;   SEE IF ANY KEYS ARE PRESSED
```

```
000D D8                  CLRD               ; FETCH THE BINARY
000E 20 6A 1F            CALL  GETKEY       ;   KEY VALUE FROM THE KEYBOARD.
0011 C9 15               CMP   .A,#21       ;   IF NO KEY IS BEING PRESSED
0013 F0 F3               BEQ   DISPLAY      ;   RIGHT NOW, CONTINUE DISPLAY.  IF
0015 C9 06               CMP   .A,#6        ;   A KEY LARGER THAN "5" IS PRESSED,
0017 B0 E7               BC    START        ;   THEN CLEAR THE DISPLAY AGAIN.
0019 AA                  MOV   .A,.X        ; SAVE VALID KEY (0,1,2,3,4,5)

001A AD 04 17            LD    .A,RANDOM; FETCH "RANDOM" NUMBER FROM TIMER
001D 29 7F               AND   .A,#H'7F ;   AND CONVERT TO VALUE BETWEEN 0
001F D5 3B     TRYAGIN   CMP   .A,TABLE(X); AND 3, 5, 7, 9, 19, OR 99
0021 90 06               BNC   CONVERT  ;  (DEPENDING ON VALUE IN X REGISTER)
0023 38                  SETC               ;   BY REPEATEDLY SUBTRACTING 4, 6,
0024 F5 3B               SUBC  .A,TABLE(X); 8, 10, 20, OR 100 (FROM THE TABLE)
0026 4C 1F 00            BR    TRYAGIN  ;   AND CHECK AGAIN.

0029 AA        CONVERT   MOV   .A,.X        ; NUMBER IS STILL IN BINARY FORM, SO
002A A9 00               LD    .A,#0        ;   CONVERT TO DECIMAL BY COUNTING
002C F8                  SETD               ;   THE BINARY DOWN WHILE COUNTING
002D 18        NOTYET    CLRC               ;   THE DECIMAL UP.
002E 69 01               ADDC  .A,#1        ;   (THIS IS A "CHEAP AND DIRTY"
0030 CA                  DEC   .X           ;    CONVERSION METHOD!)
0031 10 FA               BP    NOTYET

0033 85 FA               ST    .A,MIDDLE ; THEN PUT POSSIBLE 2-BYTE ANSWER
0035 A9 00               LD    .A,#0      ;   INTO ADDRESS PART OF DISPLAY
0037 2A                  ROLC  .A         ;  (LEFTMOST BYTE CAN ONLY CONTAIN
0038 4C 06 00            BR    NEWVALU    ;   THE "1" AS IN "100")

003B 04 06 08  TABLE     DATB  4,6,8,10,20,100  ; MAX VALUES FOR 6 DICE
003E 0A 14 64
0000                     END   START       ; MYRON A. CALHOUN, 29XII80
```

The program is written using the proposed IEEE (Institute of Electrical and Electronic Engineers) Microprocessor Assembly Language Standard (Draft 11) as it applies to the 6502 microprocessor. Although it differs slightly from the assembly language seen in other **COMPUTE!** articles, it should be easily understandable. According to Wayne P. Fischer, Chairman of the IEEE Computer Society's Microprocessor Assembly Language Standard Subcommittee, "The impetus for the development of this standard was the helter skelter proliferation of microprocessor mnemonic codes, the inconsistent and conflicting use of operands, the varying definition of address modes, and other annoying anomalies of the various assembly languages. The standard will transform this mishmash of languages into one that is consistent, easily understood, and easily used"(1).

The program itself is rather simple and the comments should explain it sufficiently. About the only "trick" is the method used to convert a binary number in the accumulator into a BCD number in the display. Beginning at the label CONVERT (at address H'0029), the program performs a "brute force" conversion by counting the binary value downward (after moving it to the index register) while simultaneously counting the BCD value upward in the accumulator *in decimal mode*. The value 100 (decimal) causes the CARRY bit to be set, and care must be taken to move the "1" to the display.

The TABLE values (at location H'003B) may be changed if other maximum die values are desired. The maximum length of the table is the immediate operand of the instruction at location H'0015.

The program is short enough that loading before a game takes but a few minutes. It has even gotten my boy a little interested in computers!

(1) Fisher, Wayne P., "Microprocessor Assembly Language Draft Standard", IEEE Computer Magazine, December 1979, pp. 96-109.

©

New Products

New Business Software Packages

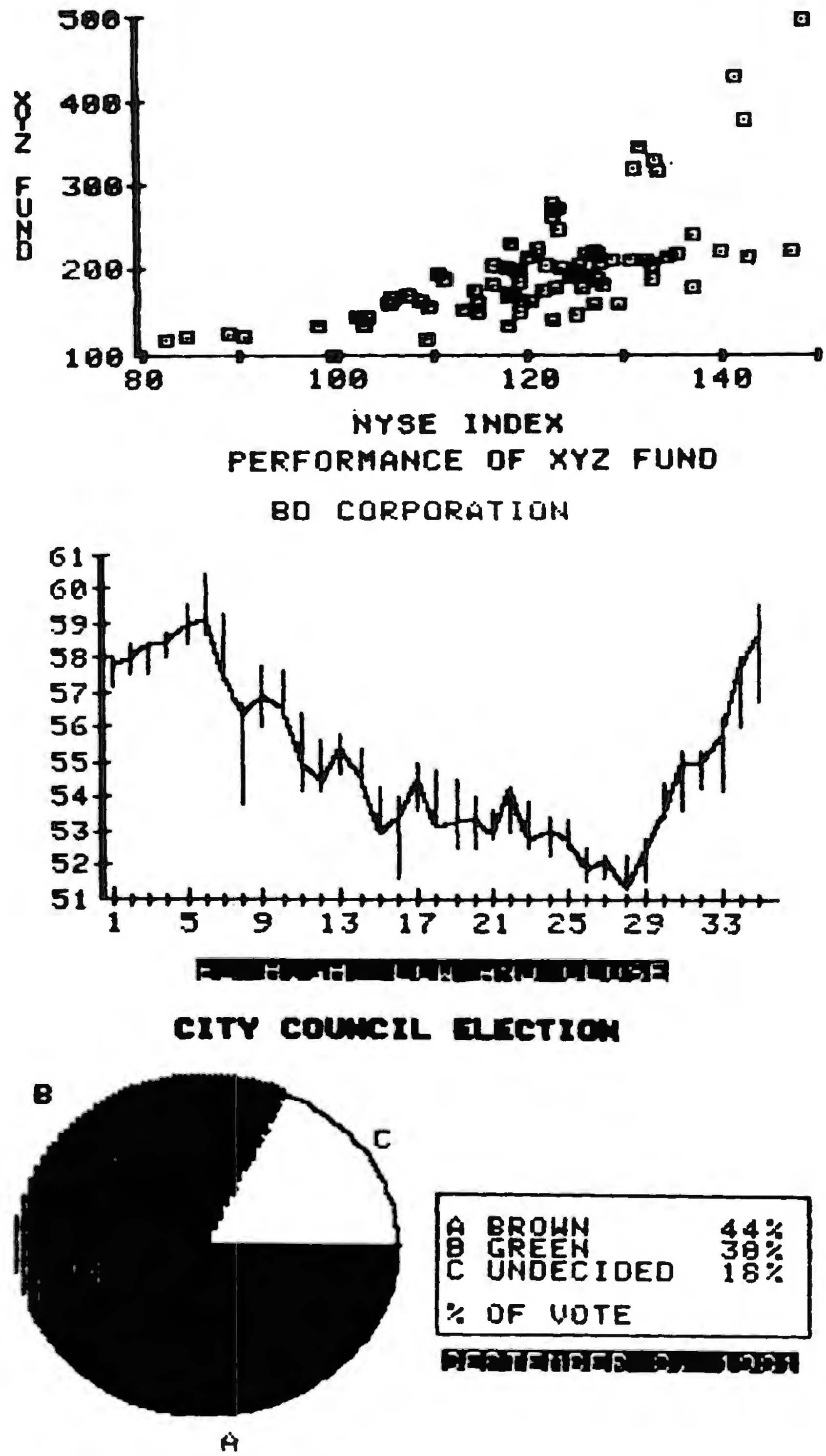
SUNNYVALE, CA — May 1, 1981 — Four new software packages for business use have been introduced by Personal Software Inc. The new software more than doubles the firm's line of generic software applications packages for major brand personal computers.

The new software products include the VisiPlot™ program, a high-resolution plotting and graphics package; the VisiDex™ program, a highly flexible and personal information system; the VisiTrend/VisiPlot™ program, a combination of VisiPlot graphics and a program for time-series manipulation, trend forecasting and descriptive statistics; and the VisiTerm™ program, which allows a personal computer to communicate with a variety of computers ranging from mainframes to micros.

Also being introduced is an enhanced version of the VisiCalc™ program, which is available for the Apple II and Apple II Plus computers. Compatible with Apple's 16-sector disk storage format, the updated VisiCalc features 17 new commands and operations which make computerized numerical model-building much easier and more powerful.

A major feature of many of the Personal Software™ programs is the ability to pass information between programs. VisiPlot and VisiTrend, for example, can share data directly with the recently updated VisiCalc program.

With VisiTerm program files from VisiCalc, VisiPlot and VisiTrend/VisiPlot, or files created with VisiDex, can be sent



Sample printout from the VisiPlot and the VisiTrend/VisiPlot programs illustrate inverse type and shading. Producible graphs include bar, area, scatter, pie, line and high-low. VisiPlot and VisiTrend/VisiPlot will graph on the Apple Silentype, Trendcom 200, Integral Data Systems' Paper Tyger and NEC Spinwriter Printers.

between computers over phone lines or through other connections. VisiTerm acts as a terminal for accessing large computer systems from a personal computer, or for allowing one personal computer to transfer informatin to another.

The four new products bring to seven the number of packages in Personal Software's line of "generic" business software. Generic software can be used for a wide variety of tasks and applications by people without programming experitse.

All four new programs operate on the Apple II and Apple II Plus personal computers with 48K bytes of memory and are 16-sector compatible, with or without Apple's Language System. VisiDex and VisiTerm require one disk drive. VisiPlot and VisiTrend/VisiPlot require two disk drives and Applesoft Basic.

Suggested retail prices are: VisiDex, $199.95; Visi-Plot, $179.95; VisiTrend/VisiPlot $259.95; VisiTerm $149.95.

CompuServe

COLUMBUS, Ohio — CompuServe Incorporated (Corporate Headquarters, 5000 Arlington Centre Boulevard, Columbus, OH 43220 614/457-8600) announced today that more than 10,000 customers now comprise the active list of subscribers to the CompuServe Information Service.

The 10,000 figure reflects the actual number of paying customers and does not include demonstration accounts, media accounts or employees.

The geographic distribution of the 10,000 customers shows high concentrations in and around New York City and the eastern seaboard, the "Silicon Valley" area of California and the areas around Los Angeles, populous areas of the South and industrial cities of the Midwest.

The CompuServe Information Service provides information and services to owners of personal computers and computer terminals in the main subject areas of news, finance, entertainment, home information, electronic mail and personal computing. Customers access CompuServe by a local telephone call in more than 260 U.S. cities.

Specific offerings include the latest news from a national news wire and major newspapers; family information including food recipes, nutrition, meal planning, home improvement and self development; financial information such as current and historical data on securities, commodities pricing and financial analysis; entertainment including multi-player games and simulations; electronic mail for message sending and retrieving; and personal computing services which include programming languages, software purchases and technical documentation.

CompuServe is an H&R Block Company.

New Floppy Disk System For PET™

CGRS Microtech, Inc. has introduced a new low cost, high performance floppy disk system for the Commodore Pet series of computers. PEDISK II is now available and can be purchased with 5¼" or 8" disk drives. A small 2½ x 5 inch disk controller board mounts inside the machine and contains the PDOS software ROM and all the disk control circuitry. An external disk drive(s) and cable complete the system.

One, two, or three drives connect to the PEDISK II controller board providing the fastest mass storage system available for the 2000, 4000, or 8000 series Commodore Pet. The single drive 5¼" system retails for $595.00 and offers 143 Kbytes of storage. The dual drive 5¼" quad density system provides 572 Kbytes of high speed storage and retails for $1195.00

PEDISK II is compatible with all Commodore disk systems and both can be simultaneously used in the same machine. With appropriate software, files can be transferred from one disk system to the other.

One of the unusual features of the 8" PEDISK II System is its ability to exchange diskettes with large computers. Data files can be entered and edited on a small Pet/PEDISK system and the diskettes can be transferred to a larger 3740 format computer for more extensive processing or storage. Data files, mailing lists, etc. that are available on large business computers can also be read and processed by remote distributed PEDISK II Systems.

The PDOS II software package links the resident BASIC language to the disk system by adding a new repertoire of disk commands. !OPEN, !CLOSE, !INPUT, and !PRINT provide the basis of a powerful file handling package. Sequential, Random, Indexed, or Relative file types are accomodated in the standard PDOS software package. !LOAD, !SAVE, !LIST, and !RUN commands provide complete program control from BASIC, PDOS II software also offers a full DOS-mode of operation for all disk diagnostic and utility functions. Diskette formatting, backup copying, diagnostics,

New Product releases are selected from submissions for reasons of timeliness, available space, and general interest to our readers. We regret that we are unable to select all new product submissions for publication. Readers should be aware that we present here some edited version of material submitted by vendors and are unable to vouch for its accuracy at time of publication.

and file reorganization capabilities are all available.

The PEDISK II floppy disk system is available in several versions. Either 5¼" or 8" disk drives are available. The system can accomodate 40 or 80 track drives in the 5¼" version. Standard versions are available with either a single or dual drives. For more information, contact your local dealer or CGRS MICROTECH, Inc. P.O. Box 102, Langhorne, PA 10947 215-757-0284.

INSTANT ROM!

Greenwich Instruments Limited announce INSTANT ROM, a new programming aid for microcomputers and microcontrollers.

INSTANT ROM is 2k bytes or 4k bytes of CMOS memory with battery back-up which plugs into any standard ROM/EPROM socket, or into the ROM expansion sockets of the PET and similar microcomputers. It can be treated as RAM, and machine language programs can be easily written into it.

The difference is that the program stays when the power is switched off. Therefore, it behaves exactly like ROM.

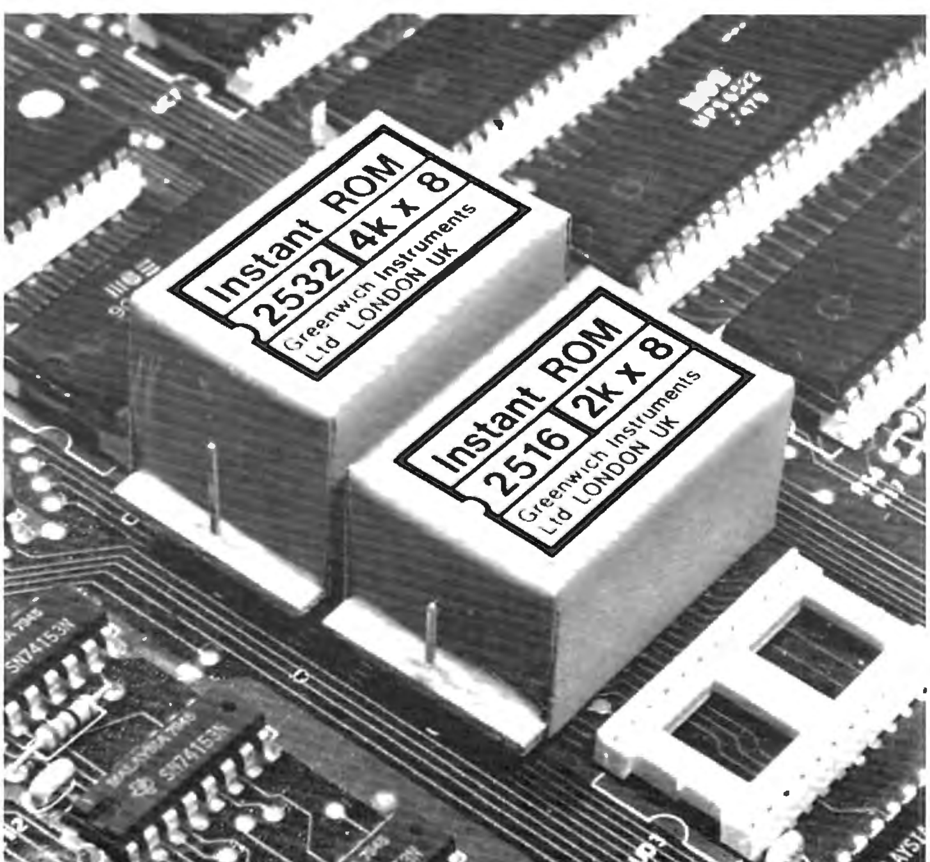
Programs can be edited or altered immediately, without any erasure problems, yet the program is permanent.

When used with personal computers (particularly the PET), it is now possible to write security and utility programs quickly, run and debug them, and finally consign the program to EPROM. But since battery life is typically 6 years, it is possible to ship a system with INSTANT ROM and supply the ROM later.

"Custom font" character generators for microcomputers and printers can also be quickly produced with INSTANT ROM; the results can be observed and put into service immediately.

Prices

2k devices: 2516 (for use with PET) and 2716: £39 inc. postage
4k devices: 2532 (for use with PET) and 2732: £56 inc. postage
VAT extra 15%



Commodore International Introduces "Micro-Mainframe" Computer

NORRISTOWN, PA — Commodore International Ltd. (NYSE: CBU) has announced the introduction of the "Micro-Mainframe", a new generation computer that combines the power and languages available on mainframe systems with the low cost of microcomputers.

Applications developed on the Micro-Mainframe can be up-loaded to a mainframe system, and executed without modification.

The computer is based on the standard Commodore Business Machines Model 8032 microc computer, featuring an integrated green phosphor 12 inch (80 x 25) display, 73-key typewriter style keyboard with standard upper/lower case, numeric keypad, and full cursor control.

The Micro-Mainframe is a pseudo 16 bit 6809 based system with 36K ROM, 96K user RAM and 2K screen RAM (134K total) that supports all current CBM peripherals except the C2N cassette recorder.

The communication facility supports the standard RS232 C interface with speeds up to 9600 baud.

Files are stored in true ASCII format for communication and compatibility with mainframe systems.

The Micro-Mainframe allows the generation, testing, editing, and debugging of program source files in the interpretive mode.

These files can then be executed on the Micro-Mainframe or transmitted and executed on the mainframe system utilizing the same language interpreters.

An extensive software package for the Micro-Mainframe has been developed by Waterloo

Computing Systems Limited to meet the requirements of the University of Waterloo, Waterloo, Ontario. This portable software is particularly suited to microcomputers, but identical versions are available on medium and large scale systems. Thus a user is not limited by the capacity of the micro; the identical program will run without modification on many of the largest and fastest systems available.

The package consists of interpreters for various languages, an editor, an operating system (supervisor) and an assembly language development system.

The four language interpreters include Waterloo microBASIC, Waterloo microPascal, Waterloo microFORTRAN and Waterloo microAPL. COBOL is under development by Waterloo Computing Systems Limited. These language interpreters have been designed specifically for teaching computer programming. Their design emphasizes good error diagnosis and debugging capabilities which are useful in educational and other program development environments.

Waterloo microBASIC includes ANS Minimal BASIC, with certain minor exceptions, and several extensions such as structured programming control, long names for variables and other program entities, character-string manipulation, callable procedures and multi-line functions, sequential and relative file capabilities, integer arithmetic, debugging facilities, and convenient program entry and editing facilities.

Waterloo microPascal is an extensive implementation of Pascal, corresponding very closely to draft proposals being produced by the international standards Organization (ISO) Pascal Committee. The ISO draft language is a refinement of the language originally defined by Wirth, varying only in minor aspects.

This implementation includes

sophisticated features such as text file support, pointer variables, and multi-dimensioned arrays. A significant feature of Waterloo microPascal is its powerful interactive debugging facility.

Waterloo microFORTRAN is a special dialect designed for teaching purposes. It has many of the characteristics and much of the flavor of normal FORTRAN, but varies significantly from established standards for that language.

This language processor has many of the important characteristics of the WATFIV-S compiler, which is widely used on IBM computers, plus some features from the new FORTRAN-77 definition.

Examples of language features supported are FORMAT, subroutines and functions, multi-dimensioned arrays, extended character-string manipulation, structured programming control and file input/output. In addition, the interpreter provides a powerful interactive debugging facility.

Waterloo microAPL is intended to be a complete and faithful implementation of the IBM/ACM standard for APL with respect to the syntax and semantics of APL statements, operators and primitive functions, input/output forms, and defined functions.

System commands, system variables and system functions are those consistent with a single-user environment. There are no significant design limitations on the rank or shape of arrays or name length. The shared variable processor is omitted. Extensions include system functions supporting files of APL arrays. APL equivalents of BASIC features PEEK, POKE and SYS are included.

A text editor, known as Waterloo microEDITOR, is included which is suitable for creating and maintaining both program and source data files. It is a traditional line-oriented text editor with powerful text searching and substitution commands including global change. Full-screen support and special function keys allow text to be altered, inserted and deleted on the screen without entering commands. Facilities for repeating and editing previously issued commands further enhance the useability of this editor.

Disk-oriented Assembler and Linker programs, the Waterloo 6809 Assembler and Linker, are included which support development of general purpose Motorola 6809 machine-language programs.

The Assembler supports syntax and directives for Motorola 6809 assembly language and includes powerful macro capabilities. In addition, the Assembler supports pseudo opcodes for structured programming control, long names (labels) for meaningful identification of program segments and data, and the ability to include definitions from separate files. The Assembler produces both a listing and relocatable object file.

The Linker allows the combination of an arbitrary number of relocatable object files to produce an absolute loadable and executable program file. since it is disk-oriented, the Linker is capable of building programs which are larger than the RAM work space available. The Linker supports building of programs in segments or banks for operation in bank-switched RAM memory, as well as supporting building of programs for operation in normal RAM memory.

The Waterloo microSUPERVISOR is an operating system designed for single-user microcomputer environments. It includes a Monitor, Library and Serial Line Communication support.

The Monitor program supports loading of Linker-produced program files into bank-switched RAM memory or normal RAM memory. The Monitor also provides facilities which are useful for debugging machine-language programs. These include commands to display and alter RAM memory and 6809 microprocessor registers, utilizing full screen features for ease of use. In addition, a Monitor command permits disassembly of Motorola 6809 microprocessor instructions into assembly-language mnemonics.

A library of functions and procedures is supplied for general use by other programs included in the Package. The Library includes support functions for input/output operations to the keyboard, screen and peripheral devices. Other elements of the Library provide floating-point arithmetic, fundamental trigonometric functions, and several general purpose utility functions.

A Serial Line Setup program is included which provides for the selection of programmable characteristics, such as baud rate. The program includes support for establishing communication with a host computer, through a serial line, for accessing the host's files and peripheral devices.

Reference manuals, textbooks and instructors' guides are available for each software component of the system.

The Micro-Mainframe can operate as a stand-alone system, with the wide variety of software available, as a mainframe system development tool with the available languages and up-load/off-load capabilities, and in the educational environment for training in languages and system design.

Deliveries are scheduled for late 1981.

PET To Mainframe Interface

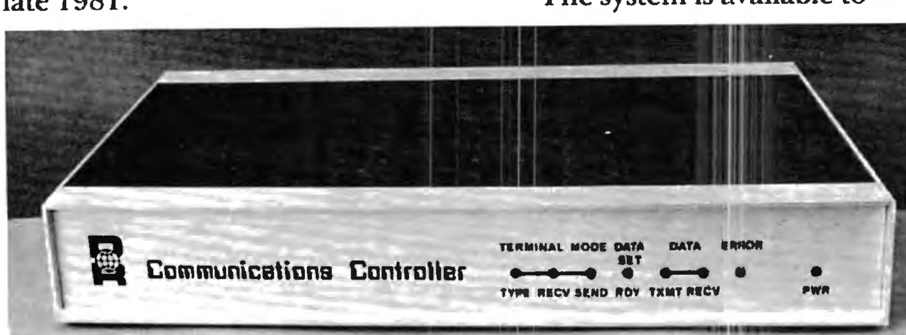
A combined hardware/software package that enables PET Microcomputers to be connected to Mainframes and act as full facility interactive terminals is available from Davidson-Richards Limited.

The system comprises a communications control box and software that controls both it and the PET when running in the communications mode. The box, which connects directly to the PET user port, is microprocessor controlled. Memory is provided to house the protocol handling routines, which are both resident in ROM and downloaded from the PET, and also act as a receive and transmit buffer.

Output from the control box is via a RS232C interface, enabling communication with the host Mainframe by direct line or telephone modem. Efficient handling techniques allow several terminals (whether PET based or not) to share a single line depending upon its capacity and quality.

In use, the PET emulates a full facility video terminal with addressable cursor, format control and protected fields. Facilities for screen hard copy print and mainframe controlled loading and running of PET Basic application programs are provided. The status protocol enables the PET to be switched from the mainframe for local program working, and returned via a 'break-in' message. Disk drive and printer operation is unaffected.

The system is available to



support ICL C01 C02 protocols, and emulate ICL 7181, 7501, 7502 and IBM 3780 and 3270 workstations. Data transmission speeds in the range 1200 to 9600 baud are available as selectable options, but other speeds can be obtained.

The cost for the interface and software for one protocol is £995.00. A complete workstation including a PET, disk drive, printer, interface and software is under £3000.00.

For more information contact:

Davidson-Richards Limited
14 Duffield Road
Derby DE1 3BB England
(0332) 366803

Price Reductions, New Software Announced For ATARI Computers

SUNNYVALE, CA — May 4, 1981 — Significant price reductions on selected computer, memory and peripheral products, new consumer and technical software, and pricing for products introduced earlier were announced today by the Computer Division of Atari, Incorporated, headquartered here.

Timed for announcement at the National Computer Conference in Chicago, these moves are "aimed at giving our customers an even better price/performance package, while adding programming aids that will appeal to those who develop software for the markets we want to address," Roger H. Badertscher, president of the division, said. Atari also announced special software and accessory packages and pricing for particular market segments, designed for sale through mass merchandisers and dealers.

The changes in suggested retail prices:

– The ATARI 400™ Personal Computer System with 16K of random access memory (RAM) has been reduced in price 37 percent, from $630.00 to $399.00, making it the most powerful personal computer in its price range. The ATARI 400 computer system with 8K RAM has been discontinued.

– Prices of the company's 8K and 16K RAM Memory Modules™ for the ATARI 800™ are now $49.95 and $99.95, respectively, reductions of 60 and 50 percent.

– The ATARI 820™ 40-column impact printer has been reduced 33 percent to $299.95, from $449.95.

– New software includes:

Personal Financial Management System™, a database-oriented system designed to help the user plan and analyze a home budget. For the ATARI 800™ computer, $74.95.

Dow Jones Investment Evaluator™ connnects an ATARI 800 computer to the Dow Jones Informationn Service to keep track of portfolio value and current financial news, $99.95.

Microsoft BASIC, a programming language, which, with the Atari computer player missile graphics system, gives serious programmers the ability to create faster, more powerful and more precise programs. For the ATARI 800 computer, $89.95.

ATARI SORCIM Macro Assembler and Program-Text Editor, both for the ATARI 800 computer, are for use by advanced programmers. Price and availability have not been determined.

Pricing

The previously introduced ATARI Accountant™ software package for small business or professional use has been priced at $1,499.85. Each of its three components, which are available separately, is priced at $499.95. They include systems for accounts receivable, general accounting and inventory control. The ATARI Word Processor has been priced at $149.95. Both of these products are for the ATARI 800 computer. Pilot, a programming language for use in creating interactive educational programs, has been priced at $89.95.

Prices of the following Atari entertainment cartridges have been reduced from $39.95 to $29.95: Basketbell, 3-D Tic-Tac-Toe and Video Easel™; Star Raiders™ has been reduced to $39.95 from $59.95. Asteroids™ and Missile Command™ games have been priced at $39.95.

All prices mentioned are manufacturer's suggested retail prices in the U.S. only. ©

Advertiser's Index

The Editor's Feedback:

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**______________
☐ **Don't yet have one...**
During the next year I expect to buy:
☐ **computer** ☐ **printer**
☐ **disk drive** ☐ **other peripherals**

Content:
Best Article This Issue (page #, title)

____

Other suggestions

5 6 7 8 9 10 11 12

The 6502 Resource Magazine
COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**______________
☐ **Don't yet have one...**
During the next year I expect to buy:
☐ **computer** ☐ **printer**
☐ **disk drive** ☐ **other peripherals**
☐ Please enter my 1 year (12 issue) subscription to COMPUTE!
☐ New subscription ☐ Renewal subscription
☐ $20.00 U.S. subscription
☐ $25.00 Canada and international surface mail. Payment in U.S. funds, drawn on U.S. bank or money order
☐ Bill me (U.S. only) $1.00 billing fee
Name, Address: ______________________________

Charge my: ☐ Visa ☐ MC
Number______________________________ Expires________/________/
5 6 7 8 9 10 11 12

Place
Stamp
Here

COMPUTE.
Post Office Box 5406
Greensboro, NC 27403

Place
Stamp
Here

COMPUTE!
Post Office Box 5406
Greensboro, NC 27403

Place
Stamp
Here

COMPUTE!
Post Office Box 5406
Greensboro, NC 27403

The Great American Solution Machine.

Meet Commodore. The business computer that's providing solutions for more than 100,000 people all over the world. Built by one of the pioneers in office machines. With a reputation for quality that can only come from vertical integration of the total manufacturing process. Commodore builds, not assembles.

Compare Commodore's word and data processing capabilities with computers costing twice or even three times as much. You'll see why so many small businesses are turning to Commodore for solutions to problems as varied as these:

☐ A car leasing company's customers were terminating too early for account profitability. Solution: A 16K Commodore. It analyzes cash flow on over 1200 accounts, identifies those for early penalties, and even writes up lease contracts. Commodore paid for itself within weeks.

☐ A law firm needed a high quality, easy-to-use, affordable word processing system. Solution: Commodore plus its WordPro software package. At a $6,000 savings.

☐ A gasoline retailer needed to inventory, order and set prices; determine Federal and state income taxes; and comply with Federal pricing and allocation regulations. All done daily, weekly, monthly and yearly. Solution: Commodore. It keeps his business on track—and Uncle Sam off his back.

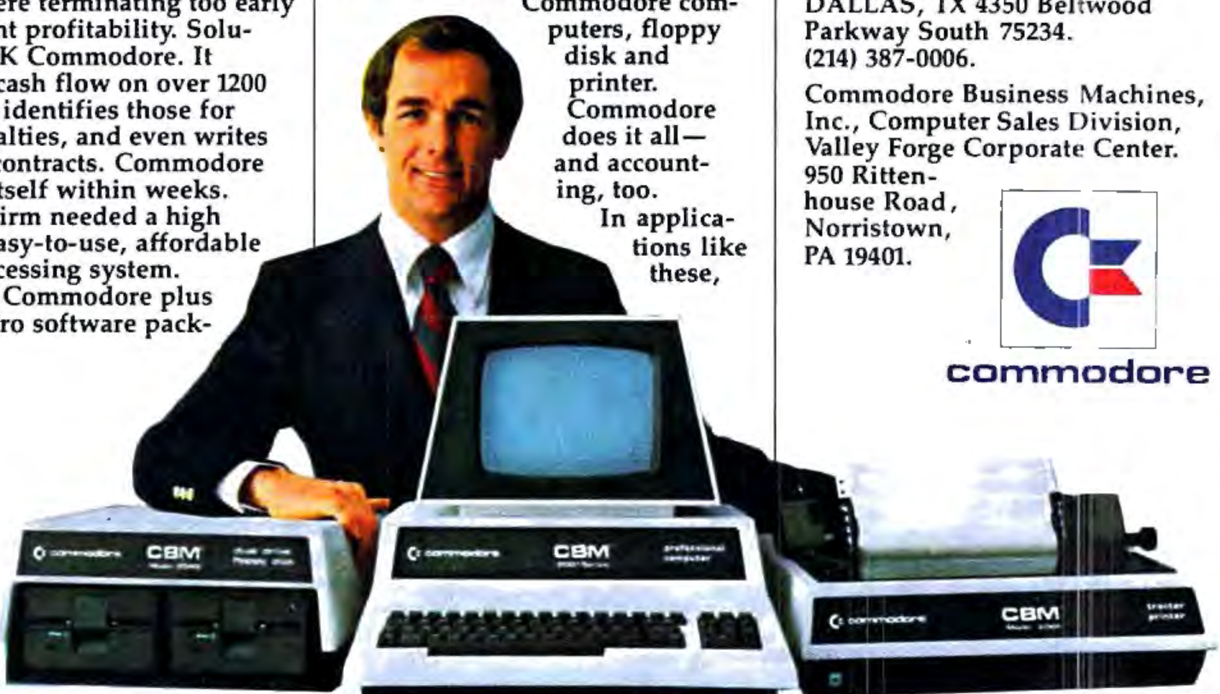
☐ A paint and wallpaper store had to inventory over 600 expensive wallpaper lines for profitability, monitor distributor sales, set and track salesmen's goals, and help the customer select the right size, pattern and quantity. Solution: Two 32K Commodore computers, floppy disk and printer. Commodore does it all—and accounting, too.

In applications like these, and many more, Commodore solves the problems that stand in the way of increased profitability. Commodore can provide the solution in your Great American business, too. Find out more by calling or writing any of Commodore's District Sales Offices.

COSTA MESA, CA 2955 N. Airway Avenue 92626. (714) 979-6307.
SANTA CLARA, CA 3330 Scott Boulevard 95051. (408) 727-1130.
DECATUR, GA 5360 Snapfinger Woods Drive 30035. (404) 987-3311.
BENSENVILLE, IL 790 Maple Lane 60106. (312) 595-5990.
NORRISTOWN, PA 950 Rittenhouse Road 19401. (215) 666-7950.
DALLAS, TX 4350 Beltwood Parkway South 75234. (214) 387-0006.

Commodore Business Machines, Inc., Computer Sales Division, Valley Forge Corporate Center. 950 Rittenhouse Road, Norristown, PA 19401.

commodore



COMMODORE