More Machine Language For Beginners

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

VIC-20 News And Updates

COMPUTE!

$2.50
October, 1981
Issue 17
Vol. 3, No. 10
63379

The Journal For Progressive Computing™

Extending Atari Player Missile Graphics

Train Your PET/CBM To Run VIC-20 Programs

Budgeting On The Apple Computer

An Atari Program Library

High Resolution Bar Graphs For The PET/CBM

10
0 74470 63379

Mountain Computer CPS MultiFunction Card™

The Only Interface Card You Need!

Connecting a Parallel Printer?

Epson®- Centronics®- IDS Paper Tiger®- CPS handles all these printers and others with on-board intelligence to provide paging and other features found on no other card.

Connecting a Serial Printer?

Diablo®- Qume®- NEC - TI 800 Series®- CPS handles these printers and others with standard RS-232 interface providing selection of baud rates, handshakes, paging, and more.

Connecting a Modem or Terminal?

Hayes Smartmodem®- Novation CAT®- M & R Pennywhistle®- CPS handles these and others with full/half duplex operation, baud rate selection, and even a transparent terminal mode which includes a dual mode feature that permits printing of text to parallel printers while 'on-line' eliminating the need for special terminal software—and more.

Connecting with the Time?

The on-board calendar/clock provides real time and date information including day of week, day, month, year, hours, minutes, and seconds for any application requiring a time stamp—battery backed-up for over two years!

Connecting with The Source?

Used with a modem, CPS provides the connection to information utilities, such as The Source,® Dow Jones, and others. Additionally, CPS provides the connection to big-time electronic mail with programs such as Micro Courier® and Micro Telegram,® and other data transfer programs.

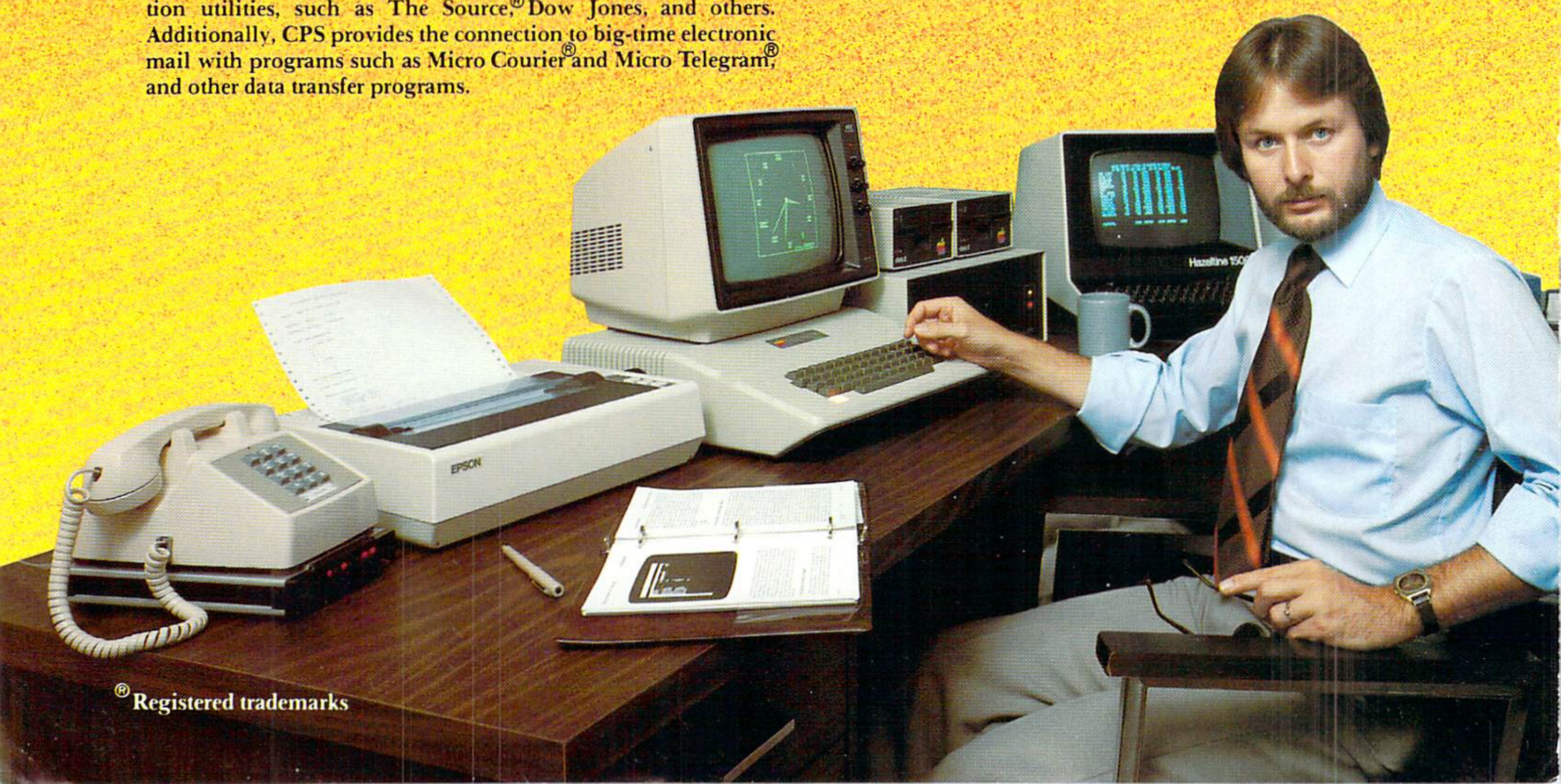
Connect with Easy Use!

CPS has no switches to set! All functions on the card are set from a user program. Menu driven screens set up your choice of all functions on CPS and store them on-board in CMOS RAM—battery backed-up (including the clock) for over two years! To change parameters, run the set-up program again—or use special commands from your keyboard. Furthermore, most existing software programs are immediately usable with CPS. *Phantom Slot Capability* permits assignment of CPS' functions to your software's pre-defined slots.

Connect with Your Apple® Dealer

Drop by your Apple dealer and see how the CPS MultiFunction Card provides the most comprehensive capabilities for RS-232C serial interface, parallel output, and real-time calendar/clock of any card available today—*all on one card*—at one low price—competitive with *any one* of the three or more single function cards that it replaces.

Mountain Computer
INCORPORATED
300 El Pueblo Road, Scotts Valley, CA 95066
(408) 438-6650 TWX: 910 598-4504



® Registered trademarks

Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

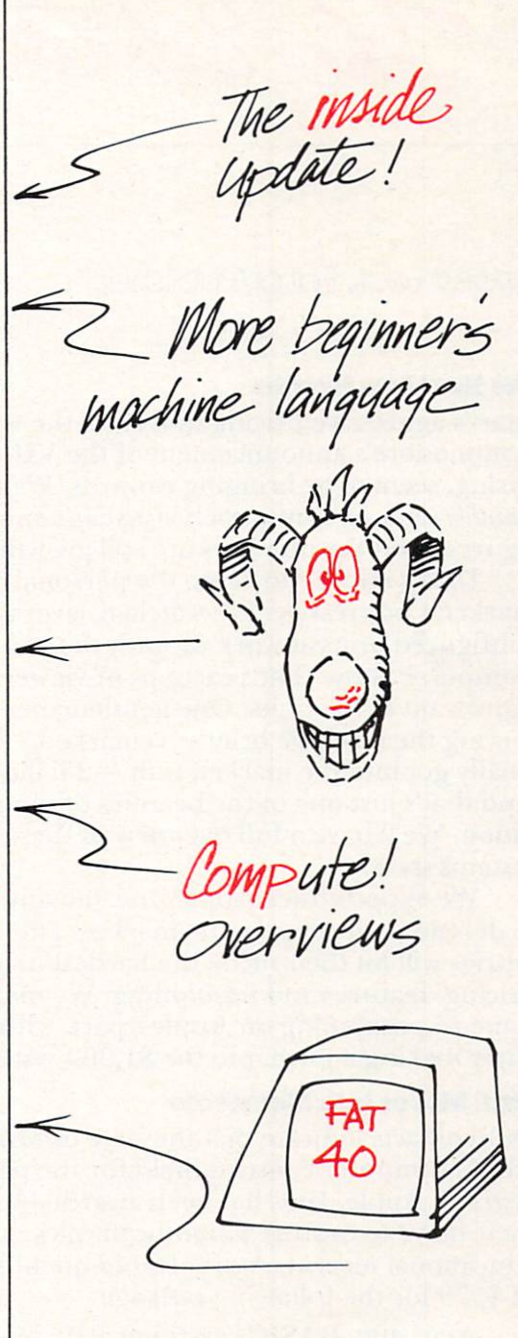
SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT
CONSUMER PRODUCTS

Microsoft Consumer Products, 400 108th Ave. N.E., Bellevue, WA 98004. (206) 454-1315

Table of Contents

October, 1981, Vol. 3, No 9



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 625 Fulton Street, Greensboro, NC 27403.

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.

The Editor's notes...

Robert Lock, Editor/Publisher

The Next Few Months

Atari's aggressive pricing moves, in the wake of Commodore's announcement of the VIC-20 last spring, seem to be bringing rewards. We hear that *monthly* sales now approach last year's *annual* sales figures. And the numbers are still growing.

IBM's initial entry into the personal computer market is impressive. We watched several fully configured units at work recently at the Midwest Computer Show. The reactions of viewers were as significant as the units. One gentleman, after observing the machine briefly, remarked, "Well they finally got into the market, huh — I'll buy one." And that's just one of the beauties of name recognition. We'll have a full overview of these new IBM systems soon.

We expect to see Apple, Inc. moving quickly to defend their place in the market. The IBM entries will hit their niche the hardest in terms of pricing, features and positioning. We might expect some re-positioning on Apple's part, with one new entry making a push into the $1,000 system area.

Atari Moves Into Minnesota

Our contacts indicate that the state of Minnesota school computer contract, held for the past three years by Apple, Inc., has been awarded to Atari. In their move to capture a significant piece of the educational market, Atari offered quantity prices of $579 for the following package:

> Atari 400, BASIC cartridge, 810 disk drive, joystick, and 13" black and white TV.

A competitive package, to say the least! Several thousand systems to start with, our sources say, and similar arrangements are being set up around the country. One of the dealer-level beefs we heard when Apple, Inc. was moving directly into the high volume sales markets was that dealers were being left out.

Atari, to their lasting benefit we're sure, will be selling through individual dealers in each town. The dealers will then carry through providing service, ongoing support and additional software, and peripherals as required on a local basis. We applaud this significant support of the dealer network.

10,000,000 Personal Computers By The End of '86?

A conservative estimate if you believe some marketing plans. The systems selling for less than $400.00 may hit that point even sooner. We expect 1982 alone to see delivery of well over 1,000,000 core units: Atari 400's, VIC-20's, and Radio Shack Color Computers.

Clarifying The Rights Of Authors

When you sell a manuscript to **COMPUTE!**, or one of our other publications, we purchase all rights to your manuscript, including accompanying software. The software rights are non-exclusive however, in that we freely give permission on request to original authors, authorizing the sale or distribution of their software. It is understood that such sale or distribution is non-exclusive, and subject to agreement by the author. We also retain the right to sell and distribute the software on a non-exclusive basis, subject to royalty and contractual agreement by the author.

In Their Continuing Quest To Respond To The US Market, Commodore Brings In Their Best

Commodore has been voicing a commitment to

Coming In November:

—Commodore's Super-Pet Revealed — The history and evolution of the new computer.

—An Atari Program For Writing Programs

—A Data Base Management System For The Atari

—And Much More!

Robert C. Lock, Publisher/Editor
Kathleen Martinek, Publication Assistant
Richard Mansfield, Assistant Editor
Charles Brannon, Editorial Assistant

Associate Editors
Jim Butterfield, Toronto, Canada
Harvey Herman, Greensboro, NC

Contributing Editors
Charles Brannon, P.O. Box 6104, Greensboro, NC 27405
Marvin DeJong, Dept. of Mathematics -Physics, The School of the Ozarks Pt. Lookout, MD 65726
Eric Rehnke, 1067 Jadestone Lane, Corona, CA 91720
David Thornburg, P.O. Box 1317, Los Altos, CA 94022
Bill Wilkinson, Optimized Systems Software, 10379-C Lansdale, Ave. Cupertino, CA 95014
Gene Zumchak, 1700 Niagara St., Buffalo, NY 14207

Georgia Papadopoulos, Art Director/ Production Manager
Terry Cash, Typesetting/Production Assistant
Harry Blair, Director, Advertising and Promotion
Joretta Klepfer, Manager, North American Retail Sales
Alice S. Wolfe, Manager, Foreign Retail Sales
Bonnie Valentino, Circulation Assistant
Sonja Whitesell, Office Assistant
Dai Rees, Shipping Department

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)

TOLL FREE
Subscription
Order Line
800-345-8112
In PA 800-662-2444

Canadian Retail Dealers should contact:
Micron Distributing
409 Queen Street West
Toronto, Ontario M5V 2A5
416-361-0690

Advertising Sales

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

Phoebe Thompson and Associates
101 Church Street
Suite 13
Los Gatos, CA 95030
408-354-5553
PHOEBE THOMPSON

If you're in the East, we're now represented by **The Gittleman Company.** You can reach them through the following offices:

New England, New York State:
The Gittleman Company
Statler Office Building
Suite 582
20 Providence Street
Boston, MA 02110
617-451-0822
JOAN DONAHUE

The Gittleman Company
Summit Office Centre
7266 Summit Avenue
Fort Washington, PA 19034
215-646-5700
DOUG JOHNSON

New York City Metro Area, Mid-Atlantic and Southeastern States:
Local Numbers:
New York 212-567-6717
Atlanta 404-523-1252

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

COMPUTE! Home Office Contacts
Advertising Sales Manager, HARRY BLAIR
Advertising/Production Coordinator, ALICE S. WOLFE

Address all advertising materials to:
625 Fulton Street
Greensboro, NC 27503 USA

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

Telephone: 919-275-9809

strengthening their US marketing operations for some time now. We're hopeful that a recent personnel move will translate that theory into practice. Kit Spencer, the former marketing head for Commodore UK, has come over to head up US marketing operations. While in England, Kit built an organization which, at one point, held 70% of the market share. Should be interesting to see how he does here if the powers that still be let him have his go at it. You'll find a candid, exciting interview with Kit in our November issue.

The Single Board Computer Gazette — A Decision And Announcement

The December issue of **COMPUTE!** will be the last with an SBC Gazette. The gradual reorientation of **COMPUTE!**, and the changing needs of our readers, contributed to this decision. While the Gazette will go away, interest won't — we'll still have occasional and timely articles relevant to all readers. And we'll still have the continuing contributions of Marvin DeJong, Gene Zumchak's column *Nuts And Volts*, and more.

Creative Computing Acquires Computers And Programming Magazine

In the push for biggest, *Creative Computing* has made a dynamic move to leap past McGraw Hill's *BYTE* magazine as the largest circulation magazine in the industry. *Creative* bought *Computers and Programming* magazine (remember *Elementary Electronics* magazine? — that's it with a name change and an audience repositioning). *Creative*'s blending the C&P audience into their own subscriber base, ending up with a projected circulation in excess of *BYTE*'s 200,000 +.

It's an interesting marriage of reader populations and we're curious to see how it all sorts out.

Telecommunications And COMPUTE!

One of my pet frustrations has been the amount of editorial paper shuffling we end up doing around here. All of our typesetting is now done inhouse on Mergenthaler equipment. By early spring we expect to be set up editorially to serve as "store and forward" hosts to our columnists. They'll be able to call our machines (PET, Atari, Apple, etc.) and transmit columns to us directly. We'll be able to edit on-line, and then load editorial material directly into our typesetting unit, thereby saving millions of keystrokes, and two entire copy proofing steps.

It's nice to think we'll actually get to the point where we can save ourselves tremendous amounts of time using the technology we're all surrounded by!

Happy Birthday — COMPUTE! Grows On

With this issue our press run has increased to 40,000. Two years ago, we were in the midst of anxiously trying to gather our first 400 subscribers. Our first issue went out to fewer than 40 dealers, world-wide. Now, two years later, this issue goes to readers in more than 50 countries, and a dealer/newsstand network just short of a thousand.

Our growth has been marked by constant compounding due, in large part, to you, our readers. Our recently completed reader survey included a question designed to help us identify where we find you. Or, better stated, where you find us. While the answers showed us our advertising works, and our new subscribers from retail outlets are important, the second largest source of new subscribers was you — the existing readers. Well over 30% of our new subscribers find **COMPUTE!** via a friend's recommendation. Facing page 40, you'll find our direct mail cards. Give one to a friend and sign them up. Thanks.

California Here We Come (And Michigan, New York, Boise...)

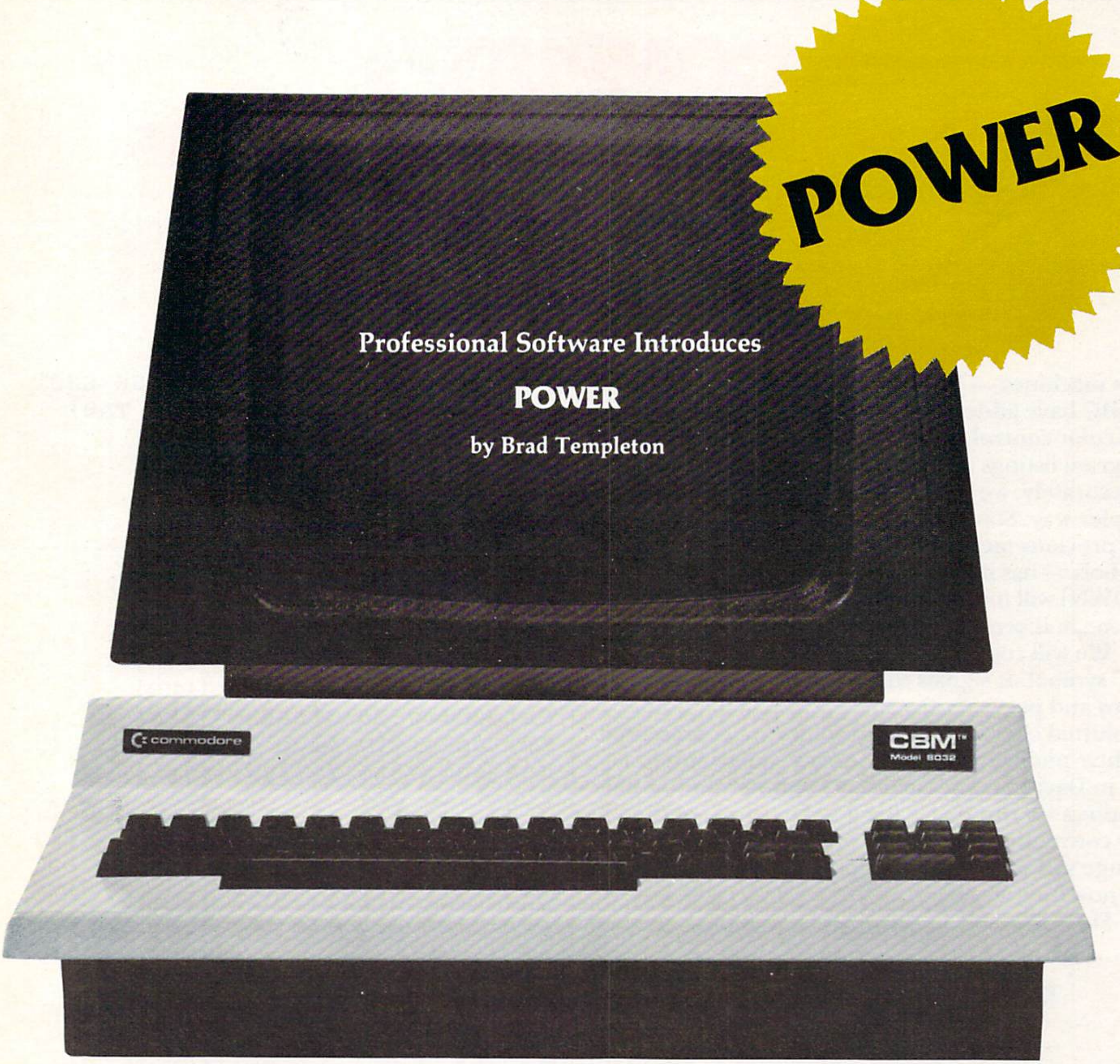
In our efforts to bring production and delivery to earlier dates, we're gradually making changes that should bring subscriber delivery to a par with store delivery. One big change we expect to implement by the December issue will involve all of you West Coast readers. Currently the mail is taking three weeks or more in some cases to get to you. As far as we can tell, there's absolutely nothing we can do with the US Postal Service to improve that delivery time. You're "Zone 8" from us, and that's that. Beginning with the December issue, we'll truck your magazines across country and mail them in California.

You should see substantially improved delivery time and be able to enjoy your **COMPUTE!** that much sooner.

COMPUTE! Books Update

Our first two books, the Atari and the PET/CBM book, were delayed during our production revamping. They are now scheduled for completion and shipping in October. Those of you who've ordered the book, and waited patiently (or not so patiently) should be assured your orders will be shipped first. In this case, we'll send them out first class mail. Sorry for the delay.

— RCL



ADD POWER TO YOUR COMMODORE COMPUTER

$89.95

POWER produces a dramatic improvement in the ease of editing BASIC on Commodore's computers. POWER is a programmer's utility package (in a 4K ROM) that contains a series of new commands and utilities which are added to the Screen Editor and the BASIC Interpreter. Designed for the CBM BASIC user, POWER contains special editing, programming, and software debugging tools not found in any other microcomputer BASIC. POWER is easy to use and is sold complete with a full operator's manual written by Jim Butterfield.

POWER's special keyboard 'instant action' features and additional commands make up for, and go beyond the limitations of CBM BASIC. The added features include auto line numbering, tracing, single stepping through programs, line renumbering, and definition of keys as BASIC keywords. POWER even includes new "stick-on" keycap labels. The cursor movement keys are enhanced by the addition of auto-repeat and text searching functions are added to help ease program modification. Cursor UP and cursor DOWN produce **previous** and next lines of source code. COMPLETE BASIC program listings in memory can be displayed on the screen and scrolled in either direction. POWER is a must for every serious CBM user.

Call us today, for the name of the Professional Software dealer nearest you.

Professional Software Inc.
166 Crescent Road
Needham, MA 02194
Tel: (617) 444-5224 Telex #951579

COMPUTE!'s New Listing Conventions For CBM

New machines — VIC and "FAT-40" — and 4.0 BASIC have added a host of new editing functions and color control codes. To make **COMPUTE!**'s program listings as easy as possible for you to type in accurately, we will list CBM programs in a new, simpler way. Starting in this issue, you will see that our previous method — reconstructed cursor symbols — has been replaced by bracketed words. [DOWN] will mean the cursor-down key. [3 LEFT] will mean three cursor-lefts, and so on.

We will continue to split program lines with the ˉ symbol. It signals that the line is continued below and prevents any spaces from being hidden. All shifted characters and graphics are represented by their underlined non-graphics equivalent. Line 110 in David Swaim's article, "High Resolution Bar Graphics for the PET," is a good example of the new conventions. We hope you will agree that this change will simplify your typing of programs. Let us know how you feel.

Here is a table of the new conventions:

Key To COMPUTE!'s CBM Listings

All Machines

Clear Screen	{CLEAR}
Home Cursor	{HOME}
Cursor Up	{UP}
Cursor Down	{DOWN}
Cursor Right	{RIGHT}
Cursor Left	{LEFT}
Insert Character	{INST}
Delete Character	{DEL}
Reverse field on	{RVS}
Reverse field off	{OFF}

CBM 8032/"FAT 40"

Set Window Top	{SET TOP}
Set Window Bottom	{SET BOT}
Scroll Up	{SCR UP}
Scroll Down	{SCR DOWN}
Insert Line	{INST LINE}
Delete Line	{DEL LINE}
Erase to Beginning	{ERASE BEG}
Erase to End	{ERASE END}
Toggle TAB	{TGL TAB}
TAB	{TAB}
ESCape key	{ESC}

VIC 20 Color Computer

Set color to Black	{BLK}
Set color to White	{WHT}
Set color to Red	{RED}
Set color to Cyan	{CYN}
Set color to Purple	{PUR}
Set color to Green	{GRN}
Set color to Blue	{BLU}
Set color to Yellow	{YEL}
Function One	{F1}
Function Two	{F2}
Function Three	{F3}
Function Four	{F4}
Function Five	{F5}
Function Six	{F6}
Function Seven	{F7}
Function Eight	{F8}
Any Non IMplemented Function	{NIM}

©

Computers And Society

David D. Thornburg
Innovision
Los Altos, CA

Speculations On The Appropriateness Of Technology ...

Many years ago the Dick Tracy comic strip featured a character named Diet Smith who had invented a magnetic levitation process which allowed (among other things) interplanetary travel. His slogan was "The nation that controls magnetism controls the Universe."

The idea that new technologies can generate social change on a large scale is more common in science fiction than it is in reality. Today we seem less concerned with technology giving us control of the universe than with the question of whether certain technologies are appropriate for any use whatsoever. If we are to believe authors like Alvin Toffler and Frank Herbert, the personal computer is soon to become an indispensable part of our lives. According to these futurists, everyone will be using these marvelous machines soon.

And yet, if the personal computer is to play such an important role in our lives, it isn't at all clear just how this is going to happen. In fact, the personal computer world seems to be entering a period of some confusion at this time — a confusion born less of technology than of the question of just what the appropriate applications for this technology are.

Before 1979, most of the personal scale computer systems in the world were in the hands of hobbyists — people who eagerly became the explorers of this new field, mapping uncharted territory and reporting their findings to the more timid. These people knew exactly what they were doing, and they were in control of their computers from the first time they turned them on.

Next came people wanting to use these machines for business applications. For these people there was a large gap between the user's expectations and the limited tasks these machines appeared able to perform. The conversion of the personal computers into a useful tool didn't happen overnight. Software pioneers, working out of bedrooms, garages, and warehouse offices generated thousands of programs for these customers in the hope that the perfect application would be found. However well intentioned the effort, until recently, the personal computer simply wasn't an adequate tool. When used for inventory control, for example, the memory capacity of most micros is too small for all but the tiniest company; and most tiny companies who know enough to want to use a computer also know that they won't be tiny companies forever.

And so, after a period where many programmers appeared to be taking the role of Eddington's monkeys, thrashing at thousands of keyboards and hoping that one of them would produce a work of Shakespeare, two magnificently appropriate applications for business users were developed. These were the electronic spread sheet (of which VisiCalc is the most popular), and word processing. Each of these applications was new for the "data processing" environment normally associated with business computers — and each of them suited the size and capabilities of the personal computer very well. Each of these applications was new for the "data processing" environment normally associated with business computers — and each of them suited the size and capabilities of the personal computer very well.

Once these applications became accepted, two things happened to the industry. Business users started buying personal computers by the hundreds of thousands, and the traditional Fortune 500 computer companies made their decision to enter the fray. It took the hardware pioneers, Commodore, Apple, Tandy, Atari, and others, to qualify the market for the new entrants — notably Xerox and IBM. In the space of a year, the personal computer went from being an inappropriate tool to being an essential tool for many thousands of businesses.

And now we see several computer manufacturers making their plunge into the newest (and largest) marketing frontier of all — the mass consumer market. This is the most dangerous market of all to enter with a new technology, since many fine product concepts have lain like so many rusted Edsels on the path to the marketplace. And yet several brave manufacturers have declared their intentions to be successful in a market which has yet to define its principal application. If you doubt the seriousness of this effort, note that, last year, computers were generally sold only through computer and office product stores. This year, computers can be found in most large department stores and catalog showrooms — places from which they are being purchased in record numbers.

Even as these machines are being sold, one must ask if their purchasers realize that they too are pioneers — that the appropriate application for personal computers in the home is yet to be defined. To explore the appropriateness of the personal computer in the home, let's look at two factors: applications and ease of use.

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

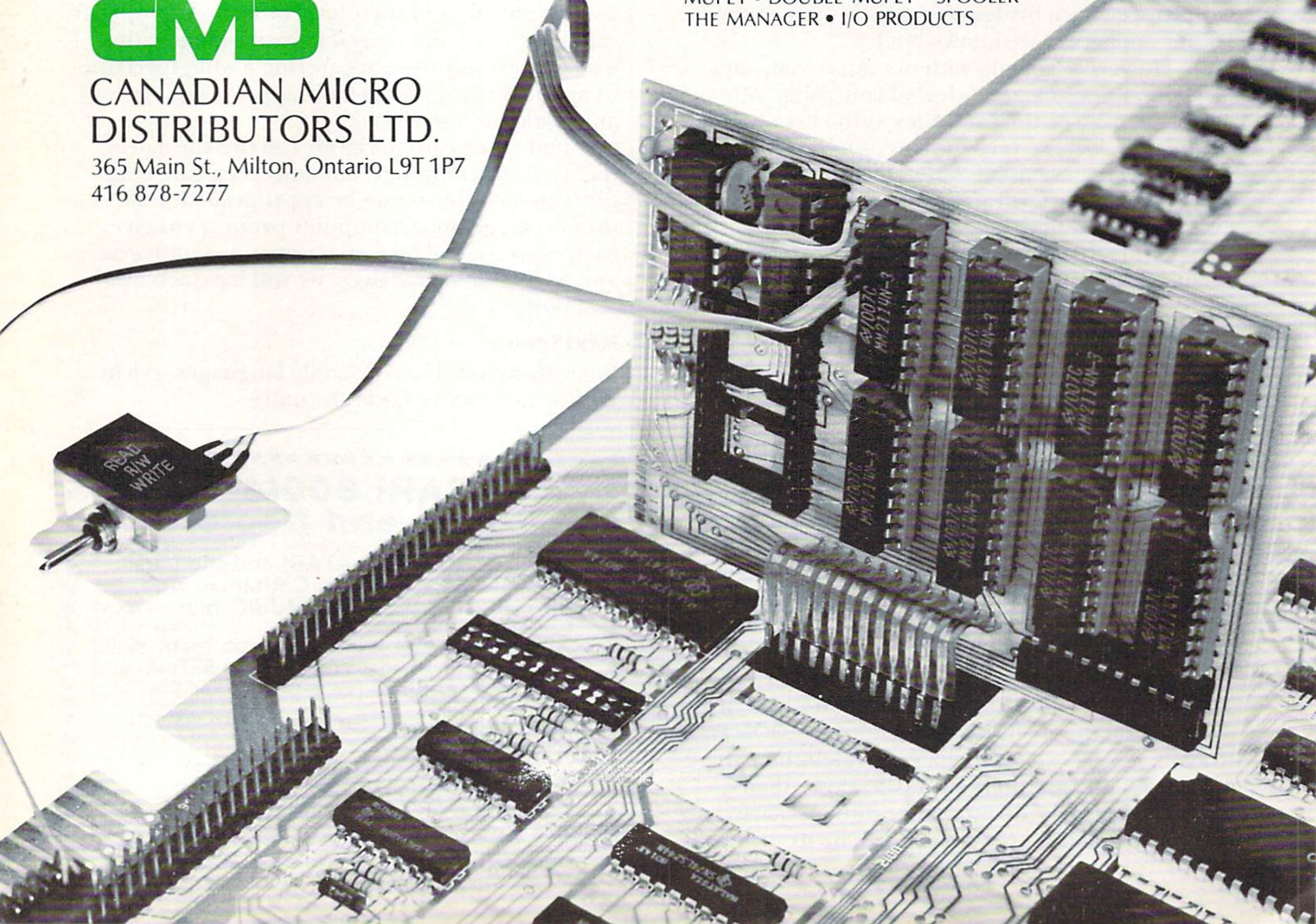
FROM

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



If we examine the appliances which are already in homes today, we can separate them into roughly three categories — utilitarian, communication, and entertainment. Utilitarian appliances include clothes washers, stoves, refrigerators, and other appliances which are used to maintain and serve the utility needs of the household. The only pure communication device most people have in their homes is the telephone, although one could put CB radio in this category as well. The remaining appliances — television, radio, stereo system, *etc.*, fall under the entertainment category. These entertainment devices are overwhelmingly communications oriented. The vast majority of this equipment is designed to receive broadcast material, or to play pre-recorded material. It wasn't until the video game that a non-communications oriented entertainment appliance entered the arena.

And now we must ask where the personal computer fits into the home. Some people envision the home computer as the wonder device which serves many functions simultaneously — controlling the lawn sprinklers, receiving the latest news from the UPI wire, and challenging its owners to a fast game of Space Evaders. If this vision is correct, then the personal computer will become the home appliance which bridges the gaps between all the other appliances we know about.

However, as I talk with potential computer owners, I detect a great deal of confusion. Most of these people see the computer as the next home appliance, but are very unclear as to how this appliance will serve them. Many people seem to think that the major useful application for these machines is to serve as a high quality video game.

While there is an awareness of the educational value of having a computer around the house, there are not an overwhelming number of well designed educational programs on the market. Nonetheless, advertising which makes parents feel guilty for not getting a computer for their children has probably increased personal computer sales to families.

Communications is another legitimate application, but many potential users are not yet ready to use the computer as a replacement for the post office or the morning paper. And home financial management sounds like a great ideal until people realize the tremendous amount of labor associated with maintaining a data base.

Unless people can see some direct benefit from their purchase, they will either defer their purchase, or end up buying a computer which lies unused on the shelf. It may be hard for **COMPUTE!** readers to accept, but I will wager that there are a whole lot of computers sitting on closet floors, unused, because the purchaser didn't realize that this machine was not yet appropriate for the tasks he or she had in mind.

Even if the magic programs were found tomorrow, computers will not be sold by the millions unless people think they are easy to use. To a consumer who is used to pushing a button on a dishwasher, or to turning two dials on a television set, a full blown computer keyboard (with keys labeled CTRL and ESC) can be quite intimidating. Also, any computer which says READY when first turned on certainly isn't ready for the average user who is used to nothing more complex than a record changer.

To be useful in the true mass market, the computer must display a list of meaningful options when it is first turned on. The user must be given as much guidance as possible. Fortunately, most of the personal computers on the market today are capable of being programmed so that a user-friendly interface program is loaded automatically when the system is powered up.

The situation is far from hopeless. Even with all its defects, the personal computer *is* being purchased by consumers who want to be on the leading edge of this technology. Those of us who understand these machines need to listen to these new pioneers when they tell us what they want. We need to be responsive to the suggestions of all users, regardless of their level of technical sophistication. Most of all we need to experiment with a wide variety of software covering a wide spectrum of applications until the truly appropriate "home" application is discovered.

Only then can we rightly call this the age of the personal computer, and only then can we say that this technology can be appropriate for all users. The personal computer promises to give each of us control of our informational universe — and when that day arrives, we will have achieved real power!

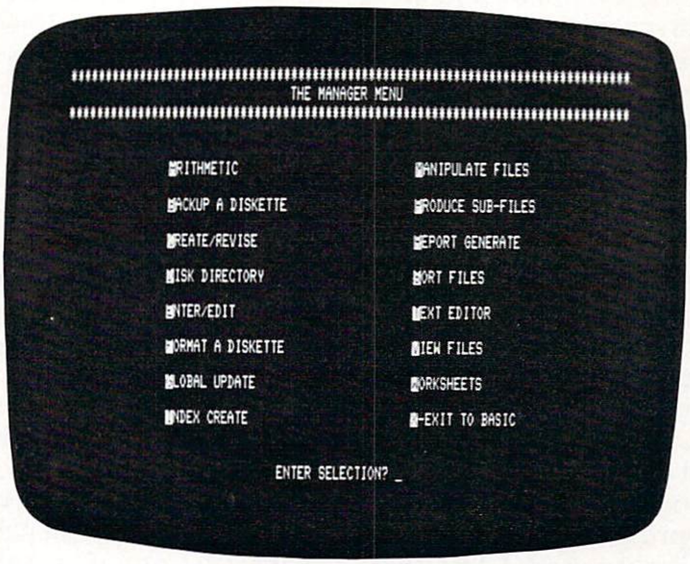
Next Time ...

We will explore User-Friendly languages — why BASIC may not be basic any more. ©

THE MANAGER

The first truly user-friendly Database Management System available at reasonable cost.

This suite of programs is ideally suited for both the businessman and programmer, for use with the CBM 8032.



For the Business User

- Uses Menu Options – no programming experience needed.
- Lets you enter data in the form you wish, then lets you recall it using any search criteria.
- Performs predefined calculations on the record in realtime as record is displayed on the screen.
- Reports can be produced using any search criteria and/or arithmetic functions.
- Useful applications can be developed quickly.

Typical Applications include –

- Inventory Control
- Mailing Lists
- Accounting systems
- Personnel
- Costing
- Gathering test data
- Budgeting
- Scheduling
- Examples of use included on disk supplied.

As Programmers Tool

- Uses standard PET ASCII files.
- Software interface is in Basic and available to the programmer.
- No special disk formatting so that word processing or other programs can be stored on the same disk.
- No ROM Based Security thus no need to open CPU.
- Fast 'n' key Sort/Merge included.
- Full realtime intra & inter record arithmetic performed on the screen as record is displayed.
- Professional software support including unique security available.

TRY IT!
IF YOU ARE
NOT SATISFIED WITHIN 30 DAYS
WE WILL RETURN YOUR MONEY

$250.00

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

Reader's Feedback

"With reference to "The World Computer," poor Mr. DeJong. He is not the first and certainly not the last person to have trouble loading a program. His real problem, however, is not software, but his perception of the entire microcomputer industry. The implication that this industry has moved out of its infancy and is maturing, or worse, has matured, is not supported by real world conditions. For example, I sold my first APPLE computer in the fall of 1976, a mere 5 years ago, a time when people didn't know APPLES from oranges (pardon the pun). BASIC was the only language, software was virtually non-existent, peripherals were few and far between, and computer literacy was a term that hadn't been coined yet. Since then great strides have been made in the field of languages, software, hardware and education, but, in spite of this progress, the industry is only approaching its infancy. Second generation computers like the ATARI, the VIC-20, the APPLE III and others from such giants as IBM and XEROX are proof of this. All of these units have expanded on the foundations laid by APPLE, Radio Shack, and others.

The driving force behind this creative work is not universality, but diversity. The opportunity to come up with a better idea, the ability to design and market a more powerful machine or program or peripheral supplies the incentive to change. The prospects of a "world computer" as described by Mr. DeJong are horrifying! First is the problem of design, with so many hands in the pie it would either end up an electronic eunuch or have so many bells, gongs and whistles that it would be frightfully expensive or be a nightmare to operate (or both).

How about an operating language? Do you use BASIC, COBOL, CP/M, SMALL TALK, FORTRAN, FORTH, or should they all be dumped in favor of a totally new language? If the latter, what happens to all the existing software? What happens to the "world computer" if the design is improved? Should any such changes be 100% compatible with older units, thus draining innovations of their potential? Lastly, why even have an APPLE, ATARI, COMMODORE, TRS-80, or any such multitude of manufacturers if, for all intents and purposes, all the machines are cast in the same mold? We could have one large, inefficient firm cranking out "generic" computers, complete with label-less white boxes. The possibility is simply too monstrous to consider seriously!" Vern L. Mastel

---

"Concerning Mr. Thornburg's "rebuttal" to my article, it was never my intention to sell "drill" type programs to anyone. The whole point of the article was to allow the teacher to teach. And then use the computer to help him with his job. My remarks concerning games were asides. They were not specifically germane to the major premise of the article, except to this extent: in practice, games too often surplant meaningful work with computers in the public schools. This is even true in colleges.

Games certainly have their place in learning. Any teacher knows that. And I would suspect that this is especially true in the home environment, where the number of computers available for the task is not an overriding consideration. My high school, however, has the use of only one computer for all of mathematics and science. We simply cannot afford games. And unless a particular school is especially affluent, *neither can the average public school.*

Now about drill: here again, I was — shall we say — misunderstood. It is my contention that if basic skills are acquired as a result of computer games, it is precisely because the program had in some way made drill palatable. These are the only games in Computer Aided Instruction, (CAI,) that to me are of any consequence. These are precisely the games Mr. Thornburg decries. The "Star Treks and "Othellos" that are played and toyed with in school are the domain of the very fine students. These latter have no need of CAI. I teach them computer programming.

The notion that drill "turns a student off" was also not my idea. Quite the reverse. I was quoting the pedagogues; that pervasive philosophic tilt in education generally attributed to John Dewey, which has produced what promises to be the least educated generation in the history of this country. It is exemplified by what is sometimes called "The Sesame Street Syndrome," the notion that something of consequence — aside from various vulgarities — can be derived from a school atmosphere of fun and games.

I teach Title I classes in remedial mathematics. If there is one common thread that unites all these students — aside from their inability to do basic arithmetic — it is their almost universal lack of personal discipline. Nobody has ever required them to do anything of consequence, certainly not in academics. There is absolutely nothing theoretical about their needs. They need to be "told," first. Then they need to acquire skills; much of it by-rote type skills like multiplication and addition facts. We provide those skills. We do it with drill, individualized and scored, with the computer helping to make it all possible. Standing over it all is the most important ingredient of all — the teacher, flesh and blood type, with all the attributes of patience, concern and even empathy for his students, that brought him to the profession." Alfred D'Attore

Ask The Readers

Robert Lock, Richard Mansfield and Readers

We are grateful for the many readers who have sent questions and answers in to this column. Please keep on letting us know what your problems are and helping us solve the questions raised by other readers. Here is this month's exchange:

" 'My Apple II: It does not compute!' wrote to complain about excessive TV interference from his Apple II computer. Recently I've dealt with a similar problem — keeping an exceedingly noisy laser from interfering with my computer! Radio-frequency interference (RFI) can be a pesky problem, but the following measures may help the situation:

If you use your TV set for a computer display, be sure you disconnect the antenna while using the computer. This is especially important in apartment buildings, where many apartments share one antenna system! You can disconnect the antenna wires with a screwdriver, or insert an antenna/computer switch between the computer, antenna, and TV set. These switches are available at many electronics and TV repair shops.

Insert an "EMI filter" in series with the AC power cord. These filters are available from several manufacturers, notably Corcom. *Interference frequently travels to other apartments through the power lines. Early Apples, with their switching power supplies, may be especially susceptible to this problem.*

Try moving the computer to a different spot in the apartment, plugging it into a different AC outlet.

If desperate, you might try wrapping printer and disk cables, and perhaps the computer itself, in aluminum foil (!). Ground the foil to the computer chassis, the AC ground, or a cold water pipe.

Find out whether you're really the guilty party. I live in an apartment with a PET, and OSI, and a Motorola computer. The OSI runs with its case open much of the time, and the Motorola computer has no case. Neither machine causes perceptible TV interference." Mark Bernstein

---

Our thanks to INSIGHT: ATARI columnist Bill Wilkinson for the following information. McBee Barbour asked (**COMPUTE!** #15) about any AMWAY distributor software which would work on the Apple II. We can suggest that one source of such a package is OnLine Microcenters, 5636 Blackstone, Fresno, CA 93710. The Atari package is currently available and an Apple version is planned if demand warrants it.

"I purchased an old OSI system consisting of a 500 CUP board (revision A), a model 430B I/O board, 12K of memory and a Teletype model ASR-33 terminal with tape reader. The system is also currently cassette based.

In 1977, OSI sold a video board (model 440) and two video support ROM's (65V prom monitor and 500VB prom) for use with a Black-and-White monitor. These items are no longer available from OSI.

Since I purchased the above for next to nothing purely as a learning tool in conjunction with my studies, I wish to establish a video terminal for minimum cost.

I would greatly appreciate hearing from any of you who may have had a similar situation." Frank Koelbl

---

"I have been using a Commodore Pet Computer now for some time together with a 3040 Floppy Disk Unit and was wondering whether you or one of your associates, e.g. Jim Butterfield, can answer a few questions for me. alternatively, a source of such information would be useful.

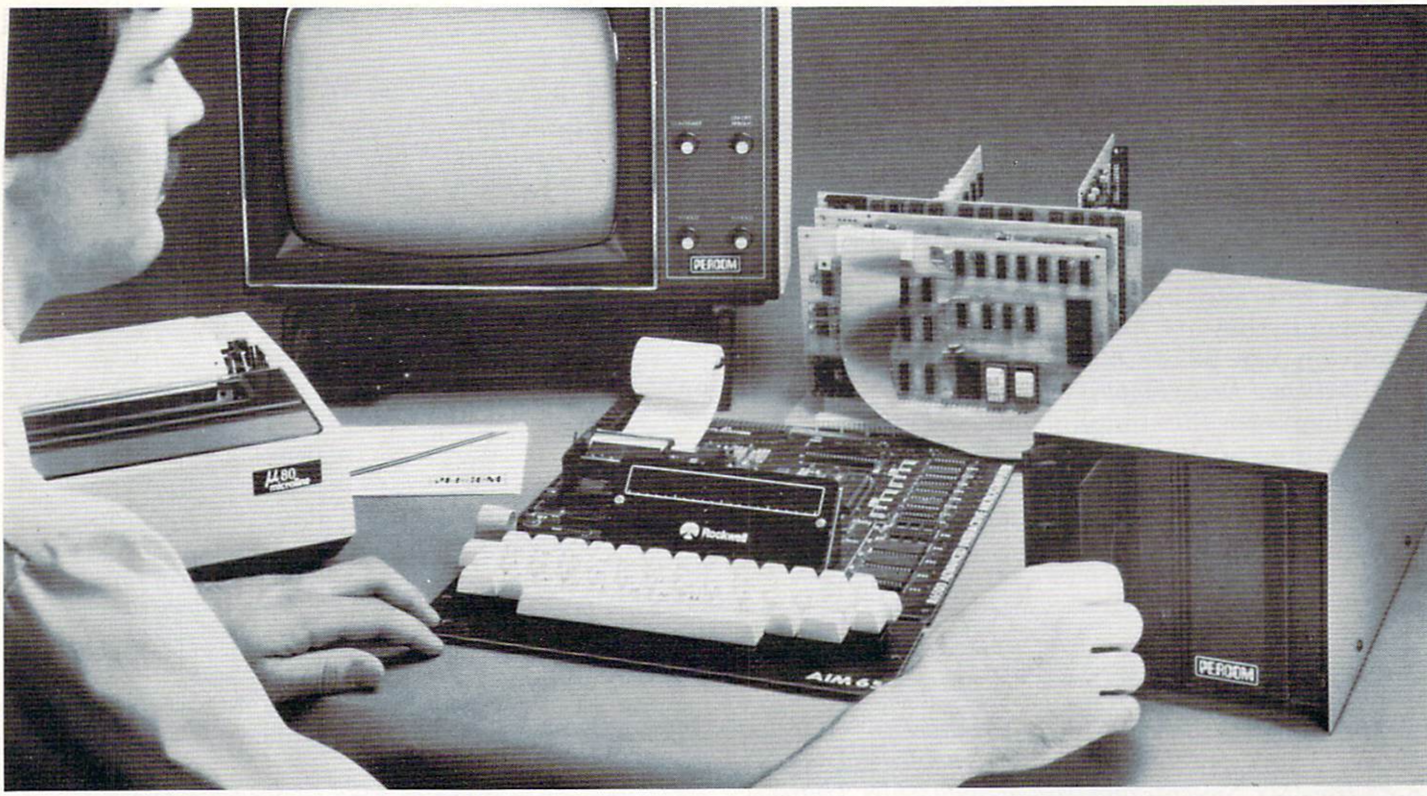
My questions concern the Disk Drive for which Commodore appears to publish very little other than the Handbook.

I think a Manual which gave some (if not all!) of the subroutines would be a useful item.

The point of most interest to me concerns the individual blocks on a formatted disk. It is possible to change the ID on a particular track & sector to be different to the main ID shown in the directory? How can one go about this? Once changed, can this be altered back again?

My other point concerns the 'U' commands. We know from the manual that 'U1' will 'READ' & 'U2' will 'WRITE,' but what other commands are there involving 'U' and what do they do which is useful. Are the routines available only by MACHINE-CODE access?

May I say I find **COMPUTE!** *the best magazine of its kind on the market, but I have great dificulty in obtaining it. I would be grateful if you would tell me if it is directly obtainable from you on a regular basis, or where the best*


PERCOM
Rockwell
PERCOM

source in London would be.

I hope you do not find my questions impertinent — they are not intended to be." M. J. Band

COMPUTE! expects to publish a program shortly which will permit easy viewing and changing of any byte on a disk including the directory, BAM, etc.). In reference to your question about the "U" commands, they are "user-defined" and described briefly on page 53 of Commodore's *User's Manual for CBM Dual Drive Floppys* (Part Number 320899).

Thanks for the compliment. **COMPUTE!** subscriptions are available from: Circulation Department, **COMPUTE!** Magazine, P.O. Box 5406, Greensboro, NC 27403 USA. The cost is $25 surface mail anywhere in the world, $38 airmail to Europe.

"Regarding Edward Sweeney's letter to you (in your August issue), VOTRAX has made every attempt to insure that every customer has received sufficient information to use Type-'N-Talk. We received a letter dated 7-6-81 from Mr. Sweeney and tried to contact him. Unfortunately, our attempt to reach him by phone was futile since his phone was found to be disconnected. We immediately responded to his letter by forwarding to him both the basic program lines necessary to use T-N-T with an Atari, as well as the necessary cable configuration (since Atari uses a non-standard cable). We subsequently received a letter dated 7-26-81 from Mr. Sweeney thanking us for fulfilling his requirements.

In order to eliminate connection problems that may arise, VOTRAX is and has been offering RS232C compatible cables for many of the personal computers including: Apple, Atari, Heath, Ohio Scientific, and Radio Shack models I, II, III and the color computer. As far as driving software is concerned, access to the RS232 Port (and thereby the T-N-T) is accomplished (in most systems) by using a simple print statement. In other words, if you want T-N-T to say "Hello", you simply print "Hello". Additionally, many of the major software houses are currently either converting existing programs or developing new programs to utilize this new dimension now available to virtually any computer.

If any of your readers have purchased T-N-T, or are contemplating purchasing T-N-T, technical questions should be directed to me at (313) 588-0341."

Douglas A. Porath
Applications Engineer, Votrax.

Our thanks to VOTRAX for the prompt clarification above. Here's a helpful answer from a reader:

Re: Edward Sweeny,

"Your Atari 800 is quite compatible with your Votrax Type'N Talk speech synthesizer. If the TNT is properly connected to a speaker, it should say, "SYSTEM READY," when you turn it on.

You also must have the Atari 850 interface and its initialization software. Set the baud rate switch #6 down on the TNT and plug TNT into your interface port #1.

If you have already run into trouble, contact Votrax. If you are sure that TNT and 850 are OK up to this point, the following little Atari Basic routine will get you started.

```
1 OPEN #2,8,0, "R1:"
2 XIO 36,#2,12,0, "R1:"
3 XIO 34,#2,48,0, "R1:"
4 PRINT #2; "TALK 2ME"
5 STOP                 "
```

Jerry White

"I have been told that there is a computer device which duplicates the tossing of the coins process associated with The I Ching, Book of Changes. *I imagine that the device provides a random selection of numbers from 1-64 which is basically what the procedure of tossing coins is all about.*

If you have any information on such a device, I would like to hear from you about it." Robert Mahon

We know of no machine dedicated to I Ching coin tossing. Computers can do it, however, with a line similar to: 10X = INT (RND(1) * 64) + 1.

"I saw a cryptic comment — I think in **COMPUTE!** *#10: "PET Exec Hello" by Gordon Campbell. Second paragraph: POKE 59458,62 (this may damage your machine). Can I damage a PET with POKES?? It scared me. We just got a (used) PET—Original ROMs. I heard you have published a PET book based on old issues of* **COMPUTE!**. *How can I get this?" Felix Rosenthal*

You *can* damage the computer with this POKE. Luckily, it is the only POKE which is known to be risky, as far as we know. You can POKE freely anywhere else. For a more complete explanation of this peculiarity, see the warning in **COMPUTE!** #14, pg. 63. To answer your second question: yes **COMPUTE!** is publishing two such collections, one for PET and one for ATARI. These books contain much from the early, out-of-print **COMPUTE!** issues (as well as some previously unpublished pieces). For ordering information, see the ads elsewhere in this issue.

"Is there a spelling program that checks spelling of words in a Applewriter file? Is there a mailing label program that drives files in Applewriter format?"

John Hudson Tiner

©

WHY THE MICROSOFT RAMCARD™ MAKES OUR SOFTCARD™ AN EVEN BETTER IDEA.

Memory — you never seem to have quite enough of it.

But if you're one of the thousands of Apple owners using the SoftCard, there's an economical new way to expand your memory dramatically.

16K ON A PLUG-IN CARD.

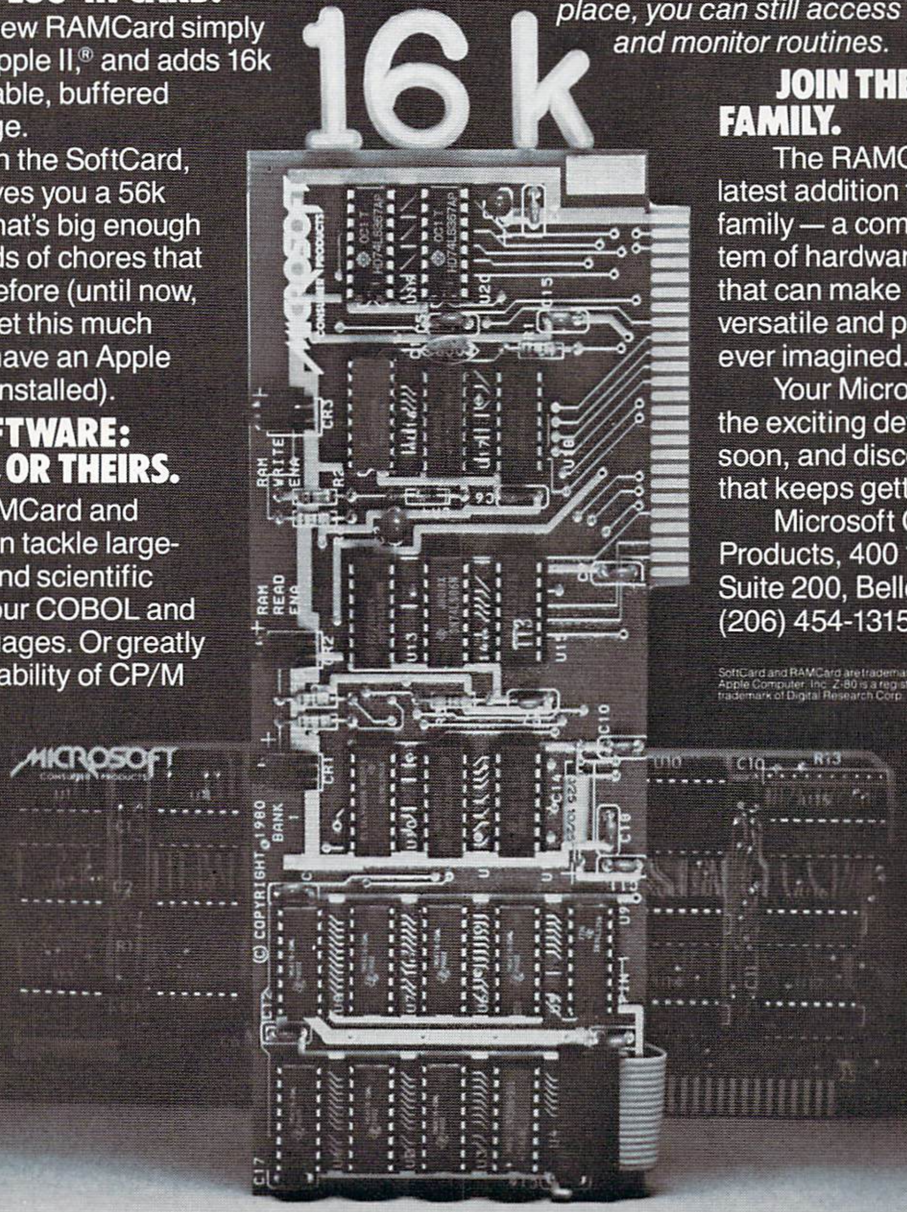
Microsoft's new RAMCard simply plugs into your Apple II,® and adds 16k bytes of dependable, buffered read/write storage.

Together with the SoftCard, the RAMCard gives you a 56k CP/M® system that's big enough to take on all kinds of chores that would never fit before (until now, the only way to get this much memory was to have an Apple Language Card installed).

GREAT SOFTWARE: YOURS, OURS, OR THEIRS.

With the RAMCard and SoftCard, you can tackle large-scale business and scientific computing with our COBOL and FORTRAN languages. Or greatly increase the capability of CP/M applications like the Peachtree Software accounting systems. VisiCalc™ and other Apple software packages can take advantage of RAMCard too.

And RAMCard gives you the extra capacity to develop advanced programs of your own, using the SoftCard and CP/M. *Even with the RAMCard in place, you can still access your ROM BASIC and monitor routines.*



JOIN THE SOFTCARD FAMILY.

The RAMCard is just the latest addition to the SoftCard family — a comprehensive system of hardware and software that can make your Apple more versatile and powerful than you ever imagined.

Your Microsoft dealer has all the exciting details. Visit him soon, and discover a great idea that keeps getting better.

Microsoft Consumer Products, 400 108th Ave. N.E., Suite 200, Bellevue, WA 98004. (206) 454-1315.

MICROSOFT

Basically Useful BASIC

Automatic DATA Statements For CBM And Atari

Dr. Harald Linder
Krefeld, West Germany

The following BASIC program converts a program from machine language into BASIC DATA statements by means of the "dynamic keyboard." For the Original PETs, the numbers 623,624,158 in the last line, must be replaced by 527,528,525.

```
1 INPUT"START ADDRESS";A:INPUT"END AD~
  ~DRESS";E:Z=2000
2 PRINT"{CLEAR}{2 DOWN}"Z"DATA";:IFA>E~
  ~THENEND
3 FORA=ATOA+15+(E<A+15)*(A+15-E)
4 PRINTMID$(STR$(PEEK(A)),2)",";:NEXT~
  ~
5 PRINT"{LEFT} ":PRINT"A="A":E="E":Z=~
  ~"Z+10":GOTO2{HOME}";
6 POKE623,13:POKE624,13:POKE158,2:END~
  ~
```

Program 1. CBM Version

```
0 BEG=7*4096:FIN=BEG+759:? "{CLEAR 2 DOW
N }"
1 FOR I=BEG TO FIN STEP 6
2 ? I;" DATA ";
3 FOR J=I TO I+5
4 ? PEEK(J);",";
5 NEXT J:? CHR$(126)
6 NL=NL+1:IF NL<15 THEN 9
7 ? "CONT":POSITION 2,0:POKE 842,13:STOP
8 POKE 842,12:NL=0:? "{CLEAR 3 DOWN}"
9 NEXT I:? "CONT"
10 POSITION 2,0:POKE 842,13:STOP
11 POKE 842,12:NL=0:? "{CLEAR 2 DOWN}"
12 FOR I=0 TO 12:? I:NEXT I:? "POKE 842,
12":GOTO 10
```

Program 2: Atari Version

©

The Beginner's Page

Richard Mansfield
Assistant Editor

From Chaos To Bits

Computers are sometimes called *data processors*. Data is processed by programs. You might type in a list of all the articles in this year's **COMPUTE!** and then write a program which will show you only the articles on, say, computer music. We will write such a program next month. What we want to see now is how data can be set up in *files* to make it easier for the computer to process it.

Our list of articles, while interesting in itself, is *raw data*. It sits in the program as DATA statements (or it could be on a disk or tape *file*) — the important thing to realize is that a program will later operate on the list, refining it into more meaningful information. The *computing* (or processing) aspect of this program might be to generate a more specific list: perhaps all articles by a particular author. However, for the computer to process data, the data must be somewhat organized already.

Organizing Data

If you look at Figure 1, you will notice that there are a number of divisions, each nested within a larger division. Here is a DATA statement taken from our proposed **COMPUTE!** index program which will help to illustrate Figure 1.

```
500 DATA FILES-DATA STORAGE TYPES*
    MANSFIELD*17
```

We can start from the outer ring of chaos and work inward. You make a stack of this year's **COMPUTE!** If your stack of magazines were burned to ash, the molecules of ink and paper would no longer have any meaningful relationship to each other and could not be called "information." Taken as a whole (as a stack) it is not *data*, exactly, because data is special: it is information organized so that it communicates a particular meaning. Your computer cannot read (yet), so the articles in **COMPUTE!** do not become meaningful data for the computer until you type them in as DATA statements or put them on tape or disk files.

Data is divided into files. An entire list of all year's articles is one *file*. A list of your stocks and bonds would be another file.

Within files there are *records*. Our DATA statement (line 500 above) is a record. It is a subdivision within the "**COMPUTE!** Articles File" which refers to a single, logical grouping of information (in this case, the information on a single article). In the financial portfolio file, all the information about a particular stock would be a record. Records

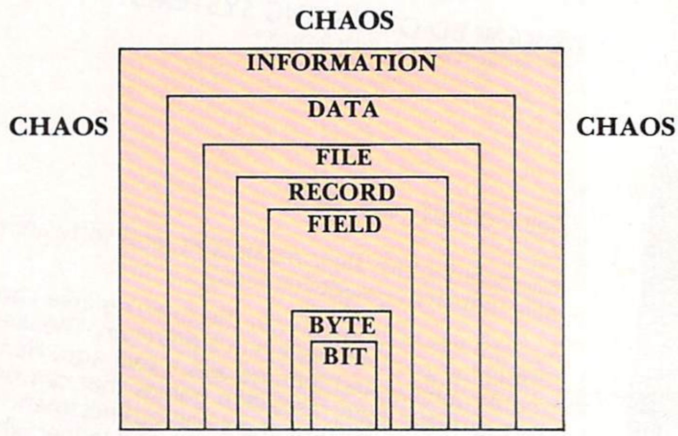


Figure 1.

are further divided into "fields" of information. We have chosen to use three fields: 1. A description of the article, 2. Author, 3. Issue Number.

As an aside, we should note that there is something special about the first word in our example record. To make it easier on the computer, one part of a record (often the first field, or part of it) is designated the *key*. Sometimes the key is a number, but we are using the first five characters of the first field ("FILES") as our key. We have decided to key this file by topics. We chose each topic name so that it would be only five letters long. FILES, MUSIC, ART , (notice the two spaces after "ART " to make it five long), ML   , (machine language), BASIC, MAPS , INTER (interfacing), DISKS, TAPE , PRINT (printers), MODEM, and any other keys we want.

Bytes and Bits

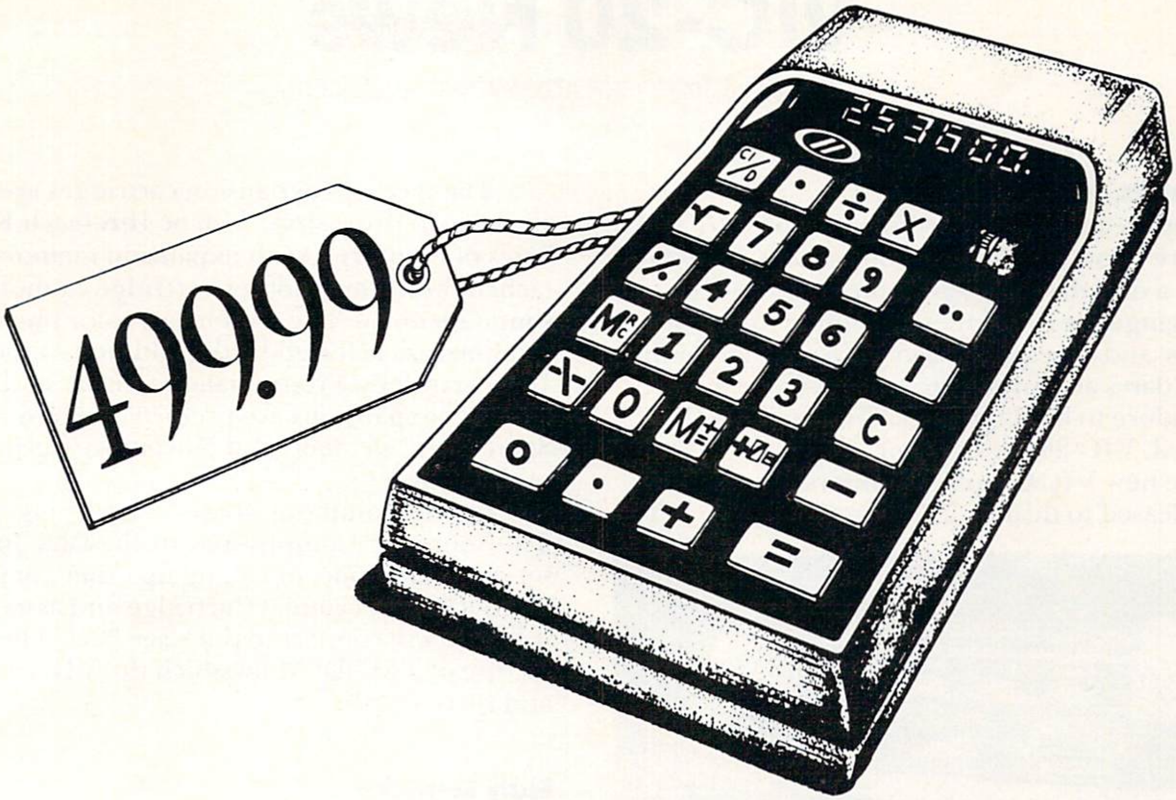
Finally, the smallest units of information are individual symbols, letters, and numbers. Each single-character piece of information is called a *byte*. A byte is able to store the numbers zero through 255. Since there are 26 letters in the alphabet, 26 capital letters, and number symbols 0 to 9, and assorted other symbols such as commas and brackets — the number of symbols we use to communicate with is less than 255. So, since a byte can store up to the number 255, each byte can "hold" a number value which represents a particular letter of the alphabet, numeral, or punctuation mark. Your computer stores the number 65, not the letter "A." A code was devised (the ASCII code) which assigns the number 65 to capital "A" and 193 to small "a." Every letter is represented by a particular number. Lower case "b" is 194.

Each byte is made up of eight *bits*. Where a byte can mean the numbers from 0 to 255, a bit can

only have two meanings: zero or one. Sometimes it is useful to think of a bit as being either *yes* or *no, on* or *off, positive* or *negative*. This "two-state" (binary) bit is often mentioned as the smallest possible unit of information. Even though they have only two states, bits can add up quickly: eight bits together make up a 256-state byte. A grouping of only two bytes can have more than 65,000 possible states — in other words, you could count up to 65,535 using only two bytes.

Processing Data

We have moved down through data from chaos to bits, from the largest to the smallest units. There are many ways to organize fields within records, records within files, and files within a large collection of data (a *database*). Some thought must go into the structure of this organization so that a program can later process the data efficiently. We decided to use the first five bytes (characters) of each of our records as the key to our **COMPUTE!** file. Next month we will build a program which will demonstrate some of the techniques of database management. This program will also illustrate the importance of those string-manipulating BASIC commands: LEFT$, RIGHT$, and MID$. ©


49.99

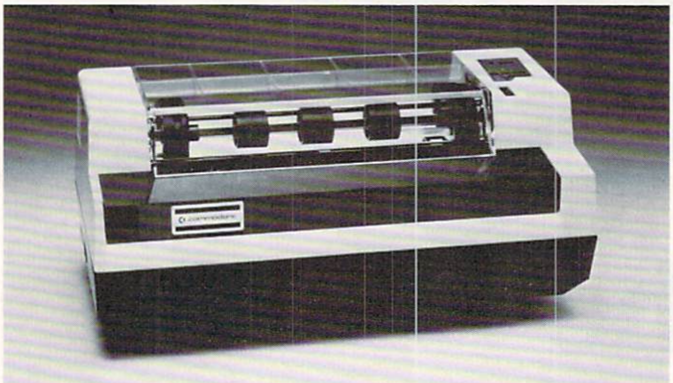
VIC-20 News

Compiled from sources by the editors.

VIC Printers, Software, Disks

Commodore has announced several software and hardware items for the VIC computer. Games, a printer, a disk drive, programmer's aid and assembly language cartridges, memory expansion modules, and an expansion interface are all coming. Release dates and prices are tentative. We expect Commodore to be shipping US produced, FCC approved, VIC-20s by October.

The new VIC Graphics Printers are expected to be released to dealers in September. It is an 80



column, twelve characters-per-inch, dot-matrix printer with 60 dots per inch (both horizontal and vertical) resolution. Its speed is 30 cps (characters per second) which means that an average typewritten page of about 275 words can be printed in about a minute.

The printer will permit the user to define his own characters. Each dot is programmable and the unit will also print the VIC graphics. It uses eight inch tractor feed paper, but can be narrowed to smaller widths for printing labels, etc.. It features a test mode and uses a ribbon cartridge (available from any Axiom distributer and soon from Commodore dealers). Seikosha manufactures the printer. Suggested retail is $395.

The VIC Disk

Sometime after Christmas, Commodore expects to begin selling a single disk drive which will attach to the VIC serial port. In addition, the drive is planned to be compatible with the 2040 disk drives used on the PET computers. An IEEE interface cartridge has also been announced which will permit PET peripherals to be attached directly to the VIC through the expansion port or an expansion module. This module will contain six slots and accept program cartridges, memory expansion cartridges, or interface cartridges.

The memory expansion cartridges are to be available in three sizes: 3, 8, or 16K (each K is 1024 bytes of memory). With expansion memory attached, however, another cartridge cannot be used simultaneously. The screen and color memory locations are affected by the addition of the 8 or 16K cartridges. From smallest to largest, these memory expansions are predicted to be available September, October, and November (respectively) of this year.

For telecommunications — attaching VIC to The Source or Compuserve, or the Dow Jones services via phone, or calling up other computers — an RS232 Terminal Cartridge and associated software will connect to the User Port. This permits the use of a MODEM by which the VIC can make and receive calls.

Early Software

Blackjack, Slither/Superslither, Biorhythm Compatibility, Space Math, Car Chase, and Blue Meanies from Outer Space are in release and will be reviewed in the fall issue of *Home and Educational COMPUTING!*. Planned for October release are: Jupiter Lander, Superslot, Night Driver, Draw Poker, VIC Avengers, and VIC Alien, a maze game.

BASIC programming will be assisted by another projected cartridge, Programmer's Aid, which will add new commands to BASIC for plotting, sound, color, music, and high resolution graphics. It will permit the user to define his keys however he wishes, provide simple music notation, color in an enclosed area, and so forth. The commands will be permitted in both BASIC programs and the immediate mode.

For machine language programmers, November is the target for a machine language monitor cartridge which will feature a simple assembler and disassembler. An intriguing feature of this software is a facility to swap zero page out and define a virtual zero page anywhere in memory. Machine language programmers know the value of zero page addressing on the 6502 microprocessor. BASIC will, of course, need its zero page when in operation.

A book, *The Programmer's Reference Manual*, is in the works too. It will contain a memory map, machine language and BASIC specifications, VIC chip details, and schematics. The title is tentative. It might be distributed as *The VIC-20 Reference Manual*.

Various VIC Memory Locations

Jim Wilcox, Vienna, WV

Editor's Note: A Full VIC memory map will appear in the fall Home and Educational COMPUTING! *issue. Here are some PEEKs and POKEs to get you started. Descriptions in parentheses have been added. — RTM*

Listed below are as many memory locations in the VIC as I could find by PEEKing around the memory. I also found out how to stop the use of the RUN/STOP key by the statement POKE788,194 and to restore the RUN/STOP key, POKE788, 191. This also stops the TI and TI$ when POKEing 788,194. I hope these are as useful to you as they were to me.

0–2 — USR function vectors. (Here is where the memory address is placed for jumping to a machine language routine from BASIC when you want to transfer a number from BASIC to the machine language routine.)

43–44 — Start of BASIC. (These two bytes contain the address where your BASIC program starts. PEEK (43) + PEEK (44) * 256 will give the address as a decimal number.)

45–46 — End of BASIC, start of variables. (Using the formula above, you can find out where your program ends in memory. The VIC starts storing its simple variables wherever there is room just above the program.)

47–48 — Array table. (The arrays are stored here.)

49–50 — End of Arrays.

51–52 — Start of Strings. (String variables.)

55–56 — End of memory. (How much RAM is available for use in BASIC. Sometimes, machine language programs are put at the "top of available RAM." 55 and 56 must be changed to fool the VIC into thinking that it has less memory for a BASIC program so it will not "write over" the machine language program. Changing the definition of end-of-memory will "protect" the machine language.)

57–58 — Current line number. (BASIC keeps track of the program line number.)

115–138 — Charget RAM code. (There is a small machine language program placed into this location each time power comes on. It gets a character in BASIC, but machine language programmers can put a JMP in it to allow the addition of new BASIC commands. Like the clock, the keyboard, and a few other items, this routine is constantly checked by BASIC to see if anything needs to be done. It can be used, therefore, as a way to append things to BASIC. You could not append to the keyboard checking routine, for example, because it is frozen into ROM. This part of BASIC's housekeeping is in RAM.)

145 — Run/Stop keys pressed, left shift pressed, polls every other of the bottom row of keys. (You could PEEK this to see if these keys were being pressed).

160–162 — The clock. (Write: 10 PRINT PEEK (160); PEEK (161); PEEK (162) [cursor home] to see it running.)

197 — Last key pressed. (Write: 10 PRINT PEEK (197) to see what the VIC sees when you press keys.)

198 — Number of keys pressed (cumulative).

203 — Last key pressed.

204 — Tells if cursor is to blink (0) or not (1).

205 — Countdown for blinking of cursor.

246 — Tells if SHIFT, Commodore, or CTRL keys are pressed.

512–600 — BASIC buffer. (A "storage" buffer is a temporary holding area where bytes wait until there is time to use them. BASIC itself uses this area).

631–640 — Keyboard buffer.

651–652 — Repeat keys pressed.

788–789 — Interrupt address. (Important in machine language programming.)

4096 — BASIC starts. (Where the first byte of your BASIC program starts.)

Update
Floating Color Floating Screen

If you are writing software for the VIC — either professionally or for your own use — you should include a line in your program which locates the screen and color memories. As it comes from the factory, the VIC screen memory is located at addresses 7680 to 8191. Memory expansion modules are going to be available soon which can add 3 or 8 or 16K to the VIC. The 3K expansion will fill a hole from addresses 1024 to 4095, and will not affect the locations of color or screen memory. Adding an 8 or 16K memory expansion will, however, **move these important memories.**

What this means is that *any programs which manipulate color or screen data* (such as the direct POKEs to screen memory used in many games) *will not work correctly* when the larger two memory expansion modules are added to the VIC. To prevent problems later — to make your programs find VIC's floating screen and color memories — you should add the following formulae which will provide the true addresses:

S (starting address of screen memory)
S=4 * (PEEK(36866)AND128)+64 * (PEEK(36869) AND120)
C (starting address of color memory)
C= 37888+4 * (PEEK(36866)AND128)

To use these formulae in your programs, you should enter the two lines above as program lines at the start of your program. (10 S=4 * (PEEK...etc.) Then, whenever you are working with these memories, simply POKE to S + X or C + X. In other words, use the S and the C instead of a numerical address. For example, to POKE to the tenth screen location, you would POKE S + 10.

Adding a 3K (3072 additional RAM bytes for your use) expansion module will not change any of the normal, expected locations of screen or color memories. It simply fills in a currently empty space from addresses 1024 to 4095. This results in BASIC programs starting at address 1024 (as they do in PETs) instead of the normal VIC starting point, 4096. (See Table 1.) However, adding an 8 or 16K of additional memory floats the screen down to 4096 (from 7680). BASIC RAM floats to a starting address of 4608. And, since one of the bits which governs where screen memory starts also controls color memory, it moves too.

Character Memories

The starting address of the character set memory does not float, so you need not check for it in programs. However, the ability to define alternative character sets is valuable. There are sixteen possible locations in VIC for the start of *character* set memories. Of these, eight can be used (the others are not allowed). Here's the formula to change the character memory location:

POKE 36869, PEEK(36869)AND15OR(X*16)

X will be a number from 0 to 15. Here are the starting locations in memory for several values of X:

X=0 (32768) this is the normal "default" starting location.
X=1 (33792) where the upper case reverse characters normally are.
X=2 (34816) normally the lower case, unreversed characters.
X=3 (35840) normally lower case reversed.
X=4 to 11 (cannot be used).
X=12 (4096) normally the start of available BASIC RAM.
X=13 (5020) normally within BASIC RAM.
X=14 (6144) normally within BASIC RAM.
X=15 (7168) normally within BASIC RAM.

These last four values of X are where you would usually want to put any specially written character set you've invented.

Table 1. General VIC Map

0–1023 — Operating System and BASIC Overhead
1024–4095 — Empty memory (3K Expansion area)
4096–7679 — BASIC RAM memory
7680–8191 — Screen Memory
8192–32767 — 24K Additional expansion RAM area

©

BUSINESS and UTILITIES

SPELLGUARD™ (CP/M only) — **Price: $219.95 Disk**

SPELLGUARD is a revolutionary new product which increases the value of your current word processing system (WORDSTAR, MAGIC WAND, ELECTRIC PENCIL, TEXTED EDITOR II and others). Written entirely in assembly language, SPELLGUARD™ rapidly assists the user in eliminating spelling and typographical errors by comparing each word of the text against a dictionary (expandable) of over 20,000 of the most common English words. Words appearing in the text but not found in the dictionary are "flagged" for easy identification and correction. Most administrative staff familiar with word processing equipment will be able to use SPELLGUARD™ in only a few minutes.

MAIL LIST 2.2 (Apple, Atari and North Star diskette only) — **Price: $34.95**

This program is unmatched in its ability to store a maximum number of addresses on one diskette (minimum of 1100 per diskette, more than 2200 for "double density" systems!). Its many features include alphabetic and zip code sorting, label printing (1, 2, or 3 up), merging of files and a unique keyword seeking routine which retrieves entries by a virtually limitless selection of user defined codes. Mail List 2.2 will even find and delete duplicate entries. A very valuable program!

FORM LETTER SYSTEM rel. 2 (Atari, North Star and Apple Diskettes only) — **Price: $34,95**

FORM LETTER SYSTEM (FLS) is the ideal program for creating and editing form letters and address lists. It contains an easy-to-use text editor which produces fully justified text. Special codes are used in the address list to obtain personalized salutations. Form letters are produced by automatically inserting each address into a predetermined portion of your letter. FLS is completely compatible with MAIL LIST 2.2, which may be used to manage and sort your address files.

FLS and MAIL LIST 2.2 are available as a combined package for $59.95.

SORTIT (North Star only) — **Price: $29.95 Diskette**

SORTIT is a general purpose sorting program written in 8080 assembly language. This program will sort sequential data files generated by NORTH STAR BASIC. Primary and optional secondary keys may be numeric or one to nine character strings. SORTIT is easily used with files generated by DYNACOMP's MAIL LIST program and is very versatile in its capabilities for all other BASIC data file sorting.

PERSONAL FINANCE SYSTEM (Atari and North Star only) — **Price: $34.95 Diskette**

PFS is a single diskette, menu-oriented system composed of ten different programs. Besides recording your expenses and tax deductible items, PFS will sort and summarize expenses by payee, and display information on expenditures by any of 26 user defined codes by month or by payee. PFS will even produce monthly bar graphs of your expenses by category! This powerful package requires only one disk drive, minimal memory (24K Atari, 32K North Star) and will store up to 600 records per disk (and over 1000 records per disk by making a few simple changes to the programs). You can record checks plus cash expenses so that you can finally see where your money goes and eliminate guesswork and tedious hand calculations.

FAMILY BUDGET (Apple only) — **Price: $34.95 Diskette**

FAMILY BUDGET is a very convenient financial record-keeping program. You will be able to keep track of cash and credit expenditures as well as income on a daily basis. You can record tax deductible items and charitable donations. FAMILY BUDGET also provides a continuous record of all credit transactions. You can make daily cash and charge entries to any of 21 different expense accounts as well as to 5 payroll and tax accounts. Data are easily retrieved giving the user complete control over an otherwise complicated (and unorganized!) subject.

INTELINK (Atari only) — **Price: $49.95 Diskette**

This software package contains a menu-driven collection of programs for facilitating efficient two-way communications through a full duplex modem (required for use). In one mode of operation you may connect to a data service (e.g., The SOURCE or MicroNet) and quickly load data such as stock quotations onto your diskette for later viewing. This greatly reduces "connect time" and thus the service charge. You may also record the complete contents of a communications session. Additionally, programs written in BASIC, FORTRAN, etc. may be built off-line using the support text editor and later "uploaded" to another computer, making the Atari a *very* smart terminal. Even Atari BASIC programs may be uploaded. Further, a command file may be built off-line and used later as controlling input for a time-share system. That is, you can set up your sequence of time-share commands and programs, and the Atari will transmit them as needed; batch processing. All this adds up to saving both connect time and your time.

TEXT EDITOR II (CP/M) — **Price: $29.95 Diskette/$33.45 Disk**

This is the second release version of DYNACOMP's popular TEXT EDITOR I and contains many new features. With TEXT EDITOR II you may build text files in chunks and assemble them for later display. Blocks of text may be appended, inserted or deleted. Files may be saved on disk/diskette in right justified/centered format to be later printed by either TEXT EDITOR II or the CP/M ED facility. Futher, ASCII CP/M files (including BASIC and assembly language programs) may be read by the editor and processed. In fact, text files can be built using ED and later formatted using TEXT EDITOR II. All in all, TEXT EDITOR II is an inexpensive, easy to use, but very flexible editing system.

DFILE (Atari and North Star diskettes only) — **Price: $19.95**

This handy program allows North Star and Atari disk users to maintain a specialized data base of all files and programs in the stack of disks which invariably accumulates. DFILE is easy to set up and use. It will organize your disks to provide efficient locating of the desired file or program.

FINDIT (North Star only) — **Price: $19.95**

This is a three-in-one program which maintains information accessible by keywords of three types: Personal (eg: last name), Commercial (eg: plumbers) and Reference (eg: magazine articles, record albums, etc). In addition to keyword searches, there are birthday, anniversary and appointment searches for the personal records and appointment searches for the commercial records. Reference records are accessed by a single keyword or by cross-referencing two or three keywords.

SHOPPING LIST (Atari and North Star only) — **Price: $12.95 Cassette/$16.95 Diskette**

SHOPPING LIST stores information on items you purchase at the supermarket. Before going shopping, it will remind you of all the things you might need, and then display (or optionally print) your shopping list and the total cost. Adding, deleting, changing and storing data is very easy. Runs with 16K.

TAX OPTIMIZER (North Star only) — **Price: $59.95 Diskette**

The TAX OPTIMIZER is an easy-to-use, menu oriented software package which provides a convenient means for analyzing various income tax strategies. The program is designed to provide a quick and easy data entry. Income tax is computed by all tax methods (regular, income averaging, maximum and alternate minimum tax). The user may immediately observe the tax effect of critical financial decisions. TAX OPTIMIZER has been thoroughly field tested in CPA offices and comes complete with the current tax tables in its data files.

EDUCATION

HODGE PODGE (Apple only, 48K Applesoft or Integer BASIC) — **Price: $19.95 Cassette/$23.95 Diskette**

Let HODGE PODGE be your child's baby sitter. Pressing any key on your Apple will result in a different and intriguing "happening" related to the letter or number of the chosen key. The program's graphics, color and sound are a delight for children from ages 1½ to 9. HODGE PODGE is a non-intimidating teaching device which brings a new dimension to the use of computers in education.

TEACHER'S PET I (Available for all computers) — **Price: $11.95 Cassette/$15.95 Diskette**

This is the first of DYNACOMP's educational packages. Primarily intended for pre-school to grade 3, TEACHER'S PET provides the young student with counting practice, letter-word recognition and three levels of math skill exercises.

MISCELLANEOUS

CRYSTALS (Atari only) — **Price: $ 9.95 Cassette/$13.95 Diskette**

A unique algorithm randomly produces fascinating graphics displays accompanied with tones which vary as the patterns are built. No two patterns are the same, and the combined effect of the sound and graphics are mesmerizing. CRYSTALS has been used in local stores to demonstrate the sound and color features of the Atari.

NORTH STAR SOFTWARE EXCHANGE (NSSE) LIBRARY

DYNACOMP now distributes the 23 volume NSSE library. These diskettes each contain many programs and offer an outstanding value for the purchase price. They should be part of every North Star user's collection. Call or write DYNACOMP for details regarding the contents of the NSSE collection.

Price: $9.95 each/$7.95 each (4 or more)
The complete collection may be purchased for $149.95

AVAILABILITY

DYNACOMP software is supplied with complete documentation containing clear explanations and examples. Unless otherwise specified, all programs will run within 16K program memory space (ATARI requires 24K). Except where noted, programs are available on ATARI, PET, TRS-80 (Level II) and Apple (Applesoft) cassette and diskette as well as North Star single density (double density compatible) diskette. Additionally, most programs can be obtained on standard (IBM format) 8" CP/M floppy disks for systems running under MBASIC.

STATISTICS and ENGINEERING

DIGITAL FILTER (Available for all computers) — **Price: $39.95 Cassette/$43.95 Diskette**

DIGITAL FILTER is a comprehensive data processing program which permits the user to design his own filter function or choose from a menu of filter forms. The filter forms are subsequently converted into non-recursive convolution coefficients which permit rapid data processing. In the explicit design mode the shape of the frequency transfer function is specified by directly entering points along the desired filter curve. In the menu mode, ideal low pass, high pass and bandpass filters may be approximated to varying degrees according to the number of points used in the calculation. These filters may optionally also be smoothed with a Hanning function. In addition, multi-stage Butterworth filters may be selected. Features of DIGITAL FILTER include plotting of the data before and after filtering, as well as display of the chosen filter functions. Also included are convenient data storage, retrieval and editing procedures.

DATA SMOOTHER (Not available for Atari) — **Price: $19.95 Cassette/$23.95 Diskette**

This special data smoothing program may be used to rapidly derive useful information from noisy business and engineering data which are equally spaced. The software features choice in degree and range of fit, as well as smoothed first and second derivative calculation. Also included is automatic plotting of the input data and smoothed results.

FOURIER ANALYZER (Available for all computers) — **Price: $19.95 Cassette/$23.95 Diskette**

Use this program to examine the frequency spectra of limited duration signals. The program features automatic scaling and plotting of the input data and results. Practical applications include the analysis of complicated patterns in such fields as electronics, communications and business.

TFA (Transfer Function Analyzer) — **Price: $19.95 Cassette/$23.95 Diskette**

This is a special software package which may be used to evaluate the transfer functions of systems such as hi-fi amplifiers and filters by examining their response to pulsed inputs. TFA is a major modification of FOURIER ANALYZER and contains an engineering-oriented decibel versus log-frequency plot as well as data editing features. Whereas FOURIER ANALYZER is designed for educational and scientific use, TFA is an engineering tool. Available for all computers.

HARMONIC ANALYZER (Available for all computers) — **Price: $24.95 Cassette/$28.95 Diskette**

HARMONIC ANALYZER was designed for the spectrum analysis of repetitive waveforms. Features include data file generation, editing and storage/retrieval as well as data and spectrum plotting. One particularly unique facility is that the input data need not be equally spaced or in order. The original data is sorted and a cubic spline interpolation is used to create the data file required by the FFT algorithm.

FOURIER ANALYZER, TFA and HARMONIC ANALYZER may be purchased together for a combined price of $49.95 (three cassettes) and $59.95 (three diskettes).

REGRESSION I (Available for all computers) — **Price: $19.95 Cassette/$23.95 Diskette**

REGRESSION I is a unique and exceptionally versatile one-dimensional least squares "polynomial" curve fitting program. Features include very high accuracy; an automatic degree determination option; an extensive internal library of fitting functions; data editing; automatic data and curve plotting; a statistical analysis (eg: standard deviation, correlation coefficient, etc.) and much more. In addition, new fits may be tried without reentering the data. REGRESSION I is certainly the cornerstone program in any data analysis software library.

REGRESSION II (PARAFIT) (Available for all computers) — **Price: $19.95 Cassette/$23.95 Diskette**

PARAFIT is designed to handle those cases in which the parameters are imbedded (possibly nonlinearly) in the fitting function. The user simply inserts the functional form, including the parameters (A(1), A(2), etc.) as one or more BASIC statement lines. Data and results may be manipulated and plotted as with REGRESSION I. Use REGRESSION I for polynomial fitting, and PARAFIT for those complicated functions.

MULTILINEAR REGRESSION (MLR) (Available for all computers) — **Price: $24.95 Cassette/$28.95 Diskette**

MLR is a professional software package for analyzing data sets containing two or more linearly independent variables. Besides performing the basic regression calculation, this program also provides easy to use data entry, storage, retrieval and editing functions. In addition, the user may interrogate the solution by supplying values for the independent variables. The number of variables and data size is limited only by the available memory.

REGRESSION I, II and MULTILINEAR REGRESSION may be purchased together for $51.95 (three cassettes) or $63.95 (three diskettes).

ANOVA (Available for all computers) — **Price: $39.95 Cassette/$43.95 Diskette**

In the past the ANOVA (analysis of variance) procedure has been limited to the large mainframe computers. Now DYNACOMP has brought the power of this method to small systems. For those conversant with ANOVA, the DYNACOMP software package includes the 1-way, 2-way and N-way procedures. Also provided are the Yates 2^{K-P} factorial designs. For those unfamiliar with ANOVA, do not worry. The accompanying documentation was written in a tutorial fashion (by a professor in the subject) and serves as an excellent introduction to the subject. Accompanying ANOVA is a support program for building the data base. Included are several convenient features including data editing, deleting and appending.

BASIC SCIENTIFIC SUBROUTINES, Volume 1 (Not available for Atari)

DYNACOMP is the exclusive distributor for the software keyed to the popular text *BASIC Scientific Subroutines, Volume I* by F. Ruckdeschel (see the BYTE/McGraw-Hill advertisement in BYTE magazine, January 1981). These subroutines have been assembled according to chapter. Included with each collection is a menu program which selects and demonstrates each subroutine.

Collection #1: Chapters 2 and 3: Data and function plotting, complex variables
Collection #2: Chapter 4: Matrix and vector operations
Collection #3: Chapters 5 and 6: Random number generators, series approximations
Price per collection: $14.95 Cassette/$18.95 Diskette

All three collections are available for $39.95 (three cassettes) and $49.95 (three diskettes).
Because the text is a vital part of the documentation, *BASIC Scientific Subroutines, Volume I* is available from DYNACOMP for $19.95 plus 75¢ postage and handling. See review in Dr. Dobbs.

ROOTS (Available for all computers) — **Price $10.95 Cassette/$14.95 Diskette**

In a nutshell, ROOTS simultaneously determines all the zeroes of a polynomial having real coefficients. There is no limit on the degree of the polynomial, and because the procedure is iterative, the accuracy is generally very good. No initial guesses are required as input, and the calculated roots are substituted back into the polynomial and the residuals displayed.

ACTIVE CIRCUIT ANALYSIS (ACAP) (48K Apple only) — **Price: $25.95 Cassette/$29.95 Diskette**

ACAP is the analog circuit designer's answer to LOGIC SIMULATOR. With ACAP you may analyze the response of an active or passive component circuit (e.g., a transistor amplifier, band pass filter, etc.). The circuit may be probed at equal steps in frequency, and the resulting complex (i.e., real and imaginary) voltages at each component juncture examined. By plotting the magnitude of these voltages, the frequency response of a filter or amplifier may be completely determined with respect to both amplitude and phase. In addition, ACAP prints a statistical analysis of the range of voltage responses which result from tolerance variations in the components. ACAP is easy to learn and use. Simply describe the circuit in terms of the elements and their placement, and execute. Circuit descriptions may be saved onto cassette or diskette to be recalled at a later time for execution or editing. ACAP should be part of every circuit designer's program library.

LOGIC SIMULATOR (Apple only; 48K RAM) — **Price: $24.95 Cassette/$28.95 Diskette**

With LOGIC SIMULATOR you may easily test your complicated digital logic design with respect to given set of inputs to determine how well the circuit will operate. The elements which may be simulated include multiple input AND, OR, NOR, EXOR, EXNOR and NAND gates, as well as inverters, J-K and D flip-flops, and one-shots. The response of the system is available every clock cycle. Inputs may be clocked in with varying clock cycle lengths/displacements and delays may be introduced to probe for glitches and race conditions. At the user's option, a timing diagram for any given set of nodes may be plotted using HIRES graphics. Save your breadboarding until the circuit is checked by LOGIC SIMULATOR.

LOGIC DESIGNER (North Star and CP/M only) — **Price: $34.95 Diskette**

LOGIC DESIGNER is an exceptional Computer Aided Design (CAD) program. With it you may convert a large and complicated digitial truth table (the functional specification) into an optimized Boolean logic equation. This equation may then be *easily* converted into a circuit design using either NAND or AND/OR gates. Operationally, LOGIC DESIGNER is composed of a BASIC program which calls in a machine language routine to reduce execution time. Example: For a 7 variable by 127 line table, the processing time is only two minutes. LOGIC DESIGNER is clearly a fast and powerful tool for building digital circuitry.

ORDERING INFORMATION

All orders are processed and shipped within 48 hours. Please enclose payment with order and include the appropriate computer information. If paying by VISA or Master Card, include all numbers on card.

Shipping and Handling Charges
Within North America: Add $1.50
Outside North America: Add 10% (Air Mail)

Delivery
All orders (excluding books) are sent First Class.

Quantity Discounts
Deduct 10% when ordering 3 or more programs. Dealer discount schedules are available upon request.

8" CP/M Disks
Add $2.50 to the listed diskette price for each 8" floppy disk (IBM soft sectored CP/M format). Programs run under Microsoft MBASIC or BASIC-80.

5¼" CP/M Disks
All software available on 8" CP/M disks is also available on 5¼" disks, North Star format.

Ask for **DYNACOMP** programs at your local software dealer. Write for detailed descriptions of these and other programs from **DYNACOMP**.

DYNACOMP, Inc.
1427 Monroe Avenue
Rochester, New York 14618
24 hour order phone: (716)586-7579 recording
Office phone (9AM-5PM EST): (716)442-8960

VISA

MasterCard

New York State residents please add 7% NYS sales tax.

Guest Commentary:

A Software Publisher's Position On Software Pricing And Service Policies

Mr. Sherwin A. Steffin
Canoga Park, CA

I. Problem

A position paper appearing in **COMPUTE!**'s June issue advanced the views of Computer Using Educators (CUE) regarding commercial software protection and licensing policies. CUE would propose a licensing arrangement that would allow software "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."

A misperception is at work here. This organization appears to hold the fundamental view that copying computer software is a right. CUE recommends that schools should not purchase software material unless it is copyable.

II. Position

In 1980, congress amended title 17, the United States Copyright Law to include the "computer program". The amended code reads:

> It is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy or adaptation of that computer program provided:
>
> (1) that such a new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that is used in no other manner, or
>
> (2) that such new copy or adaptation is for archival purposes only and that all archival copies are destroyed in the event that continued possession of the computer program should cease to be rightful (17 USC 106).

The provisions of this law, capsulized in the phrases "essential step" and "archival purposes," are clear. Software copyright infringement is illegal. Complaints and rationalizations abound: that "restrictive" policies are unjust; that such policies do not "invite" teachers to respect and honor them; or that those who "beat the system" by pirating software from big corporations are to be applauded. Yet no measure of complaint will make the crime any less a crime. When a teacher permits (even encourages) children and young adults to disregard federal law and engage in illegal activities, what is that teacher undermining?

No software publisher would deny the necessity that teachers have on-site backup copies of software...

Costly and intensive research and development are required to produce any good software system, from drill and practice to the most sophisticated computer-mediated instruction. Even the simplest program file contains a complex set of instructions. Research and development costs far exceed the trivial cost of the material, or the ultimate retail cost of the product.

No software publisher would deny the necessity that teachers have on-site backup copies of software to avoid media failures which interrupt classroom activity. Yet CAI is a tool, like any other instructional aid. The purchasing policies which govern other educational materials (books, audio-visual materials, and the like) should not be expected to undergo modification simply because a medium's format is new or unfamiliar.

No software publisher fails to understand that the computer software industry has yet to adopt consistent, balanced purchasing policies for the schools. Yet, the instigation of licensing agreements poses problems analogous to those encountered in nuclear arms limitation efforts. The geographic area involved is expansive; the diversity of management policies (even among neighboring school districts) is immense. Further, the ambiguity of relying upon individuals' ethical behavior makes an "honors system" untenable. In short, the implementation of the copying procedures which CUE proposes, even at the local district level, would be unfeasible.

III. Proposed Solution

Answers do not come easily. The problems en-

countered today cannot be solved by a simplistic licensing policy, and the future will present even more confounding issues. Yet the difficulties are not insurmountable. As the microcomputer industry expands and time sharing becomes commonplace, users and software publishers will have to look seriously (and cooperatively) at complex joint understandings.

Software purchase and use entails shared responsibilities: on the part of the software publisher, to provide the most trouble-free product possible; on the part of the user, to know how to use and operate that product carefully and effectively. With this in mind, we propose six components fundamental to any working policy:

1. Software publishers should be accountable for providing error-free instructional materials.

2. Software publishers should be responsible for rapid service (a two week turn around at most) when replacing defective materials.

3. Where schools have purchased multiple CPUs, software publishers may provvide software at sliding scale rates. An affidavit from a responsible district administrator would certify the number of CPUs purchased, protecting all parties. This sliding scale would permit educators to utilize multiple computers in one room, in multiple rooms, among several teachers, and even among several schools within a single district.

4. When intensive disk activity is connected with the use of a given system, software publishers chould allow the user to copy and archive the disk.

5. Software publishers need to make available options to the schools, offering not only the sliding rate scale, but "spare parts" (diskettes, documentation, workbooks, etc.) at a significantly reduced price. This after-sale activity could well cover little more than the costs of materials, processing, and handling. Software purchase can be made economically feasible for the school without resort to charity or criminality.

6. Software publishers need to provide free disclosure about their locked instructional systems and the policies which support them. From this, educators may choose systems appropriate to their needs.

The problems connected with software piracy, if unchecked, will create an unnecessary and unfruitful adversary relationship between software publishers and educators. This paper delineates solutions to this dilemma: better quality control of software, licensing agreements, rapid service, and available spare parts. ©

At long last. A complete line of Voice I/O peripherals for Commodore computers, starting at $119.

Let's face it. Voice I/O is a fascinating and efficient way to communicate with computers. And now, thanks to VOICETEK, Voice I/O peripherals are easily available, easy to use and very affordable.

If you own a Commodore computer, we give you a choice of three different peripherals that will enable your computer to understand your spoken commands. And two of these peripherals will talk back.

What's in a name.

We call each member of our family of Voice I/O peripherals COGNIVOX, from COGNItive and VOX. It helps you remember that when you talk, COGNIVOX thinks about what you say.

The top-of-the-line COGNIVOX model for the PET/CBM is VIO-1002. It offers natural sounding voice output and excellent performance as a speech recognizer. It costs only $249.

If the quality of voice output is not important for your application, then you can save $100 by ordering VIO-432. Priced at $149, VIO-432 is ideal for hobbyists or persons mainly interested in speech recogniton.

Finally, if you have an 8K PET, there is insufficient memory for voice response, so we offer a recognition-only COGNIVOX, model SR-100P. It costs $119, making it the lowest priced speech recognizer ever offered for sale. Yet its performance rivals that of units selling at much higher prices.

Which brings us to the next point we would like to make, namely, why we offer so much performance for so little money.

It's the technology.

Our Voice I/O peripherals are based on a technological breakthrough that made it possible to compress the required electronics onto a single integrated circuit chip. We are the only company so far that has achieved this remarkable feat. No wonder we offer such reasonably price voice peripherals.

In addition, COGNIVOX uses an exclusive non-linear, learning pattern matching algorithm to do speech recognition. Which means more reliable performance and ease of use.

What makes it talk.

COGNIVOX digitizes and stores in memory (using a data compression algorithm) the voice of the user. This gives three major advantages:

First, there are no restrictions to the words COGNIVOX can say. If you can say it (or sing it, or whistle it for that matter) your computer can do it too.
Second, It is very easy to program your favorite words: just say them in the microphone.
Third, you have a choice of voices, male, female, child, accents, etc. this unprecendented flexibility offered by COGNIVOX is a must in the personal computer environment. Voice synthesizers and the "talking chips" do not offer this flexibility and therefore we feel they are not suitable for use with personal computers.
In addition, voice output quality can be poor, especially for synthesizers. In that respect, VIO-1002 is clearly superior to anything else on the market and it is a must if voice quality is important (for example, business applications).

Some specifications

COGNIVOX can be trained to recognize words or short phrases drawn from a vocabulary of up to 32 entries chosen by the user.

Training COGNIVOX to your vocabulary is easy. All you have to do is repeat the words three times at the prompting of the computer.

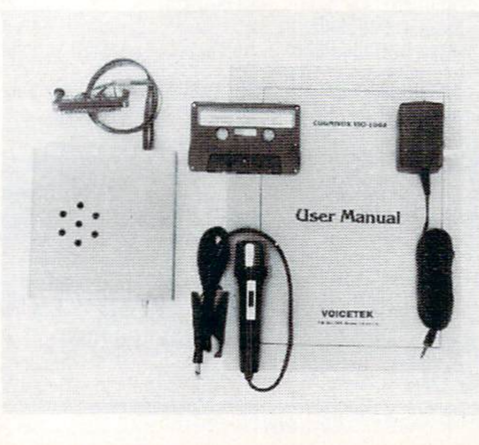
If you would like to have COGNIVOX respond to more than 32 words, you can have two or more vocabularies of 32 words and switch back and forth between them using a word.

The Voice output vocabulary can have up to 32 words phrases. Data rate is approximately 700 byte per word.

Ready to listen.

All COGNIVOX units are complete Voice I/O peripherals ready to plug in and use. They come assembled and tested and they include microphone, cassette with software and manuals. VIO units include built-in speaker and amplifier (yes, CB2 is also connected for music and sound effects).

They all plug into the user port and they receive their power from the cassette port except VIO-1002 which uses a wall transformer supplied with the unit.



Easy to use.

All you need to get COGNIVOX up and running is to plug it in and load one of the programs supplied. Load the demo program and start talking to your computer right away. Or load one of the games and discover the magic of voice control.

It is easy to write your own talking and listening programs too. A single statement in BASIC is all that you need to say a word or to recognize a word. Full instructions on how to do it are given in the manual.

Works with all versions.

COGNIVOX will work with all versions of the PET/CBM line. Old, new and newer ROMs. At least 16K of RAM is required (SR-100P will work with 8K of RAM).

If you have a disk system, you can use it to save vocabularies. Instructions are given in the manual.

Many uses.

With COGNIVOX your imagination is not the limit as the saying goes. It is the starting point. Cognivox is a super toy, an educational tool, an aid to handicapped, a data entry device while hands and eyes are busy, a foreign language translator, a sound effects generator, a telephone dialing device, an answering machine, a talking calculator. Using the IEEE 488 port you can control by voice instruments, plotters, test systems. And all these devices can talk back to you, telling you their readings, alarm conditions, even their name.

Order your COGNIVOX now.

To order by mail, give us the model number of the unit you wish to order, the make and model of your computer and your name and address. Enclose a check or money order and make sure to include $5 for shipping and handling. CA residents please add 6% tax. You may also order by phone and charge it to your Master Charge or VISA. Our phone number is (805) 685-1854 9AM to 5PM PST, Monday through Friday. Foreign orders are welcome, please add 10% for air mail shipping and handling. Payments must be in US funds. COGNIVOX is backed by an 120-day limited warranty against manufacturing defects.

VOICETEK

P.O. Box 388, Goleta, CA 93116

Part II: What Is A MODEM, And Why Do I Need One?

Michael E. Day
West Linn, OR

How To Use A Modem

The standard modem that the hobbyist normally encounters is the BELL 103 type modem. Additionally, the BELL 113A (originate) and the 113B (answer) are normally included in this group, with the specification of a 103 compatible modem often being used to indicate the 113A or 113B type modem. The 103 type modem is rated for speeds of 0 to 300 BPS, with most modems being able to operate up to 400 BPS, and the more expensive models being able to operate at up to 600 BPS. The maximum theoretical limit of the 103 type modem is 1000 BPS. However, due to the great amount of filtering and special line conditioning required to operate as the speed approaches this level, it becomes impractical to operate at these speeds. Due to this, 600 BPS is the maximum reliable speed that can be expected from the 103 type modem. It should be remembered, however, that 300 BPS is the maximum guaranteed speed of the 103 type modem. Speeds faster than this will not always work (depending on the phone line condition), and those modems capable of operating at greater speed generally cost twice as much as the lower speed types. The modem may be used at any speed less than 300 BPS. Dropping the speed to 150 or 110 BPS can often improve the reliability if the connection is very poor, and 300 BPS does not work.

The 103 has two modes of operation, the "answer" mode and the "originate" mode. The 113 modem will only work in one of the modes (113A for originate and 113B for answer) and not the other. Most of the acoustic coupled modems found on the surplus market are the originate type. This type of modem is what you need to talk to most of the computers that the hobbyist has access to (such as CBBS/NW). The originate modem is so named because it is used by the person or device that places the call (or originates the call) to the remote computer or person. The answer modem is used by the person or device that receives the call (or answers the call). The two modes could be reversed, as the phone line doesn't care. They were set that way to provide a standard as to which modem should use which mode. Since the 103 is a full duplex modem (two-day communications), two separate communication links must be established, thus the two different modes. The answer modem transmits on the high frequency link and receives on the low frequency link.

Of all the generally available modems, the 103 is the most forgiving.

A true 103 type modem will be capable of operation in either mode depending on a control function. (This could be as complex as a control sequence or as simple as a switch).

Of all the generally available modems, the 103 is the most forgiving. It will operate at any speed less than its designed maximum. It is totally transparent to any protocols that might be used as long as it is asynchronous type transmission and it requires no special handshaking (control) signals in its basic configuration. This is ideal in a portable application where it might be used in a wide variety of configurations.

PIN 1	FRAME GROUND	Tied to the modem case (if metal). Can be ignored normally.
PIN 2	TXD	Transmit Data. The data to be transmitted is presented to this pin.
PIN 3	RXD	Receive Data. The received data is present on this pin.
PIN 4	RTS	Request to send. Generally ignored by modem, it can Sometimes be used to turn the transmitter on and off (1 = on; 0 = off).
PIN 5	CTS	Clear to send. This pin is normally held high (on). Alternately it can follow RTS, or DCD, or both (depending on modem).
PIN 6	DSR	Data set ready. This signal will always be on when the modem is operational (power on).
PIN 7	LOGIC GROUND	This is the common reference ground for all the signals listed.
PIN 8	DCD	Data carrier detect. This signal will be on when the communications link has been established (the carrier signal from the remote modem is being received.) On some modems this is always on.
PIN 20	DTR	Data Terminal Ready. Depending on the modem, this can be ignored, the modem on or off (1 = on; 0 = off). Also, in some turn modems, it is used in conjunction with RI to set the operating mode.

PIN 21	SQD	Signal Quality Detector. Generally this signal is not provided. On those modems that do provide it, it indicates that a poor communications link has been established, and that there is a high probability of errors occuring.
PIN 22	RI	Ring Indicator. Those modems that do not have mode control generally ignore this signal. Those modems that do use this signal generally use it to determine which mode to place the modem in. If the RI signal was present prior to the DTR signal, then the modem is placed in the answer mode. If the RI signal was not on prior to the DTR signal, then the modem is placed in the originate mode.

All other pins are undefined in their actions and no connections should be made to them to prevent possible malfunction of the modem.

Sometimes large amounts of data transfer is desired. With a 103 type modem limited to 300 BPS, this can get to be a bit tedious after a while. The 202 type modem serves as a compromise for this type of operation. By dropping one of the communication links, transmission speed can be boosted to 1200 BPS. This allows the data to be transmitted at a much faster rate, thereby improving the throughput.

Because of the lack of the other data link, however, the control of the communications flow becomes much more difficult. Since there is only one data link available, only one modem may transmit at a time (half duplex). This means that some sort of protocol must be decided upon to determine which modem may transmit. One of the more common ones is to send an ASCII "EOT" as the last character. This tells the receiving device that transmission is over and it may turn on its modem.

The operation of the control signals is very similar to the 103 type modem. Therefore only the differences will be discussed:

PIN 4	RTS	Request to send. This signal is used to turn the transmitter on and off.
PIN 5	CTS	Clear to send. This signal is generated from RTS, DCD, and an optional time delay and indicates when a valid transmission link has been established, and transmission may begin.
PIN 6	DCD	Data carrier detect. This signal is used to indicate that a carrier is being received.
PIN 12	SDC	Secondary data carrier detect. Optional reverse channel detector signal is present when reverse channel is present.
PIN 19	SRTS	Secondary request to send. (On some BELL 202's this is PIN 11) This signal turns the reverse channel carrier on and off (1 = on; 0 = off). If this PIN is tied high, then the reverse channel is controlled by the request to send (when request to send is on, reverse channel is off, and vice versa).

The reverse channel option is normally used as a circuit assurance or interrupt channel since, as long as the reverse channel is present, the transmitting modem can be assured that the data link is being maintained, and the receiving modem can use it to request an early termination of the transmission by turning it off.

The 202, like the 103, is quite flexible in its operation. It can be used at any speed up to its maximum allowable speed. It is transparent to most protocols as long as they are asynchronous. Some means must be provided, however, for turning the transmitter on and off.

Another modem that is becoming popular is the 212 modem. This modem combines the features of the 103 and 202. It has two data links (full duplex operation), yet can operate at 1200 BPS. It is, however, very limited in its operation. The transmission protocol is fixed, and the speed must be 1200 BPS exactly. For this you get full duplex operation, which means no transmitter control is required. Also, because of the transmission method used, it is inherently more difficult to build (i.e. more expensive). Transmissions between the two linked modems occur as DIBIT synchronous, and are then converted to asynchronous for transmission to and from the connected equipment. The PIN out of the 212 is the same as the 103 type modem. The operational mode (103 or 212) is determined by the signal applied to PIN 23 (0 = 103; 1 = 212). ©

KMMM Pascal for PET/CBM $85

A subset of standard Pascal with extensions.
- Machine language Pascal Source Editor with cursor oriented window mode
- Machine Language P-Code Compiler
- P-Code to machine language translator for optimized object code
- Run-time package
- Floating point capability
- User manual and sample programs

Requires 32K and disk. Please specify configuration.

EARL for PET (disk file based) $65

Editor, Assembler, Relocater, Linker

Generates relocatable object code using MOS Technology mnemonics. Disk file input (can edit files larger than memory). Links ,multiple object programs as one memory load. Listing output to screen or printer. Enhanced editor operates in both command mode and cursor oriented "window" mode.

RAM/ROM for PET/CBM

4K or 8K bytes of soft ROM with optional battery backup.

RAM/ROM is compatible with any large keyboard machine. Plugs into one of the ROM sockets above screen memory to give you switch selected write protectable RAM.

Use RAM/ROM as a software development tool to store data or machine code beyond the normal BASIC range. Use RAM/ROM TO LOAD A ROM image where you have possible conflicts with more than one ROM requiring the same socket. Possible applications include machine language sort (such as SUPERSORT), universal wedge, Extramon, etc.

RAM/ROM - - 4K	$85
RAM/ROM - - 8K	120
Battery Backup Option	30

SUPERSORT by James Strasma $35

Supersort is an excellent general purpose machine language sort routine for PET/CBM computers. Sorts both one and two dimensioned arrays at lightning speed in either ascending or descending order. Other fields can be subsorted when a match is found, and fields need not be in any special order. Sort arrays may be specified by name, and fields are random length. Allows sorting by bit to provide 8 categories per byte. The routine works with all PET BASICs, adjusts to any memory size, and can co-exist with other programs in high memory.

SuperGraphics

by John Fluharty **$30**

SuperGraphics provides machine language extensions to Commodore BASIC to allow fast and easy plotting and manipulation of graphics on the PET/CBM video display, as well as SOUND commands.

Animations that previously were too slow or impossible without machine language subroutines now can be programmed directly in BASIC. Move blocks (or rocketships, etc.), or entire areas of the screen with a single, easy to use BASIC command. Scroll any portion of the screen up, down, left, or right. Turn on or off any of the 4000 (8000 on 8032) screen pixels with a single BASIC command. In high resolution mode, draw vertical, horizontal, and diagonal lines. Draw a box, fill a box, and move it around on the screen with easy to use BASIC commands.

The SOUND commands allow you to initiate a note or series of notes (or even several songs) from BASIC, and then play them in the background mode without interfering with your BASIC program. This allows your program to run at full speed with simultaneous graphics and music.

SuperGraphics commands include GRAPHIC, TEXT, RVS, SET, DRAW, FILL, PLOT, MOVE, PRINT, CSET, CMOVE, DISPLAY, PUT, SWAP, PAUSE, and SOUND.

Please specify machine type and ROM version, disk or tape.

FLEX-FILE

for PET/CBM Computers

Self Calculating DATA BASE
REPORT WRITER
MAILING LIST

FLEX-FILE is a set of flexible, friendly programs to allow you to set up and mair .in a data base. Print files with a versatile Report Writer or a Mail Label routine. Programmers will find it easy to add subroutines to their own programs to make use of Data Base files.

RANDOM ACCESS DATA BASE

Record size limit is 250 characters. The number of records per disk is limited only by the size of each record and the number of records per disk is limited only by the size of each record and the amount of free space on the disk. File maintenance lets you step forward or backward through a file, add, delete, or change a record, go to a numbered record, or find a record by specified field. The Find command locates any record when you enter all (or a portion of) the desired key. Field lengths may vary from record to record to allow maximum packing of information. Files may be sorted by any field, and any field may be specified as a key. Sequential files from other programs may be converted to Flex-File format, and Flex-File records may be converted to sequential (WordPro, PaperMate, other word processors may also use Flex-File data). Maximum record size, fields per record, and order of fields may be changed at any time.

MAILING LABELS

With typical record size of 127 characters, each disk can handle over 1000 records (about 2800 with 8050 drive). Labels may be printed any number wide, and may begin in any column position. There is no limit on the number or order of fields on a label, and two or three fields may be joined together on one line (like first name, last name, and title). A "type of customer" field allows selective printing.

REPORT WRITER

Print any field in any column. For numeric fields, use decimal point justification (and round to any accuracy). Define any column as a series of mathematical functions performed on other columns. These functions include arithmetic operations and various log and trig functions. Pass results of operations such as running total from row to row. At the end of the report, print total and/or average for any column. Complete record selection, including field within range, pattern match, and logical functions can be specified individually or in combination with other parameters.

FLEX-FILE by Michael Riley $60

Please specify equipment configuration when ordering.

Low Cost Disk Drive for PET/CBM

PEDISK II from cgrs Microtech is a new disk system ready to plug into your large keyboard PET/CBM.

PEDISK II offers speed, reliability, IBM compatibility. Complete system prices with DOS and cable:

5" 40 track, 1 drive, 143K	$525
5" 40 track, 1 drive, 286K	690
8" IBM 3740 format, 77 track, 250K	995

PROGRAM YOUR OWN EPROMS

Branding Iron for PET/CBM $79

EPROM Programmer with software for all ROM versions. Includes all hardware and software to program or copy 2716 and 2532 EPROMs.

CBM Software

Legal Time Accounting Package	445
Medical Accounting Package	
Complete CBM Business Software Package	
Can be tailored to meet most business requirements.	
Technician's Investment Analysis Package	500
Dow Jones Portfolio Management	129
Personal Tax Calculator	65
Tax Preparation System	445
Wordcraft 80 Wordprocessor Package	325
Pascal Development Package	235
Assembler Development Package	99
Intelligent Terminal Emulator	30
Softpac-1 (Competitive Software)	29

FORTH for PET

BY L. C. Cargile and Michael Riley **$50**

Features include:
- full FIG FORTH model.
- all FORTH 79 STANDARD extensions.
- structured 6502 Assembler with nested decision making macros.
- full screen editing (same as when programming in BASIC).
- auto repeat key.
- sample programs.
- standard size screens (16 lines by 64 characters).
- 150 screens per diskette on 4040, 480 screens on 8050.
- ability to read and write BASIC sequential files.
- introductory manual.
- reference manual.

Runs on any 16K or 32K PET/CBM (including 8032) with ROM 3 or 4, and CBM disk drive. Please specify configuration when ordering.

Available soon:

Metacompiler for FORTH $30

simple metacompiler for creating compacted object code which can be executed independently (without the FORTH system).

PaperMate 60 COMMAND WORD PROCESSOR

by Michael Riley

Paper-Mate is a full-featured word processor for CBM/PET. Paper-Mate incorporates 60 commands to give you full screen editing with graphics for all 16K or 32K machines (including 8032), all printers, and disk or tape drives.

For writing text, Paper-Mate has a definable keyboard so you can use either Business or Graphics machines. Shift lock on letters only, or use keyboard shift lock. All keys repeat.

Paper-Mate text editing includes floating cursor, scroll up or down, page forward or back, and repeating insert and delete keys. Text block handling includes transfer, delete, append, save, load, and insert.

All formatting commands are imbedded in text for complete control. Commands include margin control and release, column adjust, 9 tab settings, variable line spacing, justify text, center text, and auto print form letter (variable block). Files can be linked so that one command prints an entire manuscript. Auto page, page headers, page numbers, pause at end of page, and hyphenation pauses are included.

Unlike most word processors, CBM graphics as well as text can be used. Paper-Mate can send any ASCII code over any secondary address to any printer.

Paper-Mate functions with 16/32K CBM/PET machines, with any printer, and with either cassette or disk.

To order Paper-Mate, please specify configuration.

Paper-Mate on disk or tape 40.00

Microchess	17.00
Checker King	17.00
Gammon Gambler	17.00
Time Trek	13.45
Bridge Partner	13.45
Visicalc	155.00

Hayden Software

Complex Mathematics	12.70
Engineering Mathematics	12.70
General Mathematics	12.70
MCAP:Circuit Analysis Program	21.00
Energy Miser	24.50

Jukebox Series for PET by L C Cargile
Excellent 4 part harmony music--write for list

Automated Simulations (EPYX) Fantasy Games

MICRO-REVERSI for PET by Michael Riley **10**
super machine language version of Othello

Tunnel Vision / Kat & Mouse by Michael Riley **10**
two excellent machine language maze programs

252 Bethlehem Pike
Colmar, PA 18915

215-822-7727

A B Computers

WRITE FOR CATALOG

Add $1.25 per order for shipping. We pay balance of UPS surface charges on all prepaid orders. Prices listed are on cash discount basis. Regular prices slightly higher. Prices subject to change.

More Machine Language For Beginners

Richard Mansfield
Assistant Editor

This article has two purposes: to provide a way of insuring that private documents and programs cannot be seen or used by unauthorized persons and to explain some aspects of machine language programming. Readers who are familiar with M.L. might wish to skip the second part of the article.

The BASIC Program

The BASIC listing of Security Lock (Program 1) will run on any version PET. The M.L. routine goes into a "safe" area in the second cassette buffer, common to all ROM sets in Commodore machines, including the new 8000 series. This area is "safe" because it is below BASIC programs and is not used by PET unless a *second* cassette machine is used.

The uses of "Security Lock" are explained within the program. It is not necessary to type in the entire program. Simply copy lines 120, 130, and the DATA lines from 1000 up.

The three-letter code can be changed, as described in the program, to any combination. An additional security measure — making it virtually impossible to break into a protected program — is not in the BASIC LISTing in Program 1. The reason that it cannot be illustrated is simple: the purpose of this technique is to *prevent LISTings themselves from taking place.*

We must describe how to do this since it cannot be demonstrated via a printout. First, when you include "Security Lock" within a program, you will be using a line similar to line 130 in Program 1 (REM statement removed). If you are calling the M.L. routine at the start of the program, you might type it in as line 1, thus — 1SYS867 (or, if you are not in the "graphics" mode, 1sys867).

Now, after the last character, type in a quotation mark and hit the RETURN key —: 1SYS867"

Then, using the cursor control keys, move the cursor back up to a position directly following the quotation mark. Holding the SHIFT key down, press the INSERT key nine times. Then release the INSERT and the SHIFT keys and press the DELETE key nine times. You will see nine reverse-character "t" 's which represent nine automatic deletions. Then press the RETURN key to enter this line into the rest of the program.

As you can see, any attempt to LIST the program will now delete line 1 from view, as if it were not part of the program. A brief flash on the screen is the only clue that something exists there, yet this line will operate normally during a RUN of the program. To eliminate the flash, you can use the quote/delete rub-out further into the program (as in line 130, Program 1) where it will be unlikely to be noticed.

Before turning to some observations on M.L. programming, it might be worthwhile to mention one modification to "Security Lock" which may prove useful. The M.L. program always prints "code?" on the screen to remind you that it is the Lock hanging up the program, not an endless loop or a hardware failure. If you simply want to freeze

... machine language routines can be listed in four ways ...

a program or file, without giving a clue as to why it's locked, eliminate the prompt word in the M.L. routine by typing in the following and then hitting RETURN:

```
FOR I=867 TO 880: POKE I,234: NEXT
```

This puts NOP (no operation) instructions into the routine, and when the SYS lands PET at 867, it slides up to 881 with no ill effects, where the input routine starts.

If you save frequently-used routines on a "Utilities" tape or disk for easy appending to future programs, this routine, like all M.L. routines, cannot be SAVEd normally (as BASIC is SAVEd). The following procedure will save M.L. routines which can later be LOADed in the usual way. Go into the Machine Language Monitor by typing SYS 1024. (If you have an Original PET, follow the instructions which came with your MLM tape.) Immediately after the dot, where the cursor should have landed, type —

```
s"Security Lock",01,035a,0399
```

and hit RETURN (for tape). For disk: .s"0:Security Lock",08,035a,0399. Note that the upper limit address must be one higher than the actual upper limit which was 0398 hex.

The Four Types Of M.L. Listings

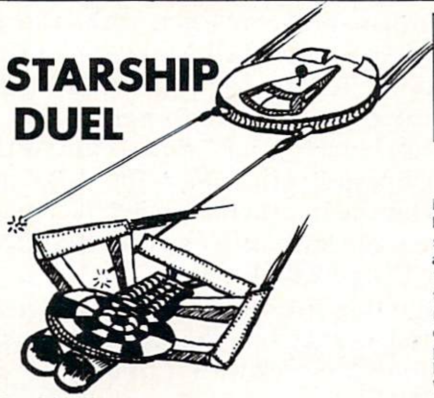
In books and magazines, machine language routines can be listed in four ways: as BASIC DATA statements (sometimes called a "BASIC loader") as a memory dump, as a simple disassembly, and as an annotated assembly. This can be confusing to the novice, so the four Programs which accompany this article illustrate the four kinds of listings possible for the same M.L. Program: Security Lock.

As a series of BASIC DATA statements (Pro-

THE PROGRAM STORE

CALL TOLL FREE 800 424-2738

STARSHIP DUEL



By John Kanopa
A two-player space game filled with action, graphics and sound. Each player starts with a fleet of starships and must destroy the opponent's fleet one-by-one. Sound easy? Not only can both ships fly around the screen, but when stopped they disappear. And if you run out of fuel during flight, your ship will explode! Four levels of difficulty will keep you occupied for many an exciting hour.

8K tape...$19.95

Q.S. FORTH

By James Albanese from Quality Software
Using FORTH programming language, you can produce programs that execute almost as fast as machine code but are much easier to write. A compiled language, this version is based on the microcomputer standard FIG-FORTH.

Included modules are: FORTH KERNAL (language for Atari), EXTENSION (added words for more power), EDITOR (to write and modify source code), IOCB (to easily set up I/O operations), and ASSEMBLER (compiles FORTH to assembler).

800, 24K 1-disk minimum...$79.95

The QUEST Vol.1

From Survival Software
The final holocaust is over,and you find yourself among the few survivors of a ravaged planet. Savagery seems to be most of the remaining people's key to survival, and just around every corner lurks a fate worse than death. Can you discover the secret to saving what's left of Earth's population? This is the first volume of a multi-part adventure for the Atari, enhanced with graphics and sound. Completion of the adventure will require the purchase of added volumes.

24K tape...$19.95 32K disk...$19.95

Programs for ATARI 400/800
Unless Otherwise Noted

A Warner Communications Company

TEXT WIZARD

By William Robinson from Datasoft
Full-featured, disk-based word processing system for the Atari. With TEXT WIZARD and a suitable printer (e.g. Atari 825/Centronics 737, Epson MX-80), you can manipulate text to your heart's desire. Once you have arrived at a final version, printing commands can be imbedded into the text. Your completed copy can then be printed -- with exactly the format specifications you desire. TEXT WIZARD requires at least one disk drive and 32K of RAM; for longer text files, 48K is desireable.

32K disk...$99.95

DUNGEON CAMPAIGN

By Clardy & Aldrich from Image
A delightful rendition of a Dungeons & Dragons scenario, full of colorful graphics and vivid sound. As you and your stalwart band of adventurers grope around the dungeon the walls in front, back and on both sides of you appear, forming an intricate maze. You must fight -- or buy off -- all manner of beasties in your quest to kill the dragon and escape with gold and jewels. Pterodactyls can lift you up and drop you in unlikely locations, poison gas can blind you, and serpents and spectres track you on your quest. Fun for all ages.

8K & 16K versions on tape...$24.95

BASIC A+

From OSS
Written by the authors of Atari Basic, BASIC A+ is fully compatible. You will be able to program much more easily and effectively with the many enhancements offered. With BASIC A+, you get full error messages, record I/O, fast binary I/O, TRACE function and direct access to player/missile graphics. Additional versatility is gained through such useful commands as PRINT USING, WHILE, and IF...ELSE. Get BASIC A+ for a whole new world of programming power and ease.

32K disk...$79.95

THE PROGRAM STORE
Franchises Available

TANK TRAP

By Don Ursem from Quality
Imagine the horror of a sinister tank following you, with the intent of running you down. As you move to escape the ominous form, you can build a wall to fend off the tank or even trap it. This is your plight in TANK TRAP. Played with keyboard or joystick input, you must trap the tank before it runs over you. There are four levels of difficulty, each one adding new complications to your effort.

16K tape...$12.95 32K disk...$14.95

IRIDIS 1

From The Code Works
More than a series of programs, IRIDIS is a continuing series of useful tutorials that will enhance your understanding of the Atari and your programming skills. This first volume includes: CLOCK (a graphic clock with moving hands and chimes), ZAP (a fast-moving chase game), and two graphics demo programs (LOGO and POLYGONS).

The comprehensive booklet accompanying IRIDIS 1 gives complete listings and a line-by-line explanation of each program. Additional features include articles of interest to all Atari users from novice to experienced "hackers." A real buy!

16K tape...$ 9.95
24K disk...$12.95

By Lance Micklus from Adventure Int.
An enhanced version of Micklus' famous STAR TREK III, designed to utilize the Atari's superior graphics. This program was considered best of all the "Treks" reviewed by ANALOG Magazine.

Your mission is to explore all 192 sectors of the Omega VI region of the galaxy. You must inspect all class M planets and destroy any evil Klingons found. STAR TREK 3.5 is a complex strategy game, made very realistic with its graphic displays and sound. Fun and challenging for novice through expert.

32K tape (disk compatible)...$19.95

Visit Our Baltimore, MD Store: W. Bell Plaza – 6600 Security Blvd

VISA master charge **TO ORDER CALL TOLL FREE 800 424-2738** For information Call (202) 363-9797

MAIL ORDERS: Send check or M.O. for total purchase price, plus $1.00 postage & handling. D.C. residents, add 6% tax. Charge card customers: include all embossed information on card.

Prices Subject to Change

THE PROGRAM STORE
4200 Wisconsin Avenue NW, Dept. U10 Box 9609
Washington, D.C. 20016

gram 1), M.L. code is a part of a larger BASIC program. This gives little or no information about the nature of the M.L. section. It is typed, then RUN, as a subroutine of the host program. And the reader is frequently cautioned to type the program *exactly* as it appears. This is because a single error in M.L. will usually crash the entire program. But much typing time can be saved, if the M.L. routine is all that's wanted from the BASIC program, by looking for three things: a READ-loop, a SYS, and DATA statements. Line 120 is a READ-loop which POKEs the M.L. routine into memory and line 130 contains the SYS to enter that routine at the proper address. In BASIC, the DATA for an M.L. routine are decimal numbers.

The next step up toward clarity, though the program's meaning is still not easily recognizable, is a "memory dump" (Program 2). (This is sometimes called a "hex dump.") It is a table of hexadecimal numbers. The first number is the address of the first datum on its line. In Program 2, the "dump" shows that address 035a contains a 43, address 035b contains 4f, etc. As before, to make such a program your own, you copy in the information, being careful to copy precisely. In this case, however, you must first enter the M.L. Monitor and then type:

.M 035a 0392 (RETURN key)

This will put a memory dump on screen of what currently exists in these memory cells. To put in the new data, you just type over what appears on the screen, observing the spaces between each hexadecimal number and hitting RETURN when each line has been changed.

A third type of M.L. printout is a list of each machine language instruction in terms of its function. This is a *disassembly*, (Program 3), and resembles a LISTing in BASIC, though in a highly abbreviated form. Any series of numbers can be examined by a *disassembler*, a program which translates raw data into M.L. instruction mnemonics. A disassembler can be found on pg. 81 of **COMPUTE!** #8. If the numbers are part of an M.L. routine, the disassembler will list them as in Program 3. If it cannot make sense of what it sees (if it were examining memory which contained BASIC code for example) it would print a series of question marks.

A disassembler usually prints out four *fields*, or zones of information. It is easy to see that the first four characters in Program 3 represent memory addresses. This is the "address field" and is similar to the first four characters of Program 2, the memory dump, except that here the number of bytes in the second field, the "data field," can be 1, 2, or 3 — so the numbers in the address field will increase irregularly. The second, "data field," also corresponds to Program 2's dump, but there is the same irregularity as different numbers group themselves together. This grouping is then *translated* in the third and fourth fields — the "instruction" field and the "operator" field. These last two fields are "mnemonic" (easy to remember) representations of the information contained in the raw hex numbers of the "data field" which precedes them. The "instructions" tell the computer what to do and the "operators" tell the computer what to do it *to*. In the phrase, "drive a car," *drive* is the instruction, *car* is the operator. In LDY #$00, LDY (load the Y register) is the instruction, #$00 (zero) is the operator. The same structure exists in BASIC — POKE 32768,41 or PRINT "Hello."

The reason that the disassembly must group its information irregularly is that different instructions are designed to work with different sized operators. INY, (increase the value of the Y register by 1), has no explicit operator since the "1" is implied within the instruction itself. LDY #$00 has a one-byte operator, 00, so it is two bytes long: LDY and 0. To instruct the computer to compare the number in the accumulator with the number in address $0360, we need three bytes, CMP plus two bytes to represent a number as large as 0360. Any one byte can only hold a number up to 255.

Full Source Code

Finally, Program 4 illustrates the clearest way that an M.L. program can be presented: as an annotated assembly listing. (It is also called "source code.") This contains within it the four fields of the disassembly, but adds three more fields — line numbers, labels, and comments.

Such listings represent the program rather elaborately by M.L. standards. Such programs are written using an "assembler" program which accepts mnemonics such as INY, translates them, and puts them in memory. Assemblers are either "single-pass," (simple translators of mnemonics) or complex, label-oriented powerhouses. Unfortunately, the M.L. Monitor within PET does not contain a disassembler or an assembler, but the monitor can be made to include these functions (and others) with a program such as "Supermon" or "Extramon." For short routines, simple assemblers will work well. For larger jobs — an entire arcade game would be a large job — a power assembler is needed. To my knowledge, the most advanced assemblers available to PET users are "MAE" and "ASM/TED," written by Carl Moser. (Available from A.B. Computers or Eastern House Software.)

A printout from such an assembler is much easier to understand because it contains labels and comments. Using such an assembler, it is also easier to write large programs since some of the problems associated with programming in M.L. are handled automatically by the computer.

Mnemonics are easier to manipulate than numbers, but whole words (labels) are often an improvement over mnemonics, particularly when a

program is lengthy. To clarify the additional fields found in large assembly programs, we can examine line 0180 (Program 4). It begins with two fields which are identical to the disassembly in Program 3, but the third field is a BASIC-like consecutive numbering of each line of the program. This allows the programmer to manipulate the instructions more easily, since renumbering can open up new space for additional instructions or whole sections of the program can be conveniently rearranged.

Following the line number is the label field, in this case the label, "START," since it is the beginning of the program. This is better than BASIC. Several locations (lines 350, 380, 410) are able to say IF THEN GOTO *START*, where a comparable BASIC program would need to use the line number instead of a word: IF THEN GOTO 180. This relieves the programmer of having to look up line addresses for his subroutines or major entry points as well as eliminating a frequent cause of errors.

The next two fields are the instruction and operator fields (as in a disassembly) except that now some of the operators have been replaced by labels. If, as in line 190, we see LDA (load accumulator) *TEXT* we can find the value or meaning of the label, TEXT, in three ways. We can look over to the second field, where the raw numbers are, we can look earlier in the program where TEXT is defined (line 80), or we can look at the end of the program in the "Label File." TEXT refers to data which starts at address $035A and which line 80 defines as being the word "code?"

The last field holds comments which describe the function of the line where they appear (and sometimes, subsequent lines). The semi-colon is the same as REM — anything which follows it serves to document the program, but is ignored by the assembler. This makes later modifications easier, debugging faster, and also helps to reveal the meaning of the program to others.

To review, we see a progress toward clarity, from Program 1 to Program 4, largely due to the addition of new fields of information. Program 1 contains a single field: decimal data. Program 2 adds an address field. Program 3 adds fields three and four — a translation into instruction mnemonics and operators of the raw data from field two. And Program 4 adds line numbers, labels and comments — for a total of seven fields. We have now examined the *horizontal* organization of a M.L. program, from its simplest form to its most complex. Using the most complex example, (Program 4), let's twist ourselves sideways and go on to investigate the *vertical* organization of M.L. programs.

The Four Parts Of A Computer Program

All programs — in fact, all thinking — can be broken down into four essential parts: 1. Initialization and Protection, 2. Data Tables, 3. Main Loop, 4. Subroutines. Before learning a new word (thinking), a person must: 1. not be being shot to death, 2. have a dictionary, 3. start looking up the word, and 4. move his thumbs correctly, know or guess the spelling, keep his balance, etc. The order of these elements is important. Without protection, any M.L. routine between addresses 1024 and the screen RAM at 32768 can be overwritten by a BASIC program either by a LOAD or because BASIC puts some of its variables up at the top of available RAM where M.L. programmers like to stick routines.

Protection can be achieved by telling PET that its memory size has shrunk — changing the numbers in addresses 52 and 53 (134,135, in Original ROMs). Then all BASIC activity will be confined to RAM below the address resulting from (PEEK (52) + PEEK (53)*256). Or, a short M.L. routine can be nestled into a space where BASIC doesn't usually go, such as the second cassette buffer. BASIC protects itself, so that is not of concern in BASIC programming.

A *table* is a collection of information (data) which the program will need. In Program 4, and in M.L. generally, the tables are placed at the beginning of the program (but sometimes at the end). It is good to get into a habit of keeping tables together and putting them at the start. In line 80, instead of an ordinary mnemonic, we have a pseudo-op, .BY, (pseudo-ops are preceded by a period). A pseudo-op is a request to the assembler program to perform some task for the programmer. In this instance, the programmer is requesting that an ASCII word, *CODE?*, be translated into bytes and stored to be used by the program later. Line 90 contains the pseudo-op, .DE, which defines the label, *SENDCHAR*, as the address in BASIC ROM which prints a character to the screen. The .DS in line 110 tells the assembler to define some storage space, three cells large, called *STORAGE* which the program will later use as a place to hold the codeword PET.

A main loop is a series of steps which control the program as a whole. It is distinct from subroutines in that it *calls* subroutines, they do not call the main loop. In a complicated M.L. program, the main loop can be a series of JSR (Jump to Subroutine) instructions which defines the order in which subroutines are performed. In BASIC, it can take the form of an ON GOTO list of addresses, a series of GOSUBS, or a loop. In simpler programs, the main loop is often merely implicit — each subroutine is already arranged within the program in the desired order of execution. The program runs more or less sequentially from start to finish. In such cases, a governing loop is only implied.

In Program 4, the instructions break into two divisions: initialization and subroutine. Since it is a simple program, there is only a fragment of what would be a main loop in a larger program. The initialization zone is often at the start of a main loop, and sets up whatever preconditions the pro-

gram will later expect (including protection). In this case, the word *code?* must be printed to the screen until the correct code is entered and the loop can be exited.

The phrase "cold start" refers to an entrance into a program at the very beginning of the initialization section. This will reset all flags, pointers, counters, etc. to their virgin condition. A "warm start" enters the mail loop beyond initialization, so that various kinds of information, modified during a program RUN, are left undisturbed. There is some ambiguity to these terms since initialization is sometimes unnecessary, or is sometimes refreshed on every entrance to the main loop (the warm start and the cold start would then be identical), or other anomalies. It is valuable, however, to develop a sense that a program has two distinct active parts and one passive part. The main loop *governs* the action of the subroutines. Data tables are passive zones of information which perform no tasks. And, before all else, an M.L. program must protect itself from a BASIC invasion. (M.L. can also require protection from interruptions, but this concept is outside of the purview of this article.)

How Security Lock Works

The main loop begins at line 180 (Program 4) and sets the Y register to zero so that it can act as an "offset" to the address called "TEXT" (a little table holding the word "CODE?"). This is much simpler than it sounds. "Offset" means *add this number to a fixed number.* TEXT has already been defined as a fixed address (035A) which is the start of the table holding the word "CODE?" So, in line 190, we LDA (load the accumulator register, a temporary resting place for bytes of data) with whatever is in the address TEXT + Y. Since Y was just loaded with a zero (LDY #0), the byte that we are putting into the accumulator will be at $035A itself. Line 200 tests to see if the whole word, CODE?, has been printed. BEQ means branch if you just loaded (LDA) a zero into the accumulator. But we didn't. Since address 035A has a 43 in it, now the accumulator also has the 43. We will only branch (go somewhere else) if it's a zero. So the branch is ignored and we continue to line 210 which jumps to a subroutine (JSR) in BASIC ROM which puts the character in the accumulator on the screen. The 43 (letter "C") will then appear and the control of the computer is returned to line 220, as in a BASIC RETURN command. Line 220 increases (increments) the number in the Y register by 1 (INY). It was a zero, so now it's a 1. Then, like a GOTO, line 230 jumps (JMP) to the line we've labeled LOOP (line 190) where the value of TEXT and Y are again added together to give the address where we will find what to put into the accumulator. This time, however, since Y now equals 1, the "effective" address is 035B, where the letter "O" is waiting to be picked up. After looping this way for a while, increasing the address each time by increasing the value of Y, we will eventually pick up a zero which we thoughtfully placed in address 035F, the end of our TEXT table (see line 80). This is a "delimiter" to let the loop know that we are finished and that it should now BEQ (branch if equal to zero) to COMPARE, line 270.

Here we load the accumulator with the code letters and put them into the previously defined (line 110) storage area within our tables. This time, rather than setting up a loop and a delimiter, we add the offset directly to the labels: STORAGE + 1, STORAGE + 3. At line 330 we again jump to subroutine in BASIC ROM which will input a single letter from the human and leave it in the accumulator, returning from the subroutine to line 340. Here, we compare (CMP) this letter in the accumulator with the first letter in the STORAGE zone, a "P." If the accumulator does not match "P," then the instruction in line 350 (BNE, branch if not equal) takes effect and the computer is thrown back to START. If it was equal, we "fall through" to line 360 where the same comparison is done for "E." Any failure of equality causes a branch to START. If all three letters match, the instruction RTS (return from subroutine) puts us back into BASIC just beyond the SYS which threw us into the M.L. routine in the first place. SYS is merely a GOSUB to M.L. subroutines.

Program 1: BASIC Loader

```
100 REM                    SECURITY LOC~
   ~K BY RICHARD MANSFIELD
110 POKE59468,14:PRINT"{CLEAR}";
120 FORI=858TO920:READR:POKEI,R:NEXT
130 REM  SYS 867
140 REM                      /ON-SCREE~
   ~N INSTRUCTIONS/
150 PRINT"{02 DOWN}     {REV} THIS IS ~
   ~A SECURITY LOCK "
160 PRINT"{05 DOWN}FROM 858 TO 920 IN~
   ~ YOUR MEMORY
170 PRINT"IS A MACHINE LANGUAGE PROGR~
   ~AM WHICH
180 PRINT"WILL NOT ALLOW THE PROGRAM ~
   ~TO PROCEED
190 PRINT"UNTIL THE WORD 'PET' IS TYP~
   ~ED."
200 PRINT
210 PRINT"{DOWN}{05 RIGHT}THIS CAN BE~
   ~ USED TO INSURE THAT
220 PRINT"A FILE OR DIARY IS PROTECTE~
   ~D FROM
230 PRINT"PRYING EYES.
240 PRINT"   IN ORDER TO DEMONSTRATE ~
   ~THIS...
250 PRINT"SIMPLY ELIMINATE THE REM
260 PRINT"FROM LINE 130 AND THEN TRY ~
   ~TO RUN
270 PRINT"THE PROGRAM WITHOUT TYPING ~
   ~IN 'PET'
280 PRINT"{DOWN}{16 RIGHT}PRESS ANY K~
   ~EY"
```

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?

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.

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
```

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

```
290 GOSUB71Ø
300 GETA$:IFA$=""THEN3ØØ
310 PRINT"{CLEAR}"
320 PRINT"{DOWN}{Ø5 RIGHT}OF COURSE, ~
   ~THEN THEY COULD
330 PRINT"TYPE LIST, TO SEE YOUR PROG~
   ~RAM,
340 PRINT"AND NOTICE THAT SUSPICIOUS ~
   ~SYS YOU'VE
350 PRINT"GOT AT THE BEGINING...AND T~
   ~HEN,
360 PRINT"WITH ENOUGH WORK, EXTRACT T~
   ~HE
370 PRINT"CODE WORD FROM THE DATA STA~
   ~TEMENTS.
380 PRINT"BUT THE ACCOMPANYING ARTICL~
   ~E GOES
390 PRINT"FURTHER YET.  IT DEMONSTRAT~
   ~ES A WAY
400 PRINT"TO EVEN HIDE THE SYS STATEM~
   ~ENT FROM
410 PRINT"ANY LISTING OF THE PROGRAM.~
   ~
420 PRINT"{DOWN}    WHAT'S MORE, YOU C~
   ~AN GO STILL
430 PRINT"DEEPER INTO A PARANOIAC'S W~
   ~ONDERLAND
440 PRINT"BY USING THIS LIST-PREVENTI~
   ~ON TRICK
450 PRINT"TO ELIMINATE ANY OTHER PART~
   ~S OF A
460 PRINT"PROGRAM FROM LISTINGS...INC~
   ~LUDING
470 PRINT"THE DATA STATEMENTS THEMSEL~
   ~VES.
480 PRINT"ANYBODY WHO CAN GET TO YOUR~
   ~ DIARY
490 PRINT"AFTER ALL THIS SECRECY IS S~
   ~O
500 PRINT"BRILLIANT THAT THEY DESERVE~
   ~ TO
510 PRINT"READ IT!
520 PRINTSPC(19);"ANY KEY"
530 GETA$:IFA$=""THEN53Ø
540 PRINT"{CLEAR}   SINCE 'PET' IS PE~
   ~RHAPS AN OBVIOUS
550 PRINT"CODE, YOU MAY USE ANY OTHER~
   ~ 3 LETTERS
560 PRINT"BY POKEING DIFFERENT VALUES~
   ~ INTO ADDRS:
570 PRINT"882, 887, & 892.  (A=65...Z~
   ~=9Ø)
580 PRINT"{DOWN}"SPC(19);"ANY KEY"
590 GETA$:IFA$=""THEN59Ø
600 PRINT"{CLEAR}{Ø3 DOWN}     ANOTHER~
   ~ USE WOULD BE TO LOCK
610 PRINT"UP YOUR PET WHEN YOU ARE NO~
   ~T THERE.
620 PRINT"SIMPLY LEAVE THE SECURITY P~
   ~ROGRAM
630 PRINT"IN ITS LOCATION (NO PROGRAM~
   ~ YOU WRITE
640 PRINT"WILL DISTURB IT, NOT EVEN T~
   ~YPING NEW)
650 PRINT"WHEN YOU WANT TO LOCK, JUST~
   ~ TYPE SYS867
660 PRINT"(THE () ARE UNNECESSARY) AN~
   ~D WALK
670 PRINT"AWAY...NOBODY CAN GET INTO ~
   ~THE PROGRAM
680 PRINT"EXCEPT YOU.
690 PRINT"{DOWN}  NOW, PLEASE REMOVE ~
   ~THE REM FROM 13Ø
695 PRINT"AND THEN RUN THE DEMONSTRAT~
   ~ION."
700 LIST 13Ø:REM      ------ START/GRAP~
   ~HICS ------
710 FORI=33767TO32925STEP-4Ø:POKEI,3Ø~
   ~:POKEI+4Ø,32:FORB=1TO95:NEXTB:NEX~
   ~TI
720 FORI=32927TO32917STEP-1:POKEI,31:~
   ~POKEI+1,32:FORT=1TO2:NEXTT:NEXTI
730 FORI=32917TO32925:POKEI+1,31:POKE~
   ~I,32:FORT=1TO5Ø:NEXTT:NEXTI:X=X+1~
   ~
740 IFX<3THEN72Ø
750 POKEI,32
760 FORI=32925TO33767STEP4Ø:POKEI,3Ø:~
   ~POKEI-4Ø,32:FORT=1TO1ØØ:NEXTT:NEX~
   ~TI
770 REM                ------  END/GRAP~
   ~HICS  ------
780 RETURN
1000 DATA 67,79,68,69,63,Ø,Ø,Ø,Ø
1100 DATA 16Ø,Ø,185,9Ø,3,24Ø,7,32,21Ø~
   ~,255,2ØØ
1200 DATA 76,1Ø1,3,169,8Ø,141,96,3,16~
   ~9,69
1300 DATA 141,97,3,169,84,141,98,3,32~
   ~,2Ø7
1400 DATA 255,2Ø5,96,3,2Ø8,219,32,2Ø7~
   ~,255
1500 DATA 2Ø5,97,3,2Ø8,211,32,2Ø7,255~
   ~
1600 DATA 2Ø5,98,3,2Ø8,2Ø3,96
READY.
```

Program 2: Memory Dump

```
.
.:  035a 43 4f 44 45 3f 00 00 00
.:  0362 00 a0 00 b9 5a 03 f0 07
.:  036a 20 d2 ff c8 4c 65 03 a9
.:  0372 50 8d 60 03 a9 45 8d 61
.:  037a 03 a9 54 8d 62 03 20 cf
.:  0382 ff cd 60 03 d0 db 20 cf
.:  038a ff cd 61 03 d0 d3 20 cf
.:  0392 ff cd 62 03 d0 cb 60 58
.
```

Program 3: Disassembly

```
.
035a 43          ???
035b 4f          ???
035c 44          ???
035d 45 3f       eor $3f
035f 00          brk
0360 00          brk
0361 00          brk
0362 00          brk
0363 a0 00       ldy #$00
0365 b9 5a 03    lda $035a,y
0368 f0 07       beq $0371
036a 20 d2 ff    jsr $ffd2
036d c8          iny
036e 4c 65 03    jmp $0365
0371 a9 50       lda #$50
0373 8d 60 03    sta $0360
0376 a9 45       lda #$45
0378 8d 61 03    sta $0361
037b a9 54       lda #$54
037d 8d 62 03    sta $0362
0380 20 cf ff    jsr $ffcf
0383 cd 60 03    cmp $0360
0386 d0 db       bne $0363
0388 20 cf ff    jsr $ffcf
038b cd 61 03    cmp $0361
038e d0 d3       bne $0363
0390 20 cf ff    jsr $ffcf
0393 cd 62 03    cmp $0362
0396 d0 cb       bne $0363
0398 60          rts
.
```

Program 4: Assembler Source Code

```
                0010 ; -------- SECURITY LOCK ASSEMBLY -- MAE ----------
                0020 ;
                0030                .BA 858           ;START ASSEMBLY AT ADDRESS
                0040                .OS               ;PUT ASSEMBLED CODE THERE.
                0050 ;
                0060 ; --------------   T A B L E S   ---------------
                0070 ;
035A- 43 4F 44  0080 TEXT           .BY 'CODE?' 0 ;DEFINE TEXT
035D- 45 3F 00
                0090 SENDCHAR       .DE $FFD2         ;OUTPUT CHARACTER
                0100 FINDCHAR       .DE $FFCF         ;GET CHARACTER INPUT
0360-           0110 STORAGE        .DS 3
                0120 ;
                0130 ;
                0140 ;          ******* START OF INSTRUCTIONS *******
                0150 ;
                0160 ;### INITIALIZATION ###
                0170 ;
0363- A0 00     0180 START          LDY #0            ;INITIALIZE COUNTER
0365- B9 5A 03  0190 LOOP           LDA TEXT,Y        ;GET A LETTER
0368- F0 07     0200                BEQ COMPARE       ;IF ZERO, TEXT FINISHED
036A- 20 D2 FF  0210                JSR SENDCHAR      ;PRINT ON SCREEN
036D- C8        0220                INY               ;INCREASE COUNTER
036E- 4C 65 03  0230                JMP LOOP          ;GET NEXT CHARACTER
                0240 ;
                0250 ;### SUBROUTINE ###
                0260 ;
0371- A9 50     0270 COMPARE        LDA #$50          ;PUT   "P"   INTO STORAGE
0373- 8D 60 03  0280                STA STORAGE
0376- A9 45     0290                LDA #$45          ;      "E"
0378- 8D 61 03  0300                STA STORAGE+1
037B- A9 54     0310                LDA #$54          ;      "T"
037D- 8D 62 03  0320                STA STORAGE+2
0380- 20 CF FF  0330                JSR FINDCHAR      ;GET A CHARACTER FROM HUMA
0383- CD 60 03  0340                CMP STORAGE       ;IS IT A  "P" ?
0386- D0 DB     0350                BNE START         ;NO? START OVER AGAIN
0388- 20 CF FF  0360                JSR FINDCHAR
038B- CD 61 03  0370                CMP STORAGE+1     ;IS IT AN "E" ?
038E- D0 D3     0380                BNE START
0390- 20 CF FF  0390                JSR FINDCHAR
0393- CD 62 03  0400                CMP STORAGE+2     ;          "T" ?
0396- D0 CB     0410                BNE START
0398- 60        0420                RTS               ;CORRECT! GOTO BASIC.
                0430                .EN               ;END OF ASSEMBLY
ENDPASS
```

©

HUNTINGTON COMPUTING

PROGRAMS LISTED BELOW ARE ON DISK:

EDUCATION

Compu-Math: Arithmetic	$49.95 now	**$44.99**
The Linguist	$40.00 now	**$33.99**
Serendipity Grade Book	$169.00 now	**$143.59**
Dynacomp's Teacher Pet	$14.95 now	**$12.69**
Spanish Hangman	$29.95 now	**$26.99**
French Hangman	$29.95 now	**$26.99**
Clock (Hartley)	$29.95 now	**$26.99**
Dr. Daley's Software Library	$79.95 now	**$69.99**
Physics I	$24.00 now	**$21.49**
J&S Chemistry	$150.00 now	**$139.99**
J&S Chemistry/Biology Demo		**$5.00**
Cook's Touch Typing	$14.95 now	**$13.49**
Mastertype	$34.95 now	**$29.69**
Compak Math Grade by Concept	$495 now	**$445.49**
Statistics (Edu-Ware)	$29.95 now	**$25.39**
Algebra I (Edu-Ware)	$39.95 now	**$33.99**
Microsoft Typing	$19.95 now	**$16.99**
Basic Tutor 1-8 (each)	$32.00 now	**$28.79**
Milliken Math Sequences	$450.00 now	**$405.00**
Compupoem		**$15.95**
The Learning System	$150.00 now	**$127.49**
Snake Eggs II	$24.95 now	**$21.19**
Android Nim	$24.95 now	**$21.19**
Hartley Cass. Control Device	$79.95 now	**$71.95**
Alicia-Bilingual Reader	$29.95 now	**$25.39**
EAI Critical Reading	$39.95 now	**$35.99**
EAI Reading Comprehension	$39.95 now	**$35.99**
Chemistry (Ed. Courseware)	$32.00 now	**$28.79**
American History	$32.00 now	**$28.79**
Spelling (Ed. Courseware)	$32.00 now	**$28.79**
Astronomy	$32.00 now	**$28.79**
Biology	$32.00 now	**$28.79**
Statistics 3.0	$29.95 now	**$25.39**
Compu-Read 3.0	$29.95 now	**$25.39**
ESP		**$12.69**
Compu-Math Decimals	$39.95 now	**$33.89**
Compu-Math Fractions	$39.95 now	**$33.89**
Prescriptive Math (Hartley)	$79.95 now	**$71.99**
Word Search (Hartley)	$14.95 now	**$13.49**
Letter Recognition	$19.95 now	**$17.99**
Spelling Bee (Edu-Ware)	$29.95 now	**$25.39**
Dracula (Pendulum Press)	$34.95 now	**$29.69**
Moby Dick (Pendulum Press)	$34.95 now	**$29.69**
Time Machine (Pendulum Press)	$34.95 now	**$29.69**
Black Beauty (Pendulum Press)	$34.95 now	**$29.69**
PDI Analogies	$23.95 now	**$20.29**

UTILITIES

Visicalc Utility #3	$129.95 now	**$99.99**
Program Line Editor	$40.00 now	**$34.99**
Super Kram	$175.00 now	**$143.59**
Super Disk Copy III	$30.00 now	**$25.39**
Hayden AS Compiler	$200.00 now	**$159.00**
Expediter II	$99.95 now	**$84.99**
Pascal Programmer	$125.00 now	**$106.19**
S-C Assembler II (V. 4.0)	$55.00 now	**$46.69**
Verscalc	$100.00 now	**$84.99**
Quicktrace	$35.00 now	**$29.69**
Tiny Pascal (Programma)	$49.95 now	**$42.89**
Tiny Pascal Plus + (Abacus)	$50.00 now	**$42.49**
Apple-Cillin	$29.95 now	**$25.39**
Micro Apple 1 (incl. Book)	$24.95 now	**$22.99**
Pascal Graphics Editor	$114.95 now	**$97.69**
P.L.E. in ROM	$60.00 now	**$50.99**
Disk Fixer	$28.95 now	**$24.39**
Super Sound	$16.95 now	**$14.39**
Memory Management System	$39.95 now	**$33.89**
Super Disk Copy (Sensible)	$29.95 now	**$25.39**
Disk Recovery	$29.95 now	**$25.39**
Multi-Disk Catalog	$24.95 now	**$21.19**
Double Dos Plus	$24.95 now	**$21.19**
Aplus	$24.95 now	**$21.19**
Disk Organizer	$24.95 now	**$21.19**
Quickloader	$24.95 now	**$21.19**
Apple Doc	$34.95 now	**$29.59**
Z-Term	$99.95 now	**$84.99**
Ace	$29.95 now	**$25.39**
Co-op Complete Graphics Sys	$59.95 now	**$50.89**
P-Lisp Interpreter (Pegasys)	$99.95 now	**$89.89**
Lisa	$49.95 now	**$42.39**
Apple Forth (Programma)	$49.95 now	**$42.39**

WORD PROCESSORS

Apple Writer		**$65.99**
Magic Window	$100.00 now	**$84.99**
Easy Writer Professional	$250.00 now	**$219.00**
Letter Perfect	$150.00 now	**$127.49**
Super Text	$150.00 now	**$127.49**
Superscribe	$129.95 now	**$110.39**
Apple Pie (All Versions)	$129.95 now	**$99.99**
Executive Secretary	$250.00 now	**$212.49**
Apple Wordstar	$375.00 now	**$286.00**
Hebrew II	$60.00 now	**$50.99**
Apple Writer Extended	$29.95 now	**$25.39**

GAMES

Hi-Res Soccer	$29.95 now	**$25.39**
Apple-Oids	$29.95 now	**$25.39**
Wurst of Huntington Computing		**$19.99**
Gobbler	$24.95 now	**$21.19**
Ultima	$39.95 now	**$33.89**
Autobahn	$29.95 now	**$25.39**
Battle Cruiser Action	$44.95 now	**$38.19**
Gorgon	$39.95 now	**$33.89**
Super Stellar Trek	$39.95 now	**$33.89**
Hellfire Warrior	$39.95 now	**$33.99**
Gamma Goblins	$29.95 now	**$25.39**
Mission Asteroid	$19.95 now	**$17.99**
Wizardry	$49.95 now	**$42.49**
Star Mines	$29.95 now	**$25.39**
Warp Factor	$39.95 now	**$33.99**
Microsoft Adventure	$29.95 now	**$25.39**
Wizard and the Princess	$32.95 now	**$28.99**
Flight Simulator	$34.95 now	**$29.49**
Odyssey	$29.95 now	**$25.39**
Sargon II	$34.95 now	**$29.69**
Space Eggs	$29.95 now	**$25.39**
Hi-Res Cribbage	$24.95 now	**$21.19**
Lords of Karma (cass.)	$20.00 now	**$16.99**
Oh Shoot	now	**$19.99**
ABM	$24.95 now	**$21.19**
Computer Conflict	$39.95 now	**$35.99**
Computer Air Combat	$59.95 now	**$52.99**
Temple of Apshai	$39.95 now	**$33.89**
Zork	$39.95 now	**$33.99**
All Nibble Software		**15% off list**
Robot Wars	$39.95 now	**$33.99**
Cranston Manor	$34.95 now	**$29.69**
Dragon's Eye	$24.95 now	**$21.19**
Computer Acquire	$20.00 now	**$16.99**
Twala's Last Redoubt	$29.95 now	**$25.39**
Snoggle	$24.95 now	**$21.19**
Alien Rain	$24.95 now	**$21.19**
Alien Typhoon	$24.95 now	**$21.19**
Raster Blaster	$29.95 now	**$25.39**
3-D Skiing	$24.95 now	**$21.19**
Creature Venture	$24.95 now	**$21.19**
Galaxy Space War I	$39.95 now	**$33.99**
Hodge Podge	$23.95 now	**$21.29**
Meteoroids in Space	$19.95 now	**$16.99**
Dragon Fire	$49.95 now	**$42.49**
Pool 1.5	$34.95 now	**$29.69**
Orbitron	$29.95 now	**$25.39**
Galactic Attack	$29.95 now	**$25.39**
Swordthrust I	$29.95 now	**$25.39**
Swordthrust II	$24.95 now	**$21.19**
Swordthrust III	$24.95 now	**$21.19**
Enchanted Island/Atlantis	$21.95 now	**$18.59**
Original Adventure	$24.95 now	**$21.19**
Mad Venture	$24.95 now	**$21.19**
Crown of Arthane	$35.00 now	**$29.69**
Orbitron	$29.95 now	**$25.39**
Pulsar II	$29.95 now	**$25.39**
Sneakers	$29.95 now	**$25.39**
Invasion Force	$23.95 now	**$20.29**
Wizard's Mountain	$16.95 now	**$14.39**
Kaves of Karkhan	$49.95 now	**$42.39**
I.F. Six Micro Stories	$14.95 now	**$12.69**
Milestones	$19.95 now	**$16.89**
Inferno	$29.95 now	**$25.39**
A2-3D/A Saturn Navigator	$24.95 now	**$21.19**
Black Hole	$18.95 now	**$16.09**
The Shattered Alliance	$59.95 now	**$50.99**
Space Warrior (Broderbund)	$24.95 now	**$21.19**
OO-Topos	$32.95 now	**$27.99**
Monty Plays Scrabble	$39.95 now	**$33.89**
Poker Tournament	$20.95 now	**$17.79**
Star Thief	$29.95 now	**$25.39**
Apple Panic	$29.95 now	**$25.39**

VISIDEX
List $200.00
now $139.00
(Offer expires 12-30-81)

STOCK PROGRAMS

Portfolio Master	$75.00 now	**$63.69**
Market Charter	$129.95 now	**$110.39**
Dowlog for Market Charter	$99.95 now	**$89.99**
Investment Decisions	$99.95 now	**$84.99**
Stock Tracker	$190.00 now	**$161.49**
Stock Tracker (Auto. Ver.)	$300.00 now	**$254.99**

BUSINESS APPLICATIONS

Program	List	Now
Invoice Factory (Special)	$200.00 now	**$149.00**
Regression Trend Analysis	$26.95 now	**$22.89**
Multiple Regression	$29.95 now	**$25.39**
Microsoft Fortran	$200.00 now	**$165.69**
Microsoft Cobol-80	$750.00 now	**$637.49**
Business Pac 100	$99.95 now	**$84.99**
Desktop Plan II	$200.00 now	**$169.99**
Visicalc 3.3 Special	$200.00 now	**$149.00**
Visiplot	$179.95 now	**$152.89**
Visitrend/Visiplot	$259.95 now	**$220.89**
Visiterm	$149.95 now	**$127.39**
Complete Mailing (Avant-Garde)	$59.95 now	**$50.89**
DB Master	$229.95 now	**$194.59**
PFS	$95.00 now	**$80.69**
PFS Report	$95.00 now	**$80.69**
Data Factory	$150.00 now	**$124.00**
Thinker	$495.00 now	**$420.69**
Request	$225.00 now	**$191.19**
Super Kram	$175.00 now	**$148.69**
ASCII Express	$99.95 now	**$84.99**
BPI Accounts Receivable	$395.00 now	**$335.69**
BPI General Ledger	$395.00 now	**$335.69**
The Mail Room	$29.95 now	**$25.39**
Continental General Ledger	$250.00 now	**$212.49**
Cont. Accounts Receivable	$250.00 now	**$212.49**
Cont. Accounts Payable	$250.00 now	**$212.49**
Continental Payroll	$250.00 now	**$212.49**
Broderbund Payroll	$395.00 now	**$335.69**
Informer II	$49.95 now	**$42.49**
Creative Financing	$150.00 now	**$127.49**
Real Estate Analyzer	$150.00 now	**$127.49**
Accounting Assistant (cass.)	$7.95 now	**$6.89**
Spell Star	$250.00 now	**$212.49**
Muse Form Letter	$100.00 now	**$84.99**
Property Management System	$225.00 now	**$191.19**

PERSONAL/HOME

Program	List	Now
Interlude (disk)	$19.95 now	**$16.99**
Handwriting Analysis	$19.95 now	**$16.89**
Decision Master	$29.95 now	**$25.39**
The Correspondent	$44.95 now	**$38.19**
Diet Planning	$24.95 now	**$21.19**
Win at the Races	$39.95 now	**$33.89**
Pro Football (SDL)	$26.95 now	**$22.89**
College Football (SDL)	$26.95 now	**$22.89**
Grocery List	$19.95 now	**$16.89**
Financial Management System II	$39.95 now	**$33.89**
Creativity Tool Box	$44.95 now	**$38.19**
Home Money Minder	$34.95 now	**$29.69**

MISCELLANEOUS

Memorex Disks		**10 for $24.99**
Verbatim Datalife-plain w/hubs		**10 for $27.99**
Dysans		**10 for $49.99**
Flipsort Box		**$29.80**
Scotch Disk Cleaner	$29.95 now	**$26.99**
E-Z Port	$24.94 now	**$21.19**
Atari/TRS-80/Pet		**Write for information**

HARDWARE

D.C. Hayes Micromodem	$375.00 now	**$299.00**
D.C. Hayes Smart Modem	$279.00 now	**$249.00**
MX-80 Printer/Cable/Card	$755.00 now	**$575.00**
MX-80 Replacement Head	$39.00 now	**$34.99**
MX-100FT with Graftrax	$995.00 now	**$799.00**
MX-80FT with Graftrax	$795.00 now	**$669.00**
460G Paper Tiger	$1094.00 now	**$961.99**
560G Paper Tiger	$1394.00 now	**1225.00**
Tiger Trax	$16.95 now	**$15.25**
Mark IV DOS Boot Switch	$24.95 now	**$21.19**
Z-80 Softcard	$395.00 now	**$299.00**
Expansion Chassis	$795.00 now	**$675.00**
Mt. Comp. Music System	$545.00 now	**$479.00**
Versawriter	$249.00 now	**$219.00**
NEC 12" Green on Black	$260.00 now	**$239.00**
Videx 80-col	$350.00 now	**$299.00**
Videx Switchplate	$19.95 now	**$17.99**
Andromeda 16K Ram Expansion	$199 now	**$179.00**
Microsoft 16K Ram Expansion	$195.00 now	**$169.00**
Centronics 739		Call for Price
Lazer (Rev. 6 or 7)	$69.95 now	**$58.99**
Keyboard Plus	$119.00 now	**$99.99**
TG Joystick	$59.95 now	**$50.89**
TG Game Paddles	$39.95 now	**$33.89**
Mimco Joystick	$59.95 now	**$54.99**
Paymar LCA Rev. 7		**$29.69**

Call Toll-Free 800-344-4111 (Outside California)

HUNTINGTON COMPUTING
Post Office Box 787
Corcoran, California 93212

Order by Phone 800-344-4111
In California (209) 992-5411

SUPER DISCOUNTS
apple® SOFTWARE

VISA
master charge

We take MasterCard or VISA (Include card # and expiration date). California residents add 6% tax. Include $2.00 for postage. Foreign and hardware extra. Send for free catalog. Prices subject to change.

Undeletable Lines

Michael P. Antonovich
Wyomissing, PA

Editor's Note: Though described for Applesoft, this crafty technique works on CBM computers as well. For PETs, type SYS1024. Then .M 0400 041F and hit RETURN (where the author mentions CALL-151). — RTM

Did you ever wish that you could put your name into a program in such a way that the average computer user could not delete it and claim the program as his own? Well you can. In APPLESOFT, you normally cannot enter program lines greater than 63999, but I will show you how you can. First we have to see how the APPLE stores your program lines in memory.

The APPLE stores APPLESOFT program lines beginning at memory location $800. (The '$' sign before a number indicates that the number is in hexadecimal). Let's enter the following small program to illustrate the way a program is stored.

```
1 REM
2 A=8
3 PRINT A
4 END
```

To see how the APPLE stores this program, we have to enter the monitor with a CALL -151. However, before we list the program, there is one other piece of information that we need to determine. To add lines to an existing program, you need to know where the current program ends in memory. You can accomplish this in one of two ways. You can page through the memory to find the program's last byte. (Hint: This is the hard way.) The APPLE also stores the location of the last memory byte in locations $69 and $6A. Let's enter the monitor now to check our program.

```
CALL -151
*69.6A
0069-1E 08
*800.81F
0800-00 07 08 01 00 B2 00 0F
0808-08 02 00 41 D0 38 00 16
0810-08 03 00 BA 41 00 1C 08
0818-04 00 80 00 00 00 FF FF
```

Although you may not recognize it, the above listing is a memory dump of your program. Let's examine how our BASIC lines were translated to the above hex dump.

The first byte $00 at location $800 has no special meaning to our program. In fact, location $800 will always contain $00. The program lines begin after this point. Each line is prefixed by four bytes. The first pair of bytes stores the starting byte address of the next line. In our example, locations $801 and $802 indicate that the next line will begin at memory location $807. (Remember that the location is split into two bytes which are stored in what seems, to us humans, to be in reversed order.)

The second pair of bytes contains the line number assigned to the program line. In our example we started with the line number "1." Thus memory locations $803 and $804 indicate that the first line number is "1." In addition to the four bytes which prefix each line, each line is ended with single byte '00' to separate it from the next line. Therefore, there is a five byte overhead for each program line used. If multiple statements are combined with a colon (using one byte) on a single line, you can save four bytes for each extra line you eliminate. If you have any doubts, try it yourself with the above program.

... though described for Applesoft, this crafty technique works on CBM computers as well.

The second program line begins at memory address $807. The first four bytes indicate that the next statement will begin at location $80F and will have statement number "2." The next three bytes "41 D0 38" represent the tokens for the equality: A=8. The information we will need to understand these tokens is found in Appendix F and Appendix K of the APPLESOFT Reference Manual. I'll wait while you go get your copy.

Appendix F lists the decimal tokens for all of the keywords used by the APPLE. However, when we are in the monitor, we need the hexadecimal equivalent of the tokens. For example, the hex equivalent for END is $80, REM is $B2, and PRINT is $BA, etc. You might want to take the time now to write the hexadecimal equivalents next to the decimal values for all of the tokens.

Variable names, numbers and strings are not listed in Appendix F. These must be constructed by using the individual ASCII character representations. Appendix K lists the ASCII character set with the decimal and hexadecimal codes. Again, we are interested in the hexadecimal codes. In our example, we need the "A" or $41 and the "8" or $38.

CRYSTALWARE

APPLE 48K-w/disk

THE FINEST IN FANTASY GAME SOFTWARE

ATARI 40K-w/disk

At Crystal we are doing our best to provide the finest state-of-the-art graphic adventure software in the world. Our list of credits include the first indoor-outdoor graphic adventure, the first multi-disk graphic adventure, and now for the Atari, the first graphic adventure in the world which includes screen scrolling and animation. The era of the text adventure and games which are simple combinations of static graphics and text is rapidly drawing to a close. We attempt to utilize the full potential of your computer. True, many of our games use up to 48K and we only deal in disk products, but there are a lot of users out there who have worked hard to upgrade their systems to the max and we think they deserve games that will give their computer system a run for its money.

★★★ ADVENTURE GAMES ★★★

1-THE HOUSE OF USHER- Haunted house type adventure game with scrolling in the Atari version. Wander the creepy hallways of the three story castle based on Edgar Allen Poe's short story of the same name. Written in graphics, of course, with animation and sound. We have introduced a new mystery for another $100 prize. $29.95/1 disk

2-FANTASYLAND 2041 A.D.- The largest disk based adventure game in the world (that we know of). Enter the Hall of Heroes and prepare yourself for the greatest fantasy-role-playing game you will see for years to come. To win you must survive Congoland, Arabia, King Arthur, Captain Nemo, Olympus (a sea voyage), and Dante's Inferno (Hell itself). In both the Atari and Apple versions it takes up more than 400,000 bytes of memory and uses more than 400 hires screens. The winner of the contest described in the manual with this game will receive $1000.00 and a bronze trophy. We have pushed the award date forward to February 1982 to allow more people to participate in the contest. $59.95/6 disks

3-GLAMIS CASTLE- Yes, Pat and I are on our way to Britain to stay in the dreaded Glamis Castle. If we survive our real life adventure, we'll be measuring it and will be able to provide you with a 3-D game based on this ancient haunted site where King Duncan met his end at the hands of Macbeth. Our good friend, Mark Benioff, after much research, said there's a mystery room that has never been found in this castle and a half beast, half-man creature that guards a treasure therein. Our stay will be covered by the British media and we hope to share our experience with you through the writing of this game. $49.95/2 disks

4-BENEATH THE PYRAMIDS- You are an archaeologist in 1932 and must find your way through the perilous chambers beneath the pyramids to discover a golden statue of the cat goddess Bast. This game is in hires graphics, includes sound, your little man actually moves through the corridors which you can see on the screen. The monsters are animated and very agressive. There is a new $100 prize for the first to solve the mystery; which is a toughie! $29.95/1 disk

★★★ SPACE GAMES ★★★

5-GALACTIC QUEST- An excellent combination of Star Trek and Space Trader. Battle the animated Vegan fighters as you warp from galaxy to galaxy. At the same time, you may land on and trade with hundreds of planets. Super hires graphics and lots of sound. This has been one of our most popular games. $29.95/1 disk

6-SANDS OF MARS- Take an exciting voyage to the planet Mars via the Starship Herman. This game compared to the rest, is second only to Fantasyland 2041 A.D. It includes scrolling on the Atari and hundreds of full screen graphics. You can move your little man through the terrain of Mars; if, of course, you survive the exciting journey to Mars, which occupies the whole first disk. There is a new mystery and another $100 prize just waiting for some clever adventurer out there. Good luck! $39.95/2 disks

★★★ WAR GAMES ★★★

7-WORLD WAR III- You Atari gamers will have to see this in the Atari version to believe it! If your tired of war games which take 15 minutes a move and have a manual the size of a telephone book; but still want a complex, real-time action war game-this is it! It is designed for two arm-chair generals which may manuever up to 128 seperate type of units at a time. The game displays a map of Iran & Iraq in the first scenario and later on you will find yourself moving nuclear submarines and battleships through two world wars. This is not a boring copy of a board based game but an original war game which takes a lot of skill and may take weeks to play. $29.95/1 disk

8-WATERLOO II- If you had been Napolean would you have done a few things differently? Well as you approach this final battle you are equipped with the same forces, face the same opposition, and survey the same terrain which he did. We have done a great deal of research to make this historically accurate as well as extremely complex. Even the angle of sight, fatigue of the individual soldier, and his psychological profile are included in the calculations. Oh by the way, your opposition is no slouch. You may find it more difficult to change the course of history than you think! $49.95/2 disks

★★★ ARCADIA ★★★

9-LASAR WARS- Hires-3d space war simulation. Protect the earth from alien invaders. $29.95

10-LITTLE CRYSTAL- The first of our line of education software, which will be completed by December. It includes a very fine version of Hangman, Mr. Music; which transforms the computer into a piano, Gunk-a hilarious shoot-em up game, and Storytime- an anthology of bedtime stories featuring Herman, the cat, Oscar, the Hamster, and of course, Little Crystal. $39.95

11-IMPERIAL WALKER- A fine game pack written by our Atari programmer, Michael (graphics) Potter. Includes the Walker animation which is superb, Gunfight, and Lasar Nim, a game of 'how many robots'. $29.95

12-ADVENTURE PACK- (#1-4) $112 **13-SPACEOUT-** (#5-6) $58

14-THE WARRIOR- (#7-8) $64 **15-ARCADE-** (#9-11) $60

Special Note! For those of you who live near beautiful scenic Silicon Valley, we'll be opening up a store called Country Computer at 7453 Monterey St. in Gilroy. Hopefully, we'll be carrying a full line of Apple and Atari products and invite you to drop in for our Grand Opening II on September 15.

VISA master charge

Our order lines are open 24 hrs. a day 7 days a week.

(408) 778-2966 **CRYSTAL COMPUTER** **17429 Bluejay Dr., Morgan Hill, CA 95020**

I have now explained all of the hex codes used in our example except one, the equal sign (=). The problem is that both Appendix F and K contain an equal sign, but each has a different hex code. Which one is correct? We need a rule which will guide us with this and similar problems, so here it is. To construct a variable name, number, or string of characters, use Appendix K. Any symbol used in an arithmetic expression (such as =, (,), etc.) should be taken from Appendix F.

Finally, even though we end our program with an END statement, the APPLE does not know that this is the end of the program (for, indeed, it does not have to be). The end of the program is indicated by the byte pair '00 00' in the locations where it expects to find the next line number.

Now that we know how the APPLE interprets our APPLESOFT program and stores it in memory, we are ready to begin adding lines to our program which will be undeletable to the average APPLE user.

Normally, only program line numbers from 0 to 63999 can be used and referenced in an APPLESOFT program. Converting 63999 to hexadecimal we get $F9FF, but we *can* write larger hexadecimal numbers than in two bytes. In fact, we should be able to use the numbers from $FA00 through $FFFF or 64000 through 65536. Even though the APPLE will prevent us from entering these line numbers through the keyboard, we know enough about how the APPLE stores program lines to sneak them in.

Let's keep this example simple and assume that I just want to store my name and the date as remark statements. We could just as easily make them PRINT statements if we wanted to. The statements we want are:

```
64000 REM MICHAEL P. ANTONOVICH
64001 REM JUNE 28, 1981
```

Now enter the monitor (CALL -151) and type the following:

```
81C:37 08 00 FA
820:B2 4D 49 43 48 41 45 4C
828:20 50 2E 20 41 4E 54 4F
830:4E 4F 56 49 43 48 00 4A
838:08 01 FA B2 4A 55 4E 45
840:20 33 30 2C 20 31 39 38
848:31 00 00 00
```

Before you return to APPLESOFT, we must reset the end of program pointer. If we do not do this, the first time you run your program, any variables you store will write over the new lines you just added. Our program now ends at memory location $84C. This information must be put into locations $69 and $6A.

```
69:4C 08
```

We can now reenter APPLESOFT using CNTL-C RETURN, and list the program. There are lines 64000 and 64001 at the end. Try to delete them. HA! You cannot touch them. You can SAVE this program, reload it, RUN it, and copy it, and still those two lines will be there. In fact the only way to get rid of them is to enter the monitor, find where you want the program to end, change the last two bytes to "00 00," and change the program ending location in addresses $69 and $6A. It's easy, but only if you know how.

Remarks are not the only things that you can put into undeletable lines. You can store anything you want from program lines using tokens and character strings to machine language programs using this method.

I would like to clear one thing up before anyone gets the wrong idea. The tokens we have been using above are NOT machine language. All microcomputers use tokens to store keywords. All BASIC program lines are stored in the above manner, not in machine language. The program lines must be interpreted each and every time that they are run. However, there is a lot that you can do with undeletable lines.

[Manipulating BASIC using your monitor can also hide lines from LISTing, but they will RUN normally. To make this program print the 8, but without *line 3 appearing in the program LISTing — simply change the one hexadecimal number to 16 like this:*

```
.: 0800 00 07 08 01 00 B2 00 16
     Apple Version
.: 0400 00 07 04 01 00 8F 00 16
     PET Version
```

Also notice that PET and Apple versions are similar (both are Microsoft BASICs), but the hex number for REM is $B2 for Apple, $8F for PET. The location of the last BASIC memory byte (the top of a BASIC program) is different in the PET, too. For Original PETs, this address is found in $007C, $007D. For Upgrade and 4.0 PETs: $002A, 002B. Another difference is that Apple starts its BASIC programs at $0800, where PET starts at $0400. You will notice this reflected in the line links which contain the starting byte address of the next line. (In PET, the first links are at $0401 and $0402, and they point to $0407 where the next link points up the chain once again. It is this $0407 (or Apple's $0807) which we changed to skip over line three in our "hidden line" technique above.) Here is the complete program as it would appear in the PET:]

```
.: 0400 00 07 04 01 00 8F 00 0F
.: 0408 04 02 00 41 B2 38 00 16
.: 0410 04 03 00 99 41 00 1C 04
.: 0418 04 00 80 00 37 04 00 FA
.: 0420 B2 4D 49 43 48 41 45 4C
.: 0428 20 50 2E 20 41 4E 54 4F
.: 0430 4E 4F 56 49 43 48 00 4A
.: 0438 04 01 FA B2 4A 55 4E 45
.: 0440 20 33 30 2C 20 31 39 38
.: 0448 31 00 00 00 AA AA AA AA
```

©

Inverting A Matrix

Brian J. Flynn
Vienna, VA

Editor's Note: Although this is programmed for the TRS-80, the ideas can be quickly adapted to any computer. RM

Have you ever wanted to solve a set of simultaneous equations, or perform regression analysis, or learn a bit about how these routines work? Well, the essence of such procedures is inverting a matrix. And in doing this, an accurate and dependable routine is desirable. One such algorithm is names for the king of 19th century mathematics, Carl Friedrich Gauss, and for his French counterpart, Camille Jordan, pronounced "zhoȓ dañ." Gauss, you may recall, was the German genius who at age two detected and corrected an arithmetic slip that his father was making in tallying up a payroll. And he did this without the assistance of even a hand-held calculator!

A Level II BASIC program for Gauss-Jordan matrix inversion is presented here. The algorithm uses complete pivoting to minimize round-off error and to insure that every invertible matrix is in fact inverted, within limits of the TRS-80's computing capability.

A matrix is a rectangular array of numbers or variables, usually displayed in brackets. A square matrix is 2-dimensional with, as you guessed, as many rows as columns. And an identity matrix is square with 1's in the principal diagonal and 0's everywhere else. The principal diagonal is the line segment running from the upper left corner to the lower right corner of the matrix. Only a square matrix can be inverted, with the inverse equal to that array which when multiplied by the original matrix gives an identity matrix. In symbols, this is:

$$\mathbf{X \cdot X^{-1} = I.}$$

X is the original matrix. X^{-1} is its inverse. And I is the identity matrix. The dot between the two X's means multiplication. For example:

$$X = \begin{bmatrix} 1 & 4 & 8 \\ 2 & 1 & 3 \\ 6 & 9 & 4 \end{bmatrix}, \text{ and } X \cdot X^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To calculate X^{-1}:

Tack onto matrix X an identity matrix of equal size:

$$\left[\begin{array}{ccc:ccc} 1 & 4 & 8 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 1 & 0 \\ 6 & 9 & 4 & 0 & 0 & 1 \end{array}\right]$$
$$\quad\; X \qquad\qquad\quad I$$

The dotted line separates or partitions what is now a 3 row x 6 column matrix into two smaller matrices of equal size, X and I. The partition, in this case, is like the equator: it enables us to talk more easily about the halves of a whole.

Is the matrix nearly inverted?

"Not yet, not yet!" the Rabbit hastily interrupted. "There's a great deal to come before that!"[1]

Namely:

Transform the left portion of the 3x6 matrix into I, using row and column operations on the entire matrix to do this. What is initially the identity matrix will emerge as X^{-1}. In more detail, the following three steps are performed for as many times (iterations) as there are rows in X:

1. Search would-be key elements, illustrated below, for the one of highest absolute value.

2. Place this element in the pivot position, if it is not already there, by switching rows or columns. The pivot position for iteration #1 is the first spot in the principal diagonal, going from upper left to lower right. For iteration #2 it is the second spot, and so on.

Keep track of columns that switch place, and interchange corresponding rows in the next-to-final form inverted matrix.

3. Make the diagonal element in a column equal to 1, and make the other elements in the same column equal to 0.

For example:

Step 1:
Would-be key elements are circled.

$$\left[\begin{array}{ccc:ccc} (1) & (4) & (8) & 1 & 0 & 0 \\ (2) & 1 & 3 & 0 & 1 & 0 \\ (6) & 9 & 4 & 0 & 0 & 1 \end{array}\right]$$
$$\quad\; X \qquad\qquad\quad I$$

Module 2 in the computer program tacks I onto X. The solid circle is the position to pivot on in the first iteration. Would-be key elements, besides 1, are in the dashed circles. They are the members of X that can be moved into the pivot position by switching a row or column.

Module 3 identifies 8 as the would-be key element of highest absolute value.


JINSAM™ 8.0
space
age
micro
software
Used at NASA,
Kennedy Space Center
With Multiple Applications Related
to the Columbia Space Shuttle Project
USA

Step 2:

Old Column 3		Old Column 1			
(8)	4	1	1	0	0
3	1	2	0	1	0
4	9	6	0	0	1

To place 8 in the pivot position, columns 1 an 3 are switched (lines 3120 and 3130). Vector M "remembers" the column interchange.

Step 3:

$$\left[\begin{array}{ccc|ccc} 1 & 0.5 & 0.125 & 0.125 & 0 & 0 \\ 3 & 1 & 2 & 0 & 1 & 0 \\ 4 & 9 & 6 & 0 & 0 & 1 \end{array}\right]$$

Row 1 divided by 8.

The pivot element becomes 1 by dividing the first row in the entire matrix by 8 (lines 3140 to 3190). Next, the second and third elements in column 1 become 0's:

$$\left[\begin{array}{ccc|ccc} 1 & 0.5 & 0.125 & 0.125 & 0 & 0 \\ 0 & (-.5) & 1.625 & -0.375 & 1 & 0 \\ 0 & 7 & 5.5 & -0.5 & 0 & 1 \end{array}\right]$$

This is done by multiplying row 1 by -3 and adding the product to row 2, and by multiplying row 1 by -4 and adding the product to row 3. Lines 3200 to 3270 accomplish this.

This completes the first cycle of steps 1, 2, and 3. The circle is the new position to pivot on. The process continues as before. Skipping a few steps:

Step: Pentultimate

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0.11 & 0.13 & -0.06 \\ 0 & 1 & 0 & 0.09 & -0.39 & 0.12 \\ 0 & 0 & 1 & -0.20 & 0.50 & 0.04 \end{array}\right]$$

This matrix finally emerges. Recall, however, that columns 1 and 3 were interchanged in the second step of the first cycle. Hence, rows 1 and 3 must now be switched (module 4):

Step: Last

$$X^{-1} = \begin{bmatrix} -0.20 & 0.50 & 0.04 \\ 0.09 & -0.39 & 0.12 \\ 0.11 & 0.13 & -0.06 \end{bmatrix}$$

Module 5 is easily changed to display X^{-1} differently, e.g., to show more decimal places or to accommodate a large number of rows and columns, say 8 or more.

Finally, as an example of the routine's speed, inverting a 10 by 10 identity matrix takes about 45 to 50 seconds. This is not very fast. However, some of the algorithm's time is spent checking columns for the highest key element and for switching rows, when necessary, in the next-to-final-form inverted matrix. Some programs omit this check. But when they do, a danger arises that X will not be inverted when it really can be. And further, the elements of the inverted matrix may be of lessened precision. The preference here is for effectiveness rather than speed. But Gauss, always the perfectionist whose motto was "Few, but ripe," might challenge us: "Can't we have both?"

[1] Lewis Carroll, **Alice's Adventures in Wonderland.**

```
10 REM GAUSS-JORDAN MATRIX INVERSION; B. FLYNN; WINTER 1981
20 REM MODULE 1:ENTER DATA
30 GOSUB 1000
40 REM MODULE 2:TACK IDENTITY MATRIX ONTO DATA MATRIX
50 GOSUB 2000
60 REM MODULE 3:BEGIN INVERTING MATRIX
70 GOSUB 3000
80 REM MODULE 4:SWITCH ROWS IN THE NEXT-TO-FINAL-FORM MATRIX, IF APPROPRIATE
90 GOSUB 4000
100 REM MODULE 5:DISPLAY INVERSE MATRIX
110 GOSUB 5000
120 END
```

```
1000 REM MODULE 1
1010 DEFINT I-R:DEFDBL B,C,D,H,X
1020 CLS:PRINT"GAUSS-JORDAN MATRIX INVERSION WITH COMPLETE PIVOTING.
1030 INPUT"PLEASE ENTER THE NUMBER OF ROWS (COLUMNS) IN THE MATRIX";K
1040 DIM X(K,2*K),M(K)
1050 REM ENTER DATA
1060 FOR I=1TO K
```

Introducing:

Introducing LISP. The language that can think for itself.

With SmartWare, your microcomputer possesses intelligence. Solves problems like never before. Actually can learn from its mistakes. And educates itself in much the same way your brain operates. It's a new concept in the way information is handled in microcomputers.

We appropriately called our version of LISP, SmartWare. Because it's as limitless as the human imagination, mind, memory.

An advanced, high-level language, LISP was first developed for use in artificial intelligence on large computers. And now, LISP is available from Datasoft for use on the Apple II and Atari Computers.

Store multiple programs in memory. It can take it. Program other languages into LISP. It's no problem. Because LISP offers greater capacity and power. And, it's a faster, more streamlined language. For example, just 2 lines in LISP could equal hours of BASIC programming.

The fact: Relational data base capabilities. User and program definability. Pattern-directed invocation language. Uses syntax and data structures upon which esoteric applications may be implemented. Remembers data along with "relationships affecting it." Offers REAL power to micros.

At MIT, they say LISP is the language of the future.

At datasoft, we say, why wait. Ask your local computer dealer for SmartWare. Right now! And turn your computer on to thinking.

SmartWare by
Datasoft Inc.™
COMPUTER SOFTWARE

19519 Business Center Dr., Northridge, CA 91324
(213)701-5161

InterLISP/65 for ATARI 48K systems 149.95 & APP-L-1SP 1.7 for Apple II 48K systems 124.95.

```
1070 CLS:PRINT"PLEASE ENTER DATA.
1080 PRINT"ROW #";I;":"
1090 FOR J=1TO K
1100 PRINT"COL #";J:INPUT X(I,J)
1110 NEXT J,I
1120 RETURN
2000 REM MODULE 2
2010 REM ASSIGN 1,2,...,K TO VECTOR M
2020 FOR I=1TO K
2030 FOR J=I TO K
2040 X(I,K+J)=0:X(J,I+K)=0
2050 NEXT J
2060 X(I,K+I)=1:M(I)=I
2070 NEXT I
2080 RETURN
3000 REM MODULE 3
3010 FOR Q=1TO K
3020 IF Q=K THEN 3140
3030 REM SEARCH WOULD-BE KEY ELEMENTS FOR HIGHEST ABSOLUTE VALUE
3040 HE=ABS(X(Q,Q)):HROW=0:HCOLUMN=0
3050 FOR I=1TO K-Q
3060 DUMMY=ABS(X(Q+I,Q)):IF DUMMY>HR THEN HR=DUMMY:R=Q+I
3070 DUMMY=ABS(X(Q,Q+I)):IF DUMMY>HC THEN HC=DUMMY:C=Q+I
3080 NEXT I
3090 IF HE>=HR AND HE>=HC THEN 3140
3100 REM SWITCH ROWS, IF APPROPRIATE
3110 IF HR>=HC:FOR J=1TO 2*K:HOLD=X(R,J):X(R,J)=X(Q,J):X(Q,J)=HOLD:NEXT J
3120 REM SWITCH COLUMNS, IF APPROPRIATE
3130 IF HR<HC:FOR J=1TO K:HOLD=X(J,C):X(J,C)=X(J,Q):X(J,Q)=HOLD:NEXT
     J:H1=M(Q):M (Q)=M(C):M(C)=H1
3140 REM ADJUST KEY ROW
3150 B=X(Q,Q)
3160 IF B=0 PRINT"SINGULAR MATRIX":END
3170 FOR J=Q TO 2*K
3180 X(Q,J)=X(Q,J)/B
3190 NEXT J
3200 REM ADJUST REMAINING ROWS
3210 FOR L=1TO K
3220 IF L=K AND K=Q THEN 3280
3230 IF L=Q THEN L=L+1
3240 D=X(L,Q)
3250 FOR J=Q TO 2*K
3260 X(L,J)=X(L,J)-D*X(Q,J)
3270 NEXT J
3280 NEXT L,Q
3290 RETURN
4000 REM MODULE 4
4010 FOR I=1 TO K
4020 C=0
4030 FOR J=1 TO K
4040 IF M(J)=I THEN C=J
4050 NEXT J
4060 IF C<>I THEN FOR L=1TO K:HOLD=X(I,K+L):X(I,K+L)=X(C,K+L):X(C,K+L)=HOLD:NEXT
 L:H1=M(I):M(I)=M(C):M(C)=H1
4070 RETURN
5000 REM MODULE 5
5010 CLS
5020 FOR I=1TO K
5030 FOR J=1TO K
5040 PRINT USING"##.## ";X(I,K+J);
5050 NEXT J:PRINT
5060 NEXT I
5070 RETURN
```

©

Osborne AWAKENS the ATARI™ and Puts it to Work.

All too often, computer users who've been used to entertainment programs have trouble coming up with ideas for **practical** computing. Your computer plays space games well enough, but when you have a down to earth problem like the future of an investment or federal withholding taxes, can it perform? It can now.

Osborne/McGraw-Hill introduces **Some Common BASIC Programs for the Atari™**. This book contains 76 short, brilliantly documented programs that you can key directly into your **Atari™ 400 or 800** computer. You'll get a lot of math power including personal finance, taxes, and statistics. All the programs are listed below.

Future of an Investment
Future Value of Regular Deposits (Annuity)
Regular Deposits
Regular Withdrawals from an Investment
Initial Investment
Minimum Investment for Withdrawals
Nominal Interest Rates on Investments
Effective Interest Rate on Investments
Earned Interest Table
Depreciation Rate
Depreciation Amount
Salvage Value
Discount Commercial Paper
Principal on a Loan
Regular Payment on a Loan
Last Payment on a Loan
Remaining Balance on a Loan
Term of a Loan
Annual Interest Rate on a Loan
Mortgage Amortization Table
Greatest Common Denominator
Prime Factors of Integers
Area of a Polygon
Parts of a Triangle
Analysis of Two Vectors
Operations on Two Vectors
Angle Conversion: Radians to Degrees
Angle Conversion: Degrees to Radians
Coordinate Conversion
Coordinate Plot
Plot of Polar Equation
Plot of Functions
Linear Interpolation
Curvilinear Interpolation
Integration: Simpson's Rule
Integration: Trapezoidal Rule
Integration: Gaussian Quadrature
Derivative
Roots of Quadratic Equations
Real Roots of Polynomials: Newton
Roots of Polynominals: Half-Interval Search
Trig Polynomial
Simultaneous Equations
Linear Programming
Matrix Addition, Subtraction, Scalar Multiplication
Matrix Multiplication
Matrix Inversion
Permutations and Combinations
Mann-Whitney U Test
Mean, Variance, Standard Deviation
Geometric Mean and Deviation
Binomial Distribution
Poisson Distribution
Normal Distribution
Chi-square Distribution
Chi-square Test
Student's t-distribution
Student's t-distribution Test
F-distribution
Linear Correlation Coefficient
Linear Regression
Multiple Linear Regression
Nth Order Regression
Geometric Regression
Exponential Regression
System Reliability
Average Growth Rate, Future Projections
Federal Withholding Taxes
Tax Depreciation Schedule
Check Writer
Recipe Cost
Map Check
Day of the Week
Days Between Two Dates
Anglo to Metric
Alphabetize

Some Common BASIC Programs ed. by Lon Poole:

Atari™ edition - #53-5, $14.99 ☐
PET/CBM™ edition - #40-3, $14.99 ☐
TRS-80™ level II edition - #54-3, $14.99 ☐
Generalized BASIC edition - #06-3, $14.99 ☐

Practical BASIC Programs ed. by Lon Poole
Contains 40 additional programs with a wide range of home and business applications. (Generalized BASIC edition only) #38-1, $15.99 ☐

Make check payable to: **OSBORNE/McGraw-Hill**
630 Bancroft Way, Berkeley, CA 94710 **Dept. G8**

Phone: (415) 548-2805
TOLL FREE: 800-227-2895 Outside California

Name ______________________

Address ______________________

City/State/Zip ______________________

Plus ☐ .75/item 4th class ☐ $1.50/item UPS ☐ $2.50/item Air Mail ☐ $10.00/item Overseas
(California Residents add applicable tax.) ☐ Please send me your free catalog.

Total amount enclosed $__________ or charge my ☐ Visa ☐ Mastercard #__________ Exp. Date ___

Signature ______________________

THE apple® GAZETTE

Budgeting On The Apple

William R. Swinyard
Provo, UT

When I sat down at one of the University's dozens of Apples six weeks ago, someone had to show me where the on/off switch was. But I learned fast and, within a few weeks, I was doing pretty well at what I believed the Apple to be best at. I had maintained several "high score" records on Apple Invaders ... I was terrific at judging angle and windspeed in the Bombs game ... and I was even learning how to back away from brick walls in Maze. But I had not entered even one real line of programming.

One other thing I did discover, though, was how ... well, *nice* the Apple is. True, I had spent a little programming time on our DEC installation (and a *lot* of time on it using canned analytical software packages), but I always felt inhibited about getting too intimate with that machine because of its so formal, even intimidating, way of speaking — or should I say "shouting" — back. "FATAL PROCESSING ERROR. ILLEGAL CHARACTER IN DATA (K) [OCTAL 113] UNIT = 8/ACCESS = SEQUIN/MODE = ASCII," was typical of the helpful way the DEC had about it. What a contrast with the Apple! It almost seemed to *invite* me to stay with it just a bit longer, and even to try running a few little programs on it.

And so I did. This article is about one of the first. It simply permits the user to enter a monthly income figure then displays a budget, along with the discrepancy between the budget and the income. It has a few nice touches, I think. Program requests for budget changes are highlighted in a new budget display, there is a short subroutine to align the decimal points, and percentages of total budget are also displayed.

Briefly, here are what the sections of the program do:

Line 115 — Displys the title of the program on an otherwise blank screen for a few seconds.

Line 120 — Requests monthly income information.

Lines 240-570 — Lists out, with illustrative budget figures, the items appropriate for *my* budget. Yours may be different, and you will need to change the program listing where appropriate.

Lines 600-680 — Prints the output headings, the budget name and corresponding value, the total budget figure, and the deviation between income and budget.

Lines 690-775 — The above information is displayed for about 10 seconds, followed by requests for updated expenditures originating in these lines. If updates are made, a new display table is printed which highlights (with underlining) the updated expenditure figure.

Lines 780-830 — A subroutine for lining up the decimals.

Lines 835-900 — Prints out the fully revised expenditures table, but accompanied this time with percentage allocations of budget, if requested.

The tabular display has been arranged to just fit into the Apple screen, so a line printer is unnecessary to get full use of this program.

Several refinements to the program might make it yet more appealing for your use, but seemed like overkill for me. With little modification the program could be designed to display budget values and percentages in the first tabular display — only one extra FOR-NEXT loop would be required. The program could easily be rewritten to make it interactive in budget-area development, though this seems unnecessary for a family whose budget allocations typically fall into the same areas each month. Or the program could be easily modified to accommodate input of *gross* monthly income, and incorporate federal and state tax withholding information, FICA withholding, tax-deferred income, etc. by building in the appropriate withholding proportions (income level, number of exemptions, and claimed marital status would be required as input values). However, since Uncle Sam has been changing the FICA rates annually, I passed this improvement by in favor of the relative permanency of the program as it now stands. Still, if you are interested in simulating the effect of increasing or decreasing your number of exemptions, this modification could be worthwhile.

Figure 1 shows the initial output display after entering a net monthly income figure. Figure 2 shows the output following the program's request for any changes. You can see that the change was

NEW ROM BOARD FOR THE APPLE II*
$125.00 WITH UTILITY ROM.

With Andromeda's new ROM Board, you can plug many useful utility programs into your Apple II. Because ROM memory never forgets, you can access these utilities instantly without having to load them from disk.

The ROM Board comes with the utility ROM, which gives you five powerful options to apply to your Applesoft* programs. With the Utility ROM, you can do automatic line numbering, control a program list with a page mode, restore a crashed Applesoft* program in memory, alphabetize a disk catalogue and create a disk without DOS, giving you an extra 8K on your disk. Any of Soft Control Systems' other ROMS can be used, such as the Dual DOS in ROM, and Your'ple ROM.

You can install 2K PROMS, 4K PROMS, or even 2K RAM chips in each of the two memory sockets. So you can even have the Read - Write capability of RAM to develop PROM Programs yourself, or just have an extra 2K RAM for your machine - Language programs. Two 2732 PROMS allow a total of 8K of memory on the Board.

Now with One Year Warranty.

Don't forget the Andromeda 16K RAM Expansion Board $195.00

ANDROMEDA
INCORPORATED
P.O. Box 19144
Greensboro, NC. 27410
919 852-1482

Distributed By:

COMPUTER DATA SERVICES
P.O. Box 696
Amherst, NH. 03031
603 673-7375

*Apple II and Applesoft are trademarks.

made ("Med. and Dent.") and was highlighted with a short underline. After all changes have been made, the program gives you the option of requesting percentage breakdowns by budget area. This table is shown in Figure 3, which concludes the budgeting routine.

There is nothing very sophisticated about any of this, but the program has been a useful one for our family; there is nothing quite so engaging as seeing an inanimate "object" spring to life at your bidding. And the program was just a lot of fun to write. Let me know of your improvements of it.

Program 1.

```
10   REM  *************************
20   REM  *HOME BUDGETING PROGRAM*
30   REM  *CREATED JUNE 10, 1981 *
40   REM  *BY WILLIAM R. SWINYARD*
50   REM  *SHOWS AND CALCULATES  *
60   RÉM  *DISTRIBUTION OF INCOME*
70   REM  *INTO BUDGET CATEGORIES*
80   REM  *INCLUDING REQUESTS FOR*
90   REM  *UPDATED EXPENDITURES &*
100   REM *FOR PERCENTAGE DISPLAY*
110   REM *************************
115   HOME : PRINT : PRINT : PRINT
      : PRINT : PRINT : PRINT  TAB(
      10)"HOME BUDGETING PROGRAM":
       FOR I = 1 TO 1000: NEXT I: HOME
120   INPUT "WHAT IS NET MONTHLY I
      NCOME? ";SALARY
130   PRINT
232   DIM DD(17),DD$(17)
236   REM  **BUDGETED ITEMS IN LIS
      TING **
237   REM  **TO CHANGE BUDGET YOU
      MUST ACCESS THE LISTING DIRE
      CTLY**
240 DD$(1) = "HOUSING"
250 DD(1) = 500
260 DD$(2) = "HEAT"
270 DD(2) = 50
280 DD$(3) = "LIGHTS"
290 DD(3) = 35
300 DD$(4) = "DONATIONS"
310 DD(4) = 50
320 DD$(5) = "TELEPHONE"
330 DD(5) = 35
340 DD$(6) = "LOAN PAYMNTS"
350 DD(6) = 100
360 DD$(7) = "MED & DENT"
370 DD(7) = 35
380 DD$(8) = "GROCERIES"
390 DD(8) = 300
400 DD$(9) = "AUTO MAINT."
410 DD(9) = 25
420 DD$(10) = "GASOLINE"
430 DD(10) = 100
440 DD$(11) = "INSURANCE"
450 DD(11) = 50
.
470 DD$(12) = "CLOTHING"
480 DD(12) = 50
490 DD$(13) = "EDUCATION"
500 DD(13) = 20
510 DD$(14) = "HOME REPAIR"
520 DD(14) = 50
530 DD$(15) = "RECREATION"
540 DD(15) = 60
550 DD$(16) = "SAVINGS"
560 DD(16) = 50
570 DD$(17) = "OTHER"
580  REM  ***PRINT OUTPUT***
590  HOME
600  PRINT "MONTHLY INCOME",SALARY
605  PRINT "=============="
610 SUM = 0
620  FOR I = 1 TO 17
630  GOSUB 790: PRINT  TAB( A)I; TAB(
     4)DD$(I); TAB( B)DD(I)
635  IF I = TT THEN  PRINT  TAB(
     18)"---"
640 SUM = SUM + DD(I)
650  NEXT I
660  PRINT
670  PRINT "TOTAL BUDGETED =   "; SUM
680  PRINT "NET AFTER BUDGET = ";
     SALARY - SUM
685  IF Q = 1 THEN  GOTO 910
690  FOR X = 1 TO 10000: NEXT X
700  PRINT "*******************"
710  REM  ***CHANGES***
730  INPUT "WAS SPENDING OTHER TH
     AN ABOVE? ";B$
740  IF B$ = "N" THEN  GOTO 840
750  INPUT "TYPE ITEM NUMBER, ACT
     UAL EXPEND. ";I,X
760 TT = I
770 DD(I) = X
775  GOTO 600
780  REM SUBROUTINE FOR DECIMAL PLACEMENT
790 A = 2:B = 19:C = 25
800  IF I > 9 THEN A = A - 1
810  IF DD(I) < 10 THEN B = B + 1
820  IF DD(I) >  = 100 AND DD(I) <
     1000 THEN B = B - 1
825  IF DD(I) >  = 1000 THEN B =
     B - 2
827  IF PC > 9 THEN C = C - 1
830  RETURN
835  REM  ROUTINE TO COMPUTE & PRINT %'S
840  INPUT "WANT %'S OF BUDGET? ";C$
850  IF C$ = "N" THEN  GOTO 910
855  PRINT : PRINT
860  FOR I = 1 TO 17
```

```
865 Q = 0
870 PC = (DD(I) / SUM) * 100
880   GOSUB 790: PRINT  TAB( A)I, TAB(
    4)DD$(I); TAB( B)DD(I), TAB(
    C) INT (PC);"%"
890   NEXT I
895 Q = 1
900   GOTO 660
910   END
```

Figure 1.

```
         HOME BUDGETING PROGRAM
WHAT IS NET MONTHLY INCOME? 1600

MONTHLY INCOME  1600
==============
  1 HOUSING          500
  2 HEAT              50
  3 LIGHTS            35
  4 DONATIONS         50
  5 TELEPHONE         35
  6 LOAN PAYMNTS     100
  7 MED & DENT        35
  8 GROCERIES        300
  9 AUTO MAINT.       25
 10 GASOLINE         100
 11 INSURANCE         50
 12 CLOTHING          50
 13 EDUCATION         20
 14 HOME REPAIR       50
 15 RECREATION        60
 16 SAVINGS           50
 17 OTHER              0
TOTAL BUDGETED =      1510
NET AFTER BUDGET = 90
```

Figure 2.

```
WAS SPENDING OTHER THAN ABOVE? Y
TYPE ITEM NUMBER, ACTUAL EXPEND. 7,75
MONTHLY INCOME  1600
==============
  1 HOUSING          500
  2 HEAT              50
  3 LIGHTS            35
  4 DONATIONS         50
  5 TELEPHONE         35
  6 LOAN PAYMNTS     100
  7 MED & DENT        75
  8 GROCERIES        300
  9 AUTO MAINT.       25
 10 GASOLINE         100
 11 INSURANCE         50
 12 CLOTHING          50
 13 EDUCATION         20
 14 HOME REPAIR       50
 15 RECREATION        60
 16 SAVINGS           50
 17 OTHER              0

TOTAL BUDGETED =     1550
NET AFTER BUDGET = 50
```

Figure 3.

```
WAS SPENDING OTHER THAN ABOVE?
WANT %'S OF BUDGET? Y
 1 HOUSING         500   32%
 2 HEAT             50    3%
 3 LIGHTS           35    2%
 4 DONATIONS        50    3%
 5 TELEPHONE        35    2%
 6 LOAN PAYMNTS    100    6%
 7 MED & DENT       75    4%
 8 GROCERIES       300   19%
 9 AUTO MAINT.      25    1%
10 GASOLINE        100    6%
11 INSURANCE        50    3%
12 CLOTHING         50    3%
13 EDUCATION        20    1%
14 HOME REPAIR      50    3%
15 RECREATION       60    3%
16 SAVINGS          50    3%
17 OTHER             0    0%
TOTAL BUDGETED =     1550
NET AFTER BUDGET = 50
```

©

Named GOSUB's

M.R. Smith

In a previous article, (**COMPUTE! #**12), I showed how to use the Ampersand instruction in Applesoft to permit instructions of the form:

```
10 FIRST=1000
20 DEUX=2000
30 THIRD=3000
40 & GOSUB FIRST
50 & GOTO THIRD
```

In Applesoft BASIC, named GOSUB's and GOTO's are not normally allowed.

The machine language program given in that article did not allow the use of names in ON... GOSUB or ON...GOTO statements. The following machine language program rectifies that problem.

The following statements are now permitted:

```
60 FOURTH=4000
70 NUM=INT(1+RND(2))
80 & ON NUM GOSUB FIRST, DEUX
90 & ON NUM GOTO THIRD, FOURTH
```

Using statements of this form makes it much easier to follow programs containing a large number of subroutines. In addition, it is much easier to remember the name of a subroutine rather than its number.

I am presently working on an extension to these statements to allow the passing of variables to a subroutine. That means statements of the form:

```
& GOSUB CALCULATE(A, B, C)
```

allowing the power of FORTRAN subroutine calls.

Description Of The BASIC Program

Line 20. Load the machine language program.

Line 20–100. Demonstrates the & GOSUB and & GOTO statements.

Line 110–180. Demonstrates the & ON... GOSUB and & ON...GOTO statements.

Lines 1000–3500. Demonstration subroutines and statements.

Lines 5000–6160. This subroutine loads and checks the machine language program. Every 17th number is the simple sum of the previous 16 numbers. This allows the entry of the numbers to be checked. The machine language program can be saved using the instruction:

```
BSAVE NAMED.GO,A$300,L$88
```

and called into your programs using the instruction:

```
10 PRINT CHR$(4);"BRUN NAMED.GO"
```

before any call to the ampersand (&) statements are made.

Program 1.

```
 300.387

0300-  A9 4C 8D F5 03 A9 10 8D
0308-  F6 03 A9 03 8D F7 03 60
0310-  C9 B0 F0 0D C9 AB F0 29
0318-  C9 B4 F0 31 A2 10 4C 12
0320-  D4 20 B1 00 A9 03 20 D6
0328-  D3 A5 B9 48 A5 B8 48 A5
0330-  76 48 A5 75 48 A9 B0 48
0338-  20 B7 00 20 44 03 4C D2
0340-  D7 20 B1 00 20 7B DD 20
0348-  52 E7 4C 41 D9 20 B1 00
0350-  20 F8 E6 48 C9 B0 F0 0D
0358-  C9 AB F0 09 C9 AF D0 BC
0360-  68 20 B1 00 48 C6 A1 D0
0368-  04 68 4C 10 03 A5 A1 48
0370-  20 B1 00 20 7B DD 20 52
0378-  E7 20 B7 00 C9 2C D0 05
0380-  68 85 A1 D0 E0 68 68 60
*
```

Program 2.

```
1   REM  M. R. SMITH
2   REM  MAY 1981
3   REM
10   REM  LOAD THE ROUTINE - NORMAL
     GOSUB
20   GOSUB 5000
30   REM  ESTABLISH NAMES OF THE SUBROUTINES
40 FIRST = 1000:DEUX = 1500:THIRD = 2000
50 FOURTH = 2500:FVTH = 3000:SIXTH = 3500
60   REM  DEMONSTRATE NAMED GOSUB'S
70   &  GOSUB FIRST
80   &  GOSUB DEUX
90   REM   DEMONSTRATE   NAMED GOTO
100   &  GOTO FOURTH
110   REM   DEMONSTRATE   NAMED ON...GOSUB
120 NUM =  INT (1 + 3 *  RND (1))
130   &  ON NUM GOSUB FIRST,DEUX,THIRD
140   REM  DEMONSTRATE    NAMED ON...GOTO
150   FOR J = 1 TO 1000: NEXT : REM  DELAY
160 NUM =  INT (1 + 3 *  RND (1))
170   &  ON NUM GOTO FOURTH,FVTH,SIXTH
180   STOP
190   REM
970   REM  DUMMY SUBROUTINES AND LINES
980   REM
990   REM  FIRST SUBROUTINE
1000   PRINT : PRINT "IN SUBROUTINE FIRST": RETURN
1490   REM  SECOND SUBROUTINE
1500   PRINT : PRINT "IN A DIFFERENT NAMED SUBROUTINE"
1510   PRINT "IN SUBROUTINE DEUX": RETURN
1990   REM  THIRD SUBROUTINE
2000   PRINT : PRINT "IN SUBROUTINE THIRD": RETURN
```

```
2490  REM  LINE CALLED FOURTH
2500  PRINT : PRINT "LINE CALLED FOURTH": GOTO 120
2990  REM  LINE CALLED FVTH
3000  PRINT : PRINT "LINE CALLED FVTH": GOTO 150
3490  REM  LINE CALLED SIXTH
3500  PRINT : PRINT "LINE CALLED SIXTH": GOTO 150
4980  REM
4990  REM   LOAD IN MACHINE LANGUAGE PROGRAM
5000 LOW = 768:HIGH = 903
5010 OK = 1
5020  REM  LOAD IN GROUP OF SIXTEEN
5030  FOR J = LOW TO HIGH STEP 16
5040 CHECK = 0
5050  FOR K = J TO J + 15
5060  READ IT
5070 CHECK = CHECK + IT
5080  NEXT K
5090  REM   CHECK IF CHECKSUM OKAY
5100  READ SUM
5110 L$ = "OKAY": IF CHECK <  > SUM THEN L$ = "BAD":OK = 0
5120  PRINT L$
5130  NEXT J
5140  IF OK = 0 THEN  STOP
5150  REM  THINGS ARE OKAY - LOAD INTO MEMORY
5160  RESTORE : FOR J = LOW TO HIGH STEP 16
5170  FOR K = J TO J + 15: READ IT: POKE K,IT: NEXT K
5180  READ IT: NEXT J
5190  PRINT "BLOAD OKAY": PRINT : PRINT
5200  REM   SET THE AMPERSAND VECTOR
5210  REM  NOT NEEDED IF CALLED BY BRUN STATEMENT
5220  CALL 768: RETURN
6000  DATA 169,76,141,245,3,169,16,141,246
6010  DATA 3,169,3,141,247,3,96,1868
6020  DATA 201,176,240,13,201,171,240,41,201
6030  DATA 180,240,49,162,16,76,18,2225
6040  DATA 212,32,177,0,169,3,32,214,211
6050  DATA 165,185,72,165,184,72,165,2058
6060  DATA 118,72,165,117,72,169,176,72,32
6070  DATA 183,0,32,68,3,76,210,1565
6080  DATA 215,32,177,0,32,123,221,32,82
6090  DATA 231,76,65,217,32,177,0,1712
6100  DATA 32,248,230,72,201,176,240,13,201
6110  DATA 171,240,9,201,175,208,188,2605
6120  DATA 104,32,177,0,72,198,161,208,4
6130  DATA 104,76,16,3,165,161,72,1553
6140  DATA 32,177,0,32,123,221,32,82,231
6150  DATA 32,183,0,201,44,208,5,1603
6160  DATA 104,133,161,208,224,104,104,96,0,0,0,0,0,0,0,1134
```

©

Part II:

A Tape "EXEC" For Applesoft: Loading Machine Language Programs

Sherm Ostrowsky
Goleta, CA

Loading ML With BASIC

This has been an example of the simplest kind of EXEC file; it merely loads and runs a single ML program. Let's extend the procedure a bit more, now, and load both a ML program and an Applesoft program together. One of the most effective advanced programming tools has always been to combine higher level language (e.g., Applesoft) and lower level language (i.e., machine language) routines so that each does the jobs for which it is best suited. But it has long been a problem, much debated and written about in computing magazines, to get them both into the computer from external storage (i.e., cassette tape in this case).

Short ML routines can be included in DATA statements in your Applesoft program and POKEd into memory by a READ - POKE loop. The Lam technique, using strings rather than DATA statements, has also been used to accomplish this. But for ML subroutines of any significant length, this process takes an excruciatingly long time. Some articles have suggested "hiding" long ML routines in front of or behind Applesoft programs by changing certain "pointers" in memory before saving and after loading, to fool Applesoft into believing that they are all parts of one long BASIC program. For one reason or another, none of these techniques has proven very satisfactory.

With the advent of disk systems, this problem was at least solved for disk owners. They just write an EXEC program that loads an Applesoft program and ML subroutines individually stored on the disk. The EXEC program and DOS know where to find them (on the disk) and where to put them (in memory). It is all done automatically. Very simple — *if* you have a disk system. If not ... well, until now you were just sort of out of luck.

Now we cassette users can do the same kind of thing. It takes longer to load, named files are not available, and a little more work is required to arrange everything on the tape and set up the EXEC file (i.e., the loader) to read them and put them where they need to go, but this only has to be done once for any given set of cooperating programs. Thereafter, loading the whole group is just as simple as typing LOAD.

Let me give an example based on a previous article of mine "Clearing the Apple II Low-Resolution Graphics Screen," **COMPUTE!** #10. For those who may not have access to this issue, I'll give all the essential details as they are needed.

Consider the following short Applesoft program that simply calls a subroutine to fill the Low-Resolution screen with a random pattern of colors, asks you to select one of the sixteen low-res color numbers (0–15), then calls a ML subroutine which clears the screen to the selected color in an instantaneous flash. Here is the program:

... load both a ML program and an Applesoft program together.

```
  10 REM FLASH-CLEAR DEMO
  20 GOSUB 1000
  30 VTAB 23
  40 INPUT "SELECT COLOR (0-15):" ;C
  50 COLOR=C
  60 CALL 800
  70 FOR PAUSE = 0 TO 2000: NEXT: GOTO 20
  80 END
1000 GR
1010 FOR I=0 TO 39
1020 FOR J=0 TO 39
1030 COLOR=1 + INT (15 * RND (1))
1040 PLOT J,I
1050 NEXT J,I
1060 RETURN
```

Briefly, line 20 calls subroutine 1000 which fills the screen with random color squares. Line 30 positions the cursor in the text area below the graphics screen. Line 40 asks for your selection and line 50 sets it up to be used in the clearing operation which is performed when line 60 calls the ML subroutine which begins at location (decimal) 800. Line 80 pauses briefly and then loops around to try it all again.

In the original version, the flash-clear subroutine at 800 was stored as DATA within the main program and POKEd into memory using a READ - POKE loop; it's short enough so that this procedure is really quite adequate. But, for the purpose of this demonstration, we will have the Exec-Loader read the ML program in from tape and put it into memory where it is supposed to be. To see how to create the total tape package, you'll have to get the ML program into memory first somehow. The easiest way is to use the Monitor. Do this first, before you enter either the Applesoft loader or the Applesoft demo program. Type CALL - 151, and when you see the asterisk prompt ("*") type:

```
320:A5 30 A0 78 20 2D 03 A0 50 20 3D 03
60 88 99 00 4 99 80 4 99 00 5 99 80 5 D0
F1 60 88 99 00 6 99 80 6 99 00 7 99 80
7 D0 F1 60 (return)
```

Do it exactly as written, with a space between each pair of digits. Just keep right on typing, past the ends of lines, without stopping until you reach the end. The ML program is now in its place in memory. But where we want it to be is on tape.

The order in which things have to go on the tape is this: first the Applesoft loader, then the ML subroutine, and finally the Applesoft demo program. The demo program has to come last because, after the Exec-Loader program (which I will present below) has executed a LOAD for an Applesoft program (here, the demo), it will automatically be deleted out of memory. It is a standard part of an Applesoft LOAD to clear out any pre-existing Applesoft programs first and, although this can be circumvented, there is no reason to go to the trouble of doing so in this case. So the demo must be loaded last because the loader will cease to exist in memory at that time.

This loader is an extension of the one shown before, but with most of the "bells-and-whistles" left off to make it shorter. Aside from changing the memory locations in string Y$ to correspond to the present ML subroutine, the only other change made from the previous loader is the addition of a LOAD command for the demo program.

```
10 REM ML + AS LOADER
20 Y$ = "320.34CR D823G"
30 FOR I = 1 TO LEN (Y$): POKE 511 + I, ASC
   (MID$ (Y$,I,1)) + 128: NEXT
40 POKE 72,0: CALL - 144
50 POKE 214,85
60 LOAD
70 END
```

This version has no unnecessary features. If there's a loading error, you'll find out about it when the system prints "ERR" on the screen. Since it takes only a short time to load these little programs, there is no need to give you a message so you'll know what is going on during the loading process.

Now here's the procedure, written down in the form of an algorithm: a series of listed steps to be followed one by one.

1. Make sure that the ML subroutine is in memory locations 320 through 34C, as indeed it will be if you typed it in using the Monitor as described above.
2. Go into Applesoft (from the Monitor, type CONTROL-C (return)), and type in the Loader program, as given above.
3. Type: POKE 82,128
4. Start your cassette recorder in RECORD mode and type SAVE (ret). When the loader has been saved, stop the tape, but *do not rewind.*
5. Back to the Monitor (CALL -151). Type: 320.34CW restart recorder in

RECORD mode, *then* press (return). Stop recorder when finished, but again do not rewind.

6. Back to Applesoft. Type NEW. Type in the demo program. SAVE it in the regular way.

That's it. To check it out, rewind the tape, turn your Apple off and back on (to make sure memory is cleared out), and LOAD the tape in the usual Applesoft manner. You'll hear five beeps before it finishes loading, but, in the end, it stops just the same as any regular Applesoft tape. You run the program by typing RUN.

... we have just written the tape equivalent of an EXEC file which performs the equivalent of a BLOAD.

Does this seem like a lot of trouble to go through just to load a short machine language subroutine along with an Applesoft program? Of course it is. But suppose the ML program were a lot longer. I have a "Print Using" subroutine that is over 1000 bytes long. Have you any idea how many DATA statements it would need to POKE all that into memory from within my calling program, and how long it would take to do it? The fact is that, before the method I have been demonstrating came along, there was no good way at all for a tape user to put long ML subroutines into his Applesoft programs. Only disk users could have that convenience, with their "BLOAD" and "EXEC" commands. Well, we have just written the tape equivalent of an EXEC file which performs the equivalent of a BLOAD. And that's far from all. The possibilities seem endless. I don't want this article to seem the same, so I'll mention just one more idea and then leave you to enjoy cooking up others on your own.

Protecting Programs In Memory

COMPUTE! #11 had an article (page 76) on resolving the memory conflicts between Applesoft programs and the locations of the two hi-res graphics screen memories. The problem is that a really long Applesoft program, starting at its usual location of $0801 (which is decimal 2049) can easily grow to overflow into the first hi-res screen starting at $2000 (decimal 8192), and some can even intrude into the second hi-res screen starting at $4000. An equivalent problem can occur if you want to make use of the second page of lo-res graphics (did you know that there is a second page?) which unfortunately starts at $0800 and thus *always* conflicts with an Applesoft program.

Among the solutions mentioned in the article, the one which seems to me to be the most flexible is to move the whole Applesoft program, variables and all, to a starting location above the memory region you want to protect from interference. For example, if you just want to use page two of lo-res graphics, it would suffice to move the program up so that it begins at $0C01 (decimal 3073). If you want to use both hi-res graphics pages, on the other hand, you'd have to move the program all the way up to $6001 (decimal 24566), assuming you even have that much memory to play around with.

As a matter of fact, you don't actually "move" an Applesoft program around in memory. Although this can be done, it would require doing some repair work on pointers and relinking the program lines which is too technical for most casual users to bother with. It isn't necessary. All you have to do is arrange to have the program go directly into the desired memory locations at the time it is being read in from tape. That is a much simpler procedure, requiring only a couple of POKEs from the keyboard before typing LOAD. Even so, that puts us right back into the situation we were in with ML programs: you can't just type LOAD — you have to refer to a set of written directions in order to get the program to load right. At least you do if your memory is as poor as mine is at recalling such details without notes. And there's another point: if the program is to be usable by a non-computerist, how would you instruct such a person in the loading procedure?

So, here's another job for the Tape-EXEC, alias a loader program. Let the loader do the POKEs for you. That way, once you have gone to a little extra trouble to set the tape up right to begin with, you (or anybody) can forever after just type LOAD and let the computer handle the details. If you think it likely that you might want to use the program more than once or twice, then the extra preparations are worth the trouble in the long run.

A loader for a single Applesoft program, to be entered into memory starting at (say) $0C01, would look something like this:

```
10 REM HIGH MEMORY LOADER
20 POKE 104,12: POKE 3072,0
30 POKE 214,85
40 LOAD
50 END
```

The algorithm for preparing the loader-leader and tape is similar to the one given for the Applesoft/ML conjunction; just omit steps two and five.

If you want the program to load above hi-res page 1, just change line 20 to:

```
20 POKE 104,64: POKE 16384,0
```

If you want it to load above hi-res page 2, change line 20 to:

```
20 POKE 104,96: POKE 24576,0
```

©

Switching Cleanly From Text To Graphics

Brian Nakagawa
Fresno, CA

It is very distracting to watch the Apple II switch from TEXT to GRAPHICS mode or vice versa. Colored squares and grey lines appear briefly when switching from TEXT to LORES graphics while a screen full of inverse characters, usually @ signs, appear when switching from LORES graphics to TEXT. In going from TEXT to HIRES graphics, one sometimes sees the previous HIRES display flash on the screen then dissolve away. In most programs, the extra flash of graphics or characters is merely a distraction. However, to switch cleanly between TEXT and GRAPHICS mode would give the appearance of a more polished program.

This article will show the routines that switch cleanly from TEXT to LORES or HIRES graphics using Applesoft commands and machine level routines already available on the Apple II.

The commands given below can be typed in the immediate mode or run in a program.

TEXT To HIRES Graphics

The simplest way to turn on page 1 of HIRES graphics is to use the command HGR. This first turns the screen to that page then erases the graphics displayed. Type in the following sequence of commands to see this.

```
VTAB 24
HGR
HCOLOR=3
HPLOT 40,140 TO 240,60 TO 40,60 TO 240,140 TO
   140,20, TO 40,140
TEXT
HGR
```

Notice that when you typed in the last HGR the star reappeared briefly then was erased.

Now try these commands. (It is assumed that you are still in HIRES graphics.)

```
HPLOT 40,140 TO 240,60 TO 40,60 TO 240,140 TO
   140,20 TO 40,140
TEXT
POKE 230,32: CALL 62450: HGR
```

Notice that the screen went blank without the brief appearance by the star.

POKE 230,32 sets the HIRES page pointer to page one without changing the current screen display. CALL 62450 is a machine language routine in APPLESOFT that clears the HIRES page indicated by memory location 230. HGR then turns on page one of HIRES graphics.

HIRES page two can be switched to using similar commands. Use the above example and substitute "HGR2" for "HGR" and "POKE 230,64" for "POKE 230,32". Be aware that the text window at the bottom will not be seen.

Switching from HIRES graphics, page one or two, TEXT is easily accomplished typing in the "TEXT" command. If you wish to clear the text page first then use "HOME: TEXT".

TEXT To LORES Graphics

Fill the TEXT screen with text then type in the command "GR". The distracting display of grey lines and colored squares when switching from TEXT to LORES graphics can be avoided with the following commands. Fill the text screen with text before typing in the commands.

```
POKE 230,32: CALL 62450: HGR: CALL -1994: GR
```

If you wish to clear the text window at the bottom of the screen, type in "HOME" or add it to the end of the string of commands as follows:

```
POKE 230,32: CALL 62450: HGR: CALL -1994:
GR: HOME
```

Notice that it is first necessary to clear HIRES page one and go into that page just as was done in the TEXT to HIRES discussion. CALL -1994 changes the text page to inverse @ signs and GR put the screen into LORES page one.

If you are still in LORES graphics type in "TEXT: HOME" which will return you to a clear TEXT page. Did you notice the flash of inverse @ signs? This does not happen all the time so it may be necessary for you to go back into LORES page one with the "GR" command then type in "TEXT: HOME" again.

To avoid this, type in the following commands:

```
GR
HOME: HGR: POKE 34,0: HOME: TEXT
```

GR puts the screen into LORES page one, HOME then clears the text window at the bottom, HGR puts the screen into HIRES page one which is assumed to be clear, POKE 34,0 sets the top of the text window to the top of the screen, HOME clears the text page, and TEXT puts the screen into text mode.

Disadvantages

The code to switch from text to graphics takes more memory and executes slower. You may have noticed the pause when switching between TEXT and GRAPHICS. In a program that switches between TEXT and GRAPHICS often, one can put the switching code in a subroutine to save memory.

The additional code and slightly slower execution speed to switch cleanly is a very small price to pay for the more polished appearance of a program.

References

All of the above commands are documented in the *Applesoft Basic Programming Reference Manual.* Selected commands and the page number they appear on are listed below. POKE 230,32 and POKE 230,64 are documented as general use high resolution graphics locations on page 141. Decimal location 230 is equal to hexadecimal $E6. CALL 62450 and CALL -1994 are on page 134 and POKE 34,0 is on page 29. ©

Interfacing The CCS 7710A Asynchronous Serial Card

Sam Bassett
San Francisco, CA

The following is a list of the connections needed to set up the DT127 to work with the California Computer Systems' 7710A Asynchronous Serial Interface Card for the Apple II:

Pin on the Male DB-25P Connector			**DT127 25 pin Molex**		
Color Pin	**Name**		**Name**	**Pin**	**Recommended**
				1	No contact
3	RD	----------------→	RD	2	Red
6	DSR	----------------→	DSR	3	Green
5	CTS	----------------→	CTS	4	Brown
20	DTR	----------------	RTS	5	Blue
4	RTS	----------------	DTR	6	White
2	TD ←	----------------	TD	7	Orange
1,7	GND ←	----------------→	GND	8	Black
				9-25	No Contact

The CCS Asynchronous Board is defined as a *D*ata *C*ommunications *E*quipment (DCE) terminal, and the signals at its DB-25S (female) connector are defined in the same way that a modem's would be.

The CCS board transmits its *outgoing* signal on Pin 3 (RD). This must be connected to the DT-127's *incoming* signal connector on the CCS board — Pin 2 (TD).

So far so good — we have information signals being passed back and forth from the printer to the Apple. There is a possible problem, however — the printer can only print so fast, and the Apple can generate and send characters much faster than the printer can print them. Most printers run from 10 to 55 characters per second, which is equivalent to 100 to 550 baud. The Apple can transmit at well over 20,000 baud — 2,000 characters per second. The DT127 has a 625 character buffer built in (expandable to 16K), but if the Apple is sending characters faster than the NEC can print (55 cps), the buffer gets full, characters are lost, and weird things happen to the text that was to be printed.

All is not lost, however — the definition of RS-232 includes several hardware "*handshaking*" signals, and the CCS 7710A (unlike the Apple Inc. Serial and Intelligent Interface Boards) is set up to recognize and use these signals. When the Printer signals that it has enough characters in its buffer, the CCS board will stop sending characters until the printer sends an "OK" signal.

The DT-127 signals that it is OK to send characters to its input on Pin 2 (RD) by making Pin 6 (Data Terminal Ready — DTR) on the Molex connector high — +3 to +12 volts. The CCS board monitors Pin 4 (Request To Send — RTS) to see if the peripheral is ready to receive another character. If Pin 4 goes Low (-3 to -12 volts), the 7710A will *not* send another character until it goes High again.

The CCS 7710A board signals the peripheral that it can accept a character through its Pin 2 (TD) by making its Pin 5 (Clear To Send — CTS) High. The DT-127 watches its Pin 4 (CTS) to see if it is OK to send a message to the computer. If this signal goes Low, it will not send any characters until it goes High again.

Pins 1 & 7 on the RS-232 connector are connected to Ground, so Pin 8 of the Molex must be connected to one or both of them.

The last two signals are not absolutely necessary, but it is well to hook them up so that nothing is left hanging, or unconnected.

The DT-127's Pin 5 (Request To Send — RTS) is not implemented — it should be High at all times. It should be connected to the CCS board's Pin 20 (Data Terminal Ready — DTR), which tells the Apple: "Yeah, boss, there is somebody out there."

The CCS board's Pin 8 is also permanently at +12 volts, so it should be left unconnected, so that it does not accidentally short to ground and shut down the Apple's power. Pins 9 to 25 (except 20) are not implemented on the CCS board, so they should not be connected to anything.

The CCS board can transmit at any baud rate from 50 to 19,200, and matches the DT-127 perfectly — be sure to check Page 2-1 in the CCS manual for the correct Baud Rate Selector Switch settings, and Page 5 of the DT-127 manual, so that the baud rates being used by both your Apple and your Sellum are the same. ©

THE ATARI® GAZETTE

Cassette Boot-Tape Generation
From DOS 2.0S Binary-Load File

Raymond W. Polone
SYZYGY Microware of Texas

The binary-load file is a very easy-to-use method of accessing 6502 machine code instructions for ATARI users. The Model CXL4003 ASSEMBLER EDITOR User's Manual provides a method, using BASIC, to enter object code from a cassette tape. This article introduces a BASIC program technique that permits a user to generate a short-inter-record gap bootable cassette tape from a binary-load file. Such a tape then provides the option of using a language cartridge (i.e. BASIC or ASSEMBLER) or booting in the program with no cartridge — and no DOS!

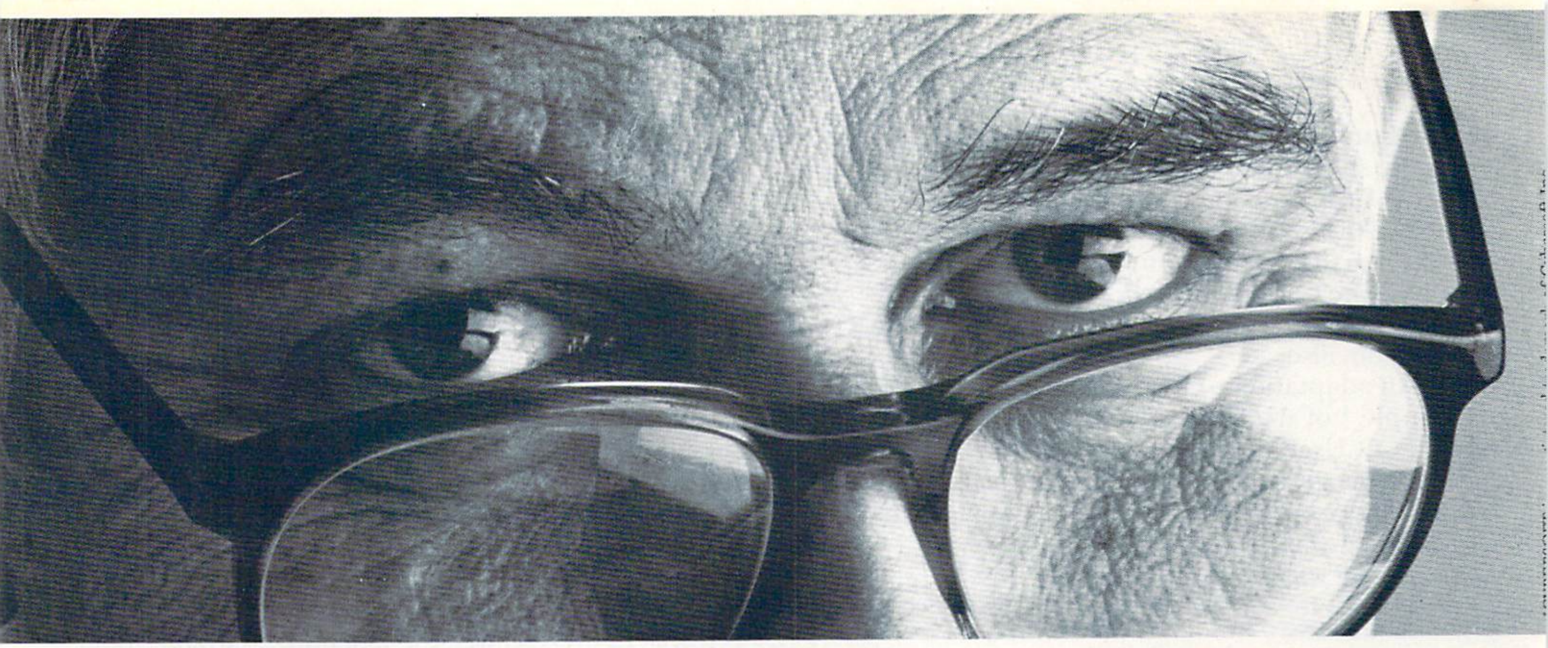
Program 1 is the ASSEMBLER/EDITOR text listing of an appropriate handler-record that must be written on the bootable tape. The initial byte of zero at statement 40 is a flag byte used by the OS in the cassette boot operation. I have assembled the second byte as a value one, but the BASIC program must provide the correct value. This byte is a count of 128 byte records in the boot process (i.e. a value of 1 indicates the boot is 128 bytes). A value of zero indicates the boot is 128*256 or 32,768 bytes. The BASIC program in Program 2 generates the correct SIZE value in statement 26. The object code of Program 1 is contained in DATA statement 56 of the BASIC program. (The BASIC program reads the DATA into BUF$ at statement 2.) The correct 128 count is then saved in BUF$ at statement 27.

The BASIC program must also provide the correct cassette boot load address in the value at statement 50 of Program 1. This BOOT value is calculated in statement 9 after the program has detected the two flag bytes of 255 ($FF) in the binary-load file. BUF$ is updated at statement 21 (the length of the boot-handler is subtracted from the binary-load file RAM location — this means that the boot-handler is appended in RAM at the beginning of the binary-load data).

The COLDSTART address or binary-load file-run address is provided to the value at statement 60 of the assembler routine by the BASIC program at statement 25 if the binary-load file is not load-and-go (no RUN-address appended to the binary-load file in locations $2E0 and $2E1). The user is prompted to enter the decimal value in statement 22. $A000 entered as decimal 40960 will result in the BASIC or ASSEMBLER cartridge gaining control (they must be resident, of course) following the cassette boot. The binary-load INIT address in $2E2 and $2E3 are not supported in this BASIC program, however, if the RUN-address has been appended to the file — that address will be used in statement 45 (RAY is a switch that indicates RUN-address has been found). The binary-load file must be a series of increasing RAM addresses! If there is a gap of addresses, the cassette boot will include the required block of zero RAM (ATARI hearts). (It may be desirable to DIMension BUF$ as the last variable in statement 1 to reduce the possibility that RAM used in the cassette put-character XIO operation will write extraneous data.) The only exception is the RUN-address which will not be included in the RAM block. (This cassette boot is not multistage.)

The remainder of Program 1 is the routine that will get control as a result of the cassette boot (POWER-ON with START depressed). BOOT? at RAM location 9 is set to 2 to indicate the cassette vector at RAM locations 2 and 3 is to be used on a SYSTEM-RESET. COLDSTART at RAM location $244 (decimal 580) is set to zero so that a SYSTEM-RESET will not result in a re-boot. The OS cassette-boot firmware stored the RUN-address at RAM vector locations $C (12) and $D (13). The reader may elect to modify the cassette SYSTEM-RESET vector to point to some WARM-START location or use the DOS vector at $A (10) and $B (11) by setting RAM location 9 to value 1. Additional features can be implemented by continuing the boot process to DOS by appropriate jumps to an OS location such as DOBOOT at $F2ED (62189).

DATA statement 57 in the BASIC program is the USR 6502 machine language routine that is used in statement 55. The ATARI IOCB use is adapted from the INTERFACE MODULE Opera-



TAKE A HARD LOOK AT OUR SOFTWARE.

Introducing CYBERSOFT™ custom ATARI® programs. The Grand Masters of programming at a mail order price.

Now that you own ATARI®, the finest, most versatile, most powerful home computer on the market, you should have the software to make your ATARI sit up and really function.

We have the "best of the best" programs from the Grand Masters of micro-computer software: Automated Simulations, Crystal, Swifty Software, Adventure International, Datasoft, Personal Software, and Dynacomp. All sold by mail. And, with the CYBERSOFT ironclad guarantee*.

Space Chase *(Swifty Software) 16K* Nifty eye-hand coordination tests... If you clear one galaxy of "planets" and avoid destruction at the hands of an alien ship you are rewarded by an attack by two aliens and then three. Game keeps score and displays high score. Colorful graphics and good playing. Cassette $14.95

Fastgammon™ *(Quality Software) 8K* The best of the computer backgammon games. You can repeat games with the same dice rolls to try different strategies. You roll dice or let computer roll for you. This machine language program challenges experienced and beginners alike. Underutilizes ATARI® graphics but playing quality more than makes up. Cassette $19.95

Invasion Orion™ *(Automated Simulations) 32K* You are the fleet commander pitted against the Klaatu somewhere in space. You play the leading role in a number of carefully crafted scenarios. This game takes more skill than luck in order to be successful. The documentation is superb. Create your own senarios. Excellent graphics, good sound. Cassette $24.95

Rescue at Rigel *(Automated Simulations) 32K* Ten people from the Orion colony at Ultima Thule (including your favorite female) have been captured. Your mother ship must leave in 60 minutes. As "Sudden" Smith, you must find and release them, but it won't be easy in the maze of corridors, rooms and levels. Oh, by the way, the place is crawling with bad guys. Cassette $29.95

Sands of Mars *(Crystal) 40K* This is the one gamesters are all talking about. A two diskette game, with diskette one putting you in command of the "Starship Herman". You must stock her, select her crew, command the launch and negotiate your way through the hazards of the Universe. Diskette two holds your adventures on the Sands of Mars. The game is saveable. Two Diskettes $39.95

Star Trek 3.5 *(Adventure International) 24K* Lots of color, lots of action as you hyperwarp through the quadrants in your search for Klingons. This program will satisfy the "trekkie" in us all. Cassette $14.95

Text Wizard™ *(Datasoft) 32K* ATARI owners have waited a long time for a high quality, full blown word processor. Text Wizard was well worth the wait. This easy-to-use and comparatively inexpensive program has received raves from user groups across the country. This program fully utilizes the unique ATARI keyboard and ease of cursor movement. Justifies right, left and center. Pagination. Word search and substitute. Diskette and ATARI 825, Centronics 737 or Epson MX-80 printer. $99.95

Adventureland *(Adventure International) 24K* Designed for the novice at adventuring and a great way to get hooked on the world of adventure. As you make your way through an enchanted world you will encounter wild animals, spell-ridden critters and more. Your task is to avoid the perils as you try to collect the 13 lost treasures. Cassette $14.95

Mission Impossible *(Adventure International) 24K* Another adventure from the great Scott Adams. Beginning "Good morning, your mission is...", your task is to save a nuclear reactor from destruction. No magic help in this one. Addiction is possible. Cassette $14.95

Bridge 2.0 *(Dynacomp) 24K* Now you can practice your bridge at home before you have to embarrass yourself at the neighbors. Bridge 2.0 bids (Goren) and plays both contract and duplicate bridge. It even doubles if you get carried away with your bidding. No graphics, but clearly arranged format makes playing easy. Cassette $17.95

Visicalc™ *(Personal Software) 32K* If you use your ATARI for any business purpose or for your personal finances, you will want Visicalc. It probably sold more computers than any program other than Star Raiders. Visicalc creates a very large accountants ledger pad upon which you write with your ATARI. The fundamental difference is, if you change any figure(s) the balance of the figures are automatically adjusted. Extensive documentation outlines the full features of this fantastic business tool. Diskette $199.95

Nominoes Jigsaw *(Dynacomp) 24K* A 60 piece jigsaw puzzle you complete on your screen. Three levels of difficulty to confound you. Scoring is a function of how many tries and how difficult the puzzle. Cassette $17.95

***CYBERSOFT Ironclad Guarantee.**

We fully guarantee the software we sell. If for any reason, you are not satisfied with your purchase, return the program within 30 days in its original packaging. We'll refund your money, no questions asked.

Our order system is as accurate and fast as our software. Just dial

1-800-247-2000 EXT. 505

(In Iowa 1-800-362-2133, extension 505). Or, you can write to CYBERSOFT at P.O. Box 505 Bellevue, Washington 98009. VISA and MasterCard gladly accepted.

Included $2.00 handling charge per order. Washington residents add 5.4% sales tax.

2803 122nd Place N.E. Bellevue, WA 98005
*Atari is a registered trademark of ATARI Inc.

cybersoft

tor's Manual C015953 and modified for cassette so that the short-inter-record gap tape can be written. BASIC USR function is explained in detail in BASIC REFERENCE MANUAL C015307 along with some information on XIO PUT CHARACTERS. The C016555 User's Manual is of tremendous assistance in understanding the ATARI operation, also.

An adaptation of the Exclusive-Or function in Example 1 of Appendix 9 in the ASSEMBLER EDITOR User's Manual is used to illustrate the BASIC program of Program 2 in this article. The following program will generate a binary load-and-go file that will perform the logic function.

```
10 OPEN#1,8,0, "D:LOADNGO.BIN"
20 FOR I=1 TO 30:READ X:PUT#1,X:NEXT I
30 CLOSE#1
40 END
50 DATA 255,255,0,6,17,6
60 DATA 104,104,133,205,104,133,204,104,69
70 DATA 205,133,213,104,69,204,133,212,96
80 DATA 224,2,225,2,0,160
```

The DATA statement 50 is binary-load information that indicates the program will load at RAM location $600 (1536). Statements 60 and 70 are the BASIC USR function machine language program. Statement 80 is the RUN-address information for the load-and-go $A000 RUN-address for BASIC or ASSEMBLER cartridge. It is only necessary to RUN this program to generate an input file for the BASIC program in Program 2.

If the resulting cassette-boot tape is used with the BASIC cartridge, the familiar READY will appear on the screen after a successful cassette-boot "START, POWER-ON, RETURN" sequence. A BASIC direct POLONE = USR(1536,65535,0) :?POLONE and RETURN will provide the response "65535". If the cassette-boot sequence is used with the ASSEMBLER cartridge, it will be possible to disassemble the machine language code that was booted-in around $600 using the DEBUGGER. This illustration is only a very elementary use of the cassette-boot tape that can be generated from a binary-load file.

Program 1. Cassette boot from binary load file assembler/editor text listing.

```
0000          20         .PAGE
0000          30         *=   $2100    ;BASIC PROVIDES CASSETTE BOOT ADDRESS
2100 0001     40 HERE    .WORD $100    ;ZERO FLAG - BASIC PROVIDES 128 BLOCK
2102 0021     50         .WORD HERE    ;CASSETTE BOOT LOAD ADDRESS
2104 0021     60         .WORD HERE    ;BASIC PROVIDES  COLDSTART ADDRESS
2106 A93C     70         LDA  #$3C     ;STOP CASSETTE
2108 8D02D3   80         STA  PACTL
210B A902     90         LDA  #$2      ;INDICATE CASSETTE BOOT
210D 8509     0100       STA  $9       ;ENABLE CASSETTE SYSTEM RESET @ BOOT?
210F A900     0110       LDA  #0
2111 8D4402   0120       STA  COLDST   ;DISABLE COLDSTART
2114 A50C     0130       LDA  $C       ;OS SAVED COLDSTART DOSINI
2116 8502     0140       STA  $2       ;PREPARE SYSTEM RESET
2118 A50D     0150       LDA  $D
211A 8503     0160       STA  $3
211C 6C0C00   0170       JMP  ($C)
D302          0180 PACTL =    $D302
0244          0190 COLDST =   $244
```

Program 2.

```
1 DIM RSTART$(7),YN$(3),NAME$(15):CORE=I
NT(0.89*FRE(0)):DIM BUF$(CORE),A$(114):R
AY=0
2 FOR I=1 TO 31:READ X:BUF$(I)=CHR$(X):N
EXT I
3 GRAPHICS 0:A$="DOS 2.0S BINARY-LOAD-DI
SK TO BOOT-TAPE by R. Polone (SYZYGY MIC
ROWARE OF TEXAS) REV1.0 1981":GOSUB 35
4 IOCB=16*2:FOR I=1 TO 7:READ X:RSTART$(
I)=CHR$(X):NEXT I
5 TRAP 5:A$="PLEASE ENTER BINARY LOAD FI
LE NAME     i.e. D1:LOADNGO.BIN (RETURN)"
:GOSUB 35:INPUT NAME$:TRAP 40000
6 TRAP 7:OPEN #2,4,0,NAME$:GOTO 8
7 ? :? "ERROR ";PEEK(195);" FILE ";CHR$(
34);NAME$;CHR$(34):CLOSE #2:END
8 TRAP 40000:GET #2,X:GET #2,Y:IF X<>255
 OR Y<>255 THEN A$="FILE NOT BINARY SAVE
 FORMAT":GOSUB 35:CLOSE #2:END
9 GET #2,X:GET #2,Y:FIRST=X+256*Y:BOOT=F
IRST-LEN(BUF$):GET #2,X:GET #2,Y:LAST=X+
256*Y:IF FIRST=736 AND LAST=737 THEN GOT
O 43
10 FOR I=FIRST TO LAST:GET #2,X:BUF$(LEN
(BUF$)+1)=CHR$(X):NEXT I
11 TRAP 18:GET #2,X:GET #2,Y:FIRSTA=X+25
6*Y:IF FIRSTA=65535 THEN GOTO 11
12 IF FIRSTA<=LAST AND FIRSTA<>736 THEN
GOTO 17
13 IF FIRSTA=736 THEN GET #2,X:GET #2,Y:
LASTA=X+256*Y:IF LASTA=737 THEN GOTO 44
14 IF FIRSTA=736 THEN GOTO 17
```


Star Warrior:
Slay the Dictator
and Save the
Civilized People
STAR WARRIOR
ARMED WITH NUCLEAR MISSILES, BLASTER & POWER GUN, YOU'RE ON A SEARCH & DESTROY MISSION AGAINST A MILITARY DICTATORSHIP!
EPYX
You owe yourself some fun time and you have just loaded the new Star Warrior into your computer. Suddenly, you are one of the Furies ...and retribution is your business.
The Interstellar Union of Civilized People has been annexed by a soul-crushing dictatorship. You have been given the mission to assassinate the military governor and then destroy the installation. That's Scenario 2.
In Scenario 1, you are on a diversionary mission for the assassination attempt. You want to do as much damage to the enemy units as possible.
The terrain is dangerous and difficult; the enemy forces are powerful and varied. (You may not see the enemy even when you are both in the same general area... but, on the other hand, the enemy may not see you either.) There is always the chance of an attack by an invisible enemy.
There are two scenarios and five levels of difficulty... level five is almost impossible to win. There are usually no fixed playing times for Scenario 2; you play until you get the governor and escape or until you are killed. But you can even set a fixed playing time in Scenario 1.
You have the choice of suits and equipment, and of movement, combat and special commands.
In other words, like all EPYX games, there are enough variations in the game so that you will never tire of playing. Each game is different and fresh. You will never get bored playing the game. Not in your or your computer's lifetime.
Star Warrior. Another bug-free, fantastic game from EPYX. With the unique EPYX lifetime warranty. If anything happens to your cassette or disk at any time and for any reason, send it back with just $5.00 for shipping and handling and we will send you a brand new one.
(Of course, there is also our 30-day unconditional guarantee: if your EPYX game has any defect whatsoever within 30 days of purchase, return it to us or your dealer and we will replace it free. No questions asked.)
Visit your dealer now and pick up Star Warrior in its good-looking, protective box with one of the best instruction books you ever read. Now available on disk for the Apple II (48K RAM with Applesoft) and Radio Shack's TRS-80 (32K RAM). And on cassette for the TRS-80 (Level II, 16K) and the Atari (32K). Only $39.95 on disk or cassette.
If your dealer is out of stock and you can't wait, order directly from Automated Simulations. $39.95 plus $2.00 for shipping and handling (and sales tax if you are in California).
Enclose your check. Or if you order by Visa or MasterCard, use our toll-free phones: In the United States: operator 861 (800) 824-7888; In California: operator 861 (800) 852-7777; In Hawaii and Alaska: operator 861 (800) 824-7919.
Order today. You and your computer deserve the fun.
AUTOMATED SIMULATIONS, INC.
Dept. SW4, P.O. Box 4247
1988 Leghorn Street
Mountain View, CA 94040
COMPUTER GAMES THINKERS PLAY
EPYX

```
15 FIRST=FIRSTA:X=FIRST-LAST-1:Y=LEN(BUF
$):IF X<>0 THEN BUF$(Y+X,Y+X)=CHR$(0)
16 GET #2,X:GET #2,Y:LAST=X+256*Y:X=LAST
-FIRST:IF X>=0 AND LEN(BUF$)+X<=CORE THE
N GOTO 10
17 TRAP 40000:A$="LOGICAL RECORD ERROR!"
:GOSUB 35:CLOSE #2:END
18 IF PEEK(195)=136 THEN TRAP 40000:GOTO
 21
19 IF PEEK(195)=5 THEN A$="RAM TOO SMALL
 IN THIS SYSTEM!  CANNOT GENERATE BOOT C
ASSETTE!":GOSUB 35:END
20 ? CHR$(155);" ERROR ";PEEK(195):END
21 CLOSE #2:X=INT(BOOT/256):Y=BOOT-256*X
:BUF$(3,3)=CHR$(Y):BUF$(4,4)=CHR$(X):IF
RAY=255 THEN GOTO 26
22 TRAP 22:A$="BINARY FILE IS NOT LOAD-A
ND-GO (NO RUN-ADDRESS APPENDED TO FILE)!
 PLEASE ENTER DECIMAL-ADDRESS":GOSUB 35
23 A$="$A000 HEX IS 40960 DECIMAL.":GOSU
B 35:INPUT AD
24 TRAP 40000:IF AD<0 OR AD>65534 THEN G
OTO 22
25 X=INT(AD/256):Y=AD-256*X:BUF$(5,5)=CH
R$(Y):BUF$(6,6)=CHR$(X)
26 SIZE=INT(LEN(BUF$)/128):IF SIZE*128<>
LEN(BUF$) THEN SIZE=SIZE+1
27 BUF$(2,2)=CHR$(SIZE)
28 A$="WRITE PREPARE BOOT TAPE!  BEEPS R
EQUIRE RETURN":GOSUB 35
29 AUX2=128:SIO=11:RW=8:LENTH=INT(LEN(BU
F$)/128)*128+128:BUF=ADR(BUF$):GOSUB 46:
CLOSE #2
30 TRAP 30:A$=" ANOTHER COPY OF BOOT TAP
E?  Y  N  RETURN":GOSUB 35:INPUT YN$:IF
YN$(1,1)<>"Y" AND YN$(1,1)<>"N" THEN GOT
O 30
31 TRAP 40000:IF YN$(1,1)="Y" THEN GOTO
28
32 TRAP 32:A$="ANOTHER BINARY DISK FILE?
  Y  N  RETURN":GOSUB 35:INPUT YN$:IF YN
$(1,1)<>"Y" AND YN$(1,1)<>"N" THEN GOTO
32
33 TRAP 40000:IF YN$(1,1)="Y" THEN RUN
34 END
35 X=PEEK(83)-PEEK(82)+1:I=X+1:Y=0:IF LE
N(A$)<1 THEN RETURN
36 IF LEN(A$)=X THEN ? A$:A$="":POKE 84,
PEEK(84)-1:RETURN
37 IF LEN(A$)<X THEN ? A$:A$="":RETURN
38 YN$=A$(I,I):IF YN$=" " OR YN$=CHR$(15
5) THEN I=I-1:? A$(1,I):A$=A$(I+2,LEN(A$
)):Y=255
39 IF Y=255 AND I=X THEN POKE 84,PEEK(84
)-1:GOTO 35
40 IF Y=255 THEN GOTO 35
41 I=I-1:IF I=0 THEN ? A$(1,X):A$=A$(X+1
,LEN(A$)):GOTO 35
42 GOTO 37
43 GOSUB 45:GOTO 9
44 GOSUB 45:GOTO 11
45 GET #2,X:BUF$(5,5)=CHR$(X):GET #2,X:B
UF$(6,6)=CHR$(X):RAY=255:RETURN
46 IF RW=8 THEN TRAP 47:LPRINT
47 TRAP 40000:OPEN #2,RW,AUX2,"C:"
48 POKE 832+IOCB+2,SIO
49 POKE 832+IOCB+4,BUF-(INT(BUF/256)*256
)
50 POKE 832+IOCB+5,INT(BUF/256)
51 POKE 832+IOCB+8,LENTH-(INT(LENTH/256)
*256)
52 POKE 832+IOCB+9,INT(LENTH/256)
53 POKE 832+IOCB+10,RW
54 POKE 832+IOCB+11,AUX2
55 RSTART=ADR(RSTART$):POLONE=USR(RSTART
,IOCB):RETURN
56 DATA 0,1,0,33,0,33,169,60,141,2,211,1
69,2,133,9,169,0,141,68,2,165,12,133,2,1
65,13,133,3,108,12,0
57 DATA 104,104,104,170,76,86,228
58 REM
59 REM -         FIGURE 2.  BASIC PROGRAM
 TO GENERATE CASSETTE BOOT
60 REM -                   FROM DOS 2.0S
 BINARY LOAD FILE
```

©

Beware The RAMTOP Dragon

K. W. Harms
Danville, CA

You've just had a brilliant idea for a program which requires some protected memory. Perhaps a special display list or character set is needed, or maybe a direct access memory "file." This article explains how to set aside that memory so that nothing will fiddle with it. Further, we'll reveal the generally unknown habits of the RAMTOP Dragon and show you three ways to make sure he doesn't gobble up your data.

The Atari offers a simple way to control how memory is internally managed by the operating system. The *Atari Connection* (Premier Issue) discussed "moving" the top memory boundary. Jim Clark in **COMPUTE!** #14 showed how to move the lower boundary. Both methods protect memory from BASIC programs.

The map gives a very simple picture of Atari's memory management. Fixed memory boundaries are presented in decimal "addresses," but boundaries which vary according to the amount of memory in your machine or the program loaded at a particular time are given names such as "ramtop." The 400 and 800 both use the same system.

When you turn on the Atari with a BASIC cartridge, it takes a few seconds before "READY" to check out the machine and enter values for the boundaries into specific locations. PEEK allows you to look at those values. For instance, the value for RAMTOP is stored in address 106. The instruction "? PEEK(106)" will tell you where the Atari thinks the end of RAM is. Appendix I of the *Basic Reference Manual* explains that the value in 106 is in "pages" of 256 bytes. Multiplying number of pages times 256 gives the last address BASIC thinks it can use (e.g., a PEEK(106) of "32" equals an address of 8192 or "8K").

The 400/800 always places the display list and display data immediately below RAMTOP. If you alter the value in RAMTOP, the Atari will push the display list and display data "downward" in RAM. This reduces the space available for your program, but leaves free RAM above the new (fake) RAMTOP. Since the Atari doesn't know about this space, it doesn't use it, usually. This is the space usually considered "reserved" for you.

Program 1 shows how to lower RAMTOP by 4 pages (1024 bytes). Remember to issue a GRAPHICS command immediately after moving RAMTOP so that the display list and data are moved below the new RAMTOP. Since line 60 will clear the screen, write down the old amount of free RAM and RAMTOP. Comparing them to the new numbers from lines 80 and 90 will show that RAMTOP is now lower and that less space is available for programs. That extra memory is now above RAMTOP and "reserved" for your exclusive use. RAMTOP is reset only by the RESET switch (and powering down/up), so that successively RUNning Program 1 will keep lowering RAMTOP until you run out of memory.

Program 1.

```
10 ? "FREE RAM= ";FRE(0)
20 RAMTOP=PEEK(106):? RAMTOP= ";RAMTOP
   ;" PAGES":? "LAST ADDRESS= ";RAMTOP*256
30 FOR W=1 TO 1000:NEXT W
40 SMALLRAM=RAMTOP-4
50 POKE 106,SMALLRAM
60 GRAPHICS 0
70 RAMTOP = PEEK(106)
80 ? "NEW FREE RAM= ";FRE(0)
90 ? "NEW RAMTOP= ";RAMTOP:" PAGES":
   ? "LAST ADDRESS= ";RAMTOP*256
100 ? "RESERVED MEMORY BEGINS AT ";
    RAMTOP*256+1
```

Although others have described ways to use the reserved space, they have not warned you about the RAMTOP Dragon who will periodically visit your reserved RAM and gobble up memory. Extensive field observations have revealed that Dragon visits upper memory on three occasions:

1. A GRAPHICS command clears the visible screen and also the first 64 bytes above RAMTOP.

2. A CLEAR command (or "PRINT CHR$(125)") clears the first 64 bytes above RAMTOP.

3. Scrolling the *text window* of a graphic mode 3 to 8 screen clears up to 800 bytes above RAMTOP.

Program 2 lets you play with the RAMTOP dragon. Lines 100 to 140 move down RAMTOP and reset the display list and data. Answer "NO" for all except the first pass or the program will lower RAMTOP each time until you are out of memory.

After an initial questioning, the next section (lines 200 to 290 first turns off the "direct memory access" for the ANTIC processor so that the program will operate faster (see **COMPUTE!** #11). It then fills the 900 bytes after RAMTOP with a sequence of values between 1 and 255. Note that the values will remain there as long as there's power to the CPU (and nothing clears them). Therefore, it's not necessary to repeat this step on subsequent RUNs.

The "Choose Action" section (lines 300 to 340) GOSUBs to the three major program sections.

The "Screen Play" routine (lines 1000 to 1100) exercises all three of the actions which call the Dragon. It clears the screen, changes graphic modes and scrolls text windows. To scroll a window, enter graphic mode 3 to 8 and then enter numerical responses for as long as you wish to scroll (the amount of scrolling appears to affect the amount of memory cleared).

The "Check Memory" routine (lines 2000 to 2100) prints addresses for the first and last positions of reserved memory and requests the starting and ending addresses you wish to check. This section allows you to look at different ranges of locations to see how much memory has been cleared by displaying these memory addresses and their values. Knowing that Dragon always leaves 0's in his path, and remembering that we loaded memory with values between 1 and 255, 0's will appear only in areas he visited. (Actually, I'm not sure whether Dragon is a he or a she.) When you're done checking and want to enter a different set of actions, an "0,0" entry will return you to the "Choose Action" section.

The "Neat Trick of the Week" is found in lines 2055 and 2075. The memory address at 53775 can be used to tell you whether a key on the keyboard is being pressed at the time you PEEK it. If a key (any key) is depressed, 53775 contains the value 251. When the key is released, 53775 will show a 255. Line 2055, then stops the program whenever any key is pressed and restarts it when the key is released. The POKEing 764 with a 255 (line 2075) clears the "halt" character so that future INPUTs, GETs, etc. aren't confused.

How can one avoid the Dragon anyway? There are many ways. You could never change graphic modes or clear or scroll the screen. This is difficult if you have any significant screen output. However, since the screen clear erases only 64 bytes, you could always clear the screen before the text window scrolls and never use that first 64 bytes. Or you could skip the first 800 bytes after RAMTOP and allow both scrolls and clears. Taking the other path, you could move the bottom of memory up and use memory below the new bottom (see Jim Clark's article). However, this requires using a (simple) machine language subroutine.

If you are using the reserved memory in a *stable* program (one with no further coding), you have another choice. Program 3 shows how to use memory below RAMTOP as your special area instead of reserving memory above RAMTOP. In your program, wait until after *all* strings and arrays are dimensioned. Then, go to the highest resolution graphic mode and PEEK at the top of your BASIC program (line 10000). Since the Atari saves data on GOSUB and FOR/NEXT statements as it encounters them in a dynamic "stack" at the top of a program, you must provide some room for this storage. Figure on 4 bytes for each *active* GOSUB (one which hasn't been RETURNed) plus 16 bytes for each *active* FOR/NEXT (while it's FORing and NEXTing). Add this allowance to the previous address (line 10010) and use the total as the *bottom* of your reserved area.

Next, PEEK at MEMTOP, the top of RAM available for BASIC programs (line 10100), and use that number as the *top* of your area.

Program 3.

```
10000 PROTOP=PEEK(14)+256*PEEK(15)
10010 MEMSTART=PROTOP+24+1: REM START
      OF YOUR MEMORY — ALLOWS FOR 2
      GOSUBS PLUS 1 FOR/NEXT
10100 MEMTOP=PEEK(741)+256*PEEK(742)
10110 MEMFINISH=MEMTOP: REM END OF
      YOUR MEMORY AREA
```

This method gives you the greatest possible amount of RAM without special code, but brings three general risks. If your BASIC program grows (by encountering an unexpected DIM or FOR/NEXT, for instance) after you have set the lower boundary, it will gnaw into the bottom of "your" memory. If the graphic mode is changed to a higher resolution mode after the upper boundary is set, the display list will push down into the reserved memory. Last, a program loaded after the boundaries are set may be larger and run into the set aside memory.

The next time you see the RAMTOP Dragon, you'll be ready!

Atari Memory Management

57344–65535	ROM for character set, OS, etc.	
55296–57343	ROM, Floating Point ROM	
53248–55295	ROM, Hardware Registers ROM	
	Unused space(!)	
40960–49151	CARTRIDGE ROM for BASIC, etc.	
RAMTOP	END of RAM	PEEK (106)
	Display List and Display Data (usually 1K in Graphics 0, around 8K in Graphics 8).	
MEMTOP	Top of RAM usable by BASIC programs.	PEEK(741)+256*PEEK(742)
PROTOP	Program top for the current BASIC program	PEEK(14)+256*PEEK(15)
	Free RAM used for programs, data storage, etc.	
BASIC LOMEM	Start of BASIC program	
	Operating System, various buffers, hardware registers, etc.	
0	Start of RAM addresses	

Program 2.

```
  50  REM SET UP VARIABLES FOR CALLS
  60  CHECK=1000:SCREEN=2000:QUIT=3000:DIM AN$(10)

 100  REM MOVE RAMTOP DOWN
 110  RAMTOP=PEEK(106)
 120  ? "MOVE RAMTOP DOWN";:INPUT AN$:IF AN$(1,1)="N" THEN 200
 130  RAMTOP=RAMTOP-5:POKE 106,RAMTOP
 140  GRAPHICS 0

 200  REM FILL 900 BYTES ABOVE RAMTOP
 210  FIRST=RAMTOP*256+1:LAST=RAMTOP*256+900
 220  ? "FILL MEMORY ABOVE RAMTOP";:INPUT AN$:IF AN$(1,1)="N" THEN 300
 230  POKE 559,0:REM TURN SCREEN REFRESHER OFF
 240  FOR POSITION=FIRST TO LAST
 250  IF VALUE=255 THEN VALUE=0
 260  VALUE=VALUE+1
 270  POKE POSITION,VALUE
 280  NEXT POSITION
 290  POKE 559,34:REM TURN SCREEN ON

 300  REM CHOOSE ACTION
 310  ? "WHAT ACTION? 1 TO CHECK RAM, 2 TO PLAY WITH SCREEN, 3 TO QUIT"
 320  INPUT ACTION
 330  ON ACTION GOSUB SCREEN,CHECK,QUIT
 340  GOTO 300

1000  REM SCREEN PLAY
1010  ? "CLEAR SCREEN":INPUT AN$
1020  IF AN$(1,1)="Y" THEN ? CHR$(125)
1030  ? "CHANGE GRAPHICS MODE";:INPUT AN$
1040  IF AN$(1,1)="Y" THEN ? "WHAT MODE";:INPUT MODE:GRAPHICS MODE
1050  IF MODE<>0 THEN ? "ENTER ANSWERS TILL DONE, THEN NO";:INPUT AN$
1060  IF AN$(1,1)<>"N" THEN GOTO 1050
1070  IF MODE <> 0 THEN GRAPHICS 0
1100  RETURN

2000  REM CHECK MEMORY
2010  ?:? "FIRST POSITION = ";FIRST:? "LAST = ";LAST:? "ENTER POSITIONS TO
      CHECK OR 0,0 TO    RETURN"
2020  INPUT START, FINISH:IF START=0 THEN GOTO 2100
2030  POKE 82,7:POKE 201,11:? :REM MOVE MARGIN, SET TAB
2040  FOR POSITION=START TO FINISH
2050  VALUE=PEEK(POSITION):? POSITION;"=";VALUE,
2055  HALT=PEEK(53775):IF HALT=251 THEN GOTO 2055
2060  NEXT POSITION
2070  POKE 82,2:REM RESTORE MARGIN
2075  POKE 764,255
2080  GOTO 2000
2100  RETURN

3000  REM QUIT
3010  ?"NORMAL END OF JOB":END
```

©

Documented Atari Bugs

Steve Hanson
Madison, WI

Although Atari has on the whole done an admirable job in getting out a bug-free computer system (at least compared to most of the other machines), there are a few bugs in the Atari computer software. This article is simply intended to be a list of bugs in operating system and BASIC software. It is intended mainly to be a guide to help you understand what is going on when unexplained things happen in your computer. Atari is aware of all of these bugs, and they will be corrected in the future.

Bugs In Atari BASIC

The Atari BASIC cartridge has a few problems. The currently known bugs include the following:

A. An input statement with no variable is not flagged as an error when input.

B. LPRINT loops cannot be stopped by hitting BREAK (This is not actually a bug in BASIC, but a bug in the OS cartridge.)

C. PRINT A = NOT B locks up the keyboard.

D. DIM L (10) generates DIM L10) as code. You must not leave a blank between a variable and its dimension or it will be interpreted incorrectly.

E. The following functions have wrong values:

LOG(0)
CLOG(0)
LOG(1)
CLOG(1)

MOST EXPONENTS (as an example, try to evaluate 2↑3. This problem is inherent in the polynomial expansion algorithm used. It is not likely to cause problems as the errors are very small, but will be noticeable when the expected value is something which is known. This is a simple rounding error, and can be handled by rounding the result when a non-integer result would be bothersome.)

E. Problems with BASIC boundary routine. This manifests itself in two bugs on the machine. Sometimes when doing line editing the machine will lock up. This usually occurs when deleting multiple lines from a program. The other problem is that any string which is an exact multiple of 256 bytes long will end up in the wrong place if moved in memory. This causes some very weird inexplicable errors in programs. It is hard to catch because it occurs when a string is a multiple of 256 bytes long, and only then. Please notice that it does not depend on the dimensioned length of a string, but on its actual length. Therefore, it cannot be avoided by never dimensioning strings to these values. Fortunately it rarely shows up since most strings are shorter than 256 characters.

F. A printed Control-R or Control-U is treated as a semicolon. I'm glad I finally found out about this as it was the cause of a great deal of trouble in a graphics dump routine I wrote.

G. You cannot use a function within a USR call line. That is, you cannot use a function to define an address for a user call or any of its parameters. You can do this if you evaluate them in a separate line, however.

H. You can dimension arrays larger than the available memory size without creating an error. Of course, there will be problems galore and error messages when you try to run the program.

These errors should mainly be corrected when the next revision of the BASIC cartridge is released. Errors caused in the operating system will be fixed with a new release of the operating system cartridge. Atari has no defiite plans as to when these will come out.

Errors In OS Cartridge

There are a few errors on the OS cartridge ROM in the Atari computer. These are the ones I know about:

As mentioned above, there is a bug in the OS which prevents the break key from interrupting LPRINT loops in BASIC.

There is another bug which you disk users have almost certainly noticed. A problem in the OS makes your disk drive "go to sleep" for a few seconds occasionally. Yes, there is a reason for this strange behavior. The same bug in the operating system is also the culprit responsible for the system occasionally retransmitting a block to the printer. Have you ever gotten a duplicate line or part of a duplicate line out to your printer (using the 850 interface.)? This is due to a bug in the OS cartridge and, at the moment, there isn't much you can do about it. However, Atari will have a fix for this eventually.

©

Graph It On The Atari

John Malcolm Neil
Portland, OR

One application for which the Atari is suitable is drawing accurate plots and graphs. With a minimum of programming effort, one can create graphs of any function — a handy aid for algebra and trigonometry homework.

The following listing is a 2K program to plot functions in graphics mode eight. It will run on either the Atari 400 or 800, providing that you have at least 16K of memory. A major feature of the program is that the user can look at any part of the Cartesian plane. One is not limited to the range of the graphics mode (320x160) when making graphs.

Line 500 sets the default limits for the range and domain of the function:

$-20<x<20,\ -10<y\leq 10$

This region is the user's *window* on the coordinate plane. To change these values, enter your own window when the computer prints "WINDOW?" in the form:

x_1,x_2,y_1,y_2

Where x_1 and x_2 are the minimum and maximum range, and y_1 and y_2 are the minimum and maximum domain.

To use the default values, just press RETURN.

The prompt "INTERVAL?" asks you to enter the interval at which to draw the tic marks. The default values are set in line 500 to one.

Enter the interval in the form:

x_1,y_1

Where x_1 is the interval on the x-axis, and y_1 on the y-axis.

Press RETURN to use the default values.

The relative positions of the x and y axes are drawn in lines 600-700; lines 800-890 draw the tic marks. The actual plotting of the function (lines 1000-1080) is rather unimpressive. The total range (Y2-Y1) is divided into 320 points, and the value at each is plotted. Should an error occur (divide by zero, cursor out of range), the program skips over that point.

Line 1030 contains the function to be graphed. To graph another, simply change it to read:

FUNCTION = (your function)

Any variables in it must be "X".

Now you are ready to graph! Try the functions in Appendix E for starters, and this one:

1030 FUNCTION = PEEK(53279)

Press some of the console keys (START, SELECT, OPTION). Wow!

A few words of warning — functions like f(x) = 0 will not be visible because the x-axis is drawn on zero. Also, if it is essential that your graph not be distorted, make sure that the range is twice the domain.

One idea I have for improvement is to couple this program with a machine language routine to dump the screen to a printer. I'm working on that one right now. In the meantime, happy graphing!

```
400 GRAPHICS 0
410 REM *** INITIAL CONSTANTS ***
500 X1=-20:X2=20:Y1=-10:Y2=10:XIN=1:YIN=1
510 LIST 1030
520 PRINT "WINDOW (X1,X2,Y1,Y2)";
530 TRAP 550
540 INPUT X1,X2,Y1,Y2
545 IF X2-X1=0 OR Y2-Y1=0 THEN 520
550 PRINT CHR$(125);"INTERVAL (X AXIS,Y AXIS)";
560 TRAP 590
570 INPUT XIN,YIN
580 IF XIN<=0 OR YIN<=0 THEN 550
590 GRAPHICS 8:SETCOLOR 2,0,0:SETCOLOR 1,0,
    10:COLOR 1:POKE 752,1
595 REM *** DRAW AXES ***
600 XTOP=Y2/(Y2-Y1)*159
610 IF Y2<0 THEN XTOP=0
620 IF Y1>0 THEN XTOP=159
630 YSIDE=ABS(X1)/(X2-X1)*319
640 IF X1>0 THEN YSIDE=0
650 IF X2<0 THEN YSIDE=319
700 PLOT 0,XTOP:DRAWTO 319,XTOP:PLOT YSIDE,0:
    DRAWTO YSIDE,159
710 REM *** DRAW TIC MARKS ***
800 TIC1=XTOP-1*(XTOP>0):TIC2=XTOP+1*(XTOP<159)
810 FX=INT(X2/XIN)*XIN
820 XV=(FX-X1)/(X2-X1)*319:IF XV<0 THEN 850
890 PLOT XV,TIC1:DRAWTO XV,TIC2
840 FX=FX-XIN:GOTO 820
850 TIC1=YSIDE-1*(YSIDE>0):TIC2=YSIDE+1*
    (YSIDE<319)
860 FY=INT(Y2/YIN)*YIN
870 YV=(Y2-FY)/(Y2-Y1)*159:IF YV>159 THEN 1000
880 PLOT TIC1,YV:DRAWTO TIC2,YV
890 FY=FY-YIN:GOTO 870
900 REM *** GRAPH FUNCTION ***
1000 PRINT CHR$(125):LIST 1030
1005 TRAP 2000
1010 C=0:FLAG=1
1020 FOR X=X1 TO X2 STEP (X2-X1)/319
1030 FUNCTION=SIN(X)
1040 IF FLAG THEN PLOT C,(Y2-FUNCTION)/(Y2-Y1)
     *159:FLAG=0:GOTO 1060
1050 DRAWTO C,(Y2-FUNCTION)/(Y2-Y1)*159
1060 C=C+1
1070 NEXT X
1080 POKE 752,0:GOTO 520
1090 REM *** ERROR HANDLER ***
2000 TRAP 2000:FLAG=1
2010 GOTO 1060
```

©

Extending Player Missile Graphics

Eric Stoltman
San Antonio, TX

One of the best features of the Atari is Player Missile Graphics. Since it has been introduced, many people have started using it in their programs, (some improving upon it, such as Larry Isaacs). This article and example program will explain how to create excellent animation, such as a walking figure or a rotating ship, with just one player.

One way to perform animation is to alternate players back and forth, but problems arise. What if the player is moved up? The other players also have to be moved up. This takes time. Another method is to alternately POKE data into the player, thus changing its shape. This can be done slowly in BASIC or quickly and easily in machine language. This program will compare both the BASIC and machine language method for changing the data of a player.

After a player is set up, additional data for other shapes must be stored in RAM. I prefer to use memory locations 256 to 511, since they are empty and are protected. This data can be manipulated by setting up pointers in an array. A subroutine can then easily retrieve this data and place it in the player's data area. This can be done in BASIC:

C = 0: A = PMBASE + 512 + Y TO PMBASE + 519 + Y: POKE A, PEEK(POINTER(FACING) + C) :C = C + 1: NEXT A

POINTER(FACING) = Array containing addresses of data.

EXAMPLE: POINTER(1) = 260, POINTER(2) = 268, etc.

Or in machine language:

A = USR(XXX,PMBASE + 512 + Y,POINTER(FACING))

XXX = Address of Machine Language subroutine.

The machine language method is not only easier, but also executes 11 times faster and provides smoother motion.

The machine language code is relocatable and can easily be modified by changing the 22nd data element so more or less data can be poked into the player's data area. For machine language programmers, I have included the machine language listing:

```
                *=      $0600
CHANGE          PLA
                PLA
                STA     $CC
                PLA
                STA     $CB
                PLA
                STA     $CF
                PLA
                STA     $CE
                LDY     #$00
LOOP            LDA     ($CE),Y
                STA     ($CB),Y
                INY
                CPY     #$08
                BNE     LOOP
                RTS
```

Line Numbers	Explanation
110-130	Poke machine language subroutine for changing player into player.
140-170	Poke data for additional shapes into memory.
180-190	Set up pointers to data.
200-250	Set up player
270-330	If trigger is pressed change player by machine language.
340-400	If trigger is not pressed change player by BASIC.

In addition to providing animation, this subroutine can move a player up or down when the vertical value changes greatly. To do this, point to an empty area of RAM (thus erasing the player) and then change the vertical value and point to the desired data. An example would be if a player went off the top of the screen and, using the method mentioned above, quickly reappeared at the bottom.

I should point out that many false players, that is, data for alternate shapes, may be stored and rotated among the four players to provide excellent animation.

```
10 REM BY ERIC STOLTMAN
20 REM THE 'I' IN POINTER SHOULD BE A '1
' SINCE 'POINT' IS A RESERVED WORD
100 REM ** INITIALIZATION **
110 FOR A=1536 TO 1560:READ I:POKE A,I:N
EXT A:REM POKE DATA FOR MACHINE LANGUAGE
 SUBROUTINE INTO MEMORY
120 REM ** MACHINE LANGUAGE DATA **
130 DATA 104,104,133,204,104,133,203,104
,133,207,104,133,206,160,0,177,206,145,2
03,200,192,8,208,247,96
140 REM ** ADDRESS OF PLAYER DATA **
150 FOR A=260 TO 323:READ I:POKE A,I:NEX
T A:REM POKE DATA INTO PROTECTED RAM
155 REM ** PLAYER DATA **
160 DATA 28,62,62,28,73,127,73,65,7,23,3
9,88,154,36,8,16,240,38,47,127,47,38,240
,0,16,8,36,154,88,39,23,7
170 DATA 65,73,127,73,28,62,62,28,8,16,3
6,89,26,228,232,224,15,100,244,254,244,1
00,15,0,224,232,228,26,89,36,16,8
180 REM ** POINTERS TO DATA **
190 DIM P01NTER(8):FOR A=1 TO 8:READ I:P
01NTER(A)=I:NEXT A:DATA 260,268,276,284,
292,300,308,316
200 REM ** SETUP PLAYER **
210 GRAPHICS 0:POKE 752,1:POKE 710,0:POK
E 559,46
220 A=PEEK(106)-8:POKE 54279,A:POKE 5327
7,3:PMBASE=256*A:POKE 53256,1:X=124:Y=48
```

```
230 FOR A=PMBASE+512 TO PMBASE+640:POKE
A,0:NEXT A
240 POKE 53248,124:POKE 704,12:FACING=1
250 C=0:FOR A=PMBASE+512+Y TO PMBASE+519
+Y:POKE A,PEEK(POINTER(FACING)+C):C=C+1:
NEXT A
270 REM ** MACHINE LANGUAGE CHANGE **
275 IF STRIG(0)=1 THEN 340
280 POSITION 5,5:? "MACHINE LANGUAGE";
285 J=STICK(0)
290 IF J=15 THEN 270
300 IF J=11 THEN FACING=FACING-1:IF FACI
NG<1 THEN FACING=8
310 IF J=7 THEN FACING=FACING+1:IF FACIN
G>8 THEN FACING=1
320 D=USR(1536,PMBASE+512+Y,POINTER(FACI
NG))
330 GOTO 270
340 REM ** BASIC CHANGE **
345 IF STRIG(0)=0 THEN 270
350 POSITION 5,5:? "BASIC            ";
355 J=STICK(0)
360 IF J=15 THEN 270
370 IF J=11 THEN FACING=FACING-1:IF FACI
NG<1 THEN FACING=8
380 IF J=7 THEN FACING=FACING+1:IF FACIN
G>8 THEN FACING=1
390 C=0:FOR A=PMBASE+512+Y TO PMBASE+519
+Y:POKE A,PEEK(POINTER(FACING)+C):C=C+1:
NEXT A
400 GOTO 340
```

Atari 400/800 Variable Name Utility

Arthur McGraw
Whitehall, OH

For compactness, Atari BASIC stores the ATASCII strings of variable names in a table called the variable name table. The program needs only a one byte reference to the variable name table to determine the desired variable name. This allows the programmer to use long descriptive variable names as the ATASCII string is stored only once, regardless of the number of times that variable name is used in the program.

When studying a program it is sometimes desirable to know what variable names are already in use. This utility will display the contents of the variable name table.

User Instructions

1. Clean out the computer's memory using the "NEW" command.
2. Type in the variable name table dump utility using line numbers that you do not normally use in writing BASIC programs. Modify the GOTO statements in lines 6 and 7 if you do not use my numbering scheme.
3. Save the utility on tape or disk using the "LIST" command.
4. Read in the desired BASIC program.
5. Read in the variable name table dump utility using the "ENTER" command. This will overlay your program with the utility program.
6. Start the utility by entering "GOTO x" where "x" is the first line number of the utility.
7. Observe the variable name table as it prints on the screen. The last character of each variable name will be printed in inverse video.

```
1 POKE 1664,PEEK(130):POKE 1665,PEEK(131)
2 IF PEEK (1664) = PEEK(132) THEN IF PEEK (1665)
  = PEEK(133) THEN ?: STOP
3 ? CHR$ (PEEK(PEEK(1664) + 256*PEEK(1665)));
4 IF PEEK (PEEK(1664) + 256*PEEK(1665)) >127
  THEN PRINT " ";
5 IF PEEK(1664) = 255 THEN POKE 1664,0: POKE
  1665,PEEK(1665) + 1: GOTO 2
6 POKE 1664,PEEK(1664) + 1: GOTO 2
```

Address List

130,131	Contains the address of the start of the variable name table.
132,133	Contains the address of the end of variable name table + 1.
1664,1665	Pointer used by variable name table dump routine. The pointer is stored in memory that is not used either by the operating system or by BASIC.

Line	Description
1	Set up pointer.
2	Check if done.
3	Print one character of variable name.
4	Print space if last character of variable name.
5-6	Increment two byte pointer.

Notes

1. Normally, the variable name table contains names no longer used by the program. Because these variable names occupy memory area, it would be desirable to remove them. Unused variable names can be removed from the variable name table by saving the program with the "LIST" command, clearing memory using the "NEW" command, and then reloading the program using the "ENTER" command.
2. If you do not have room to add this utility, you can delete as many lines of your program as needed by typing in their associated line numbers followed by RETURN. Deletion of program statements in this manner will have no effect on the variable name table. ©

Insight: Atari

Bill Wilkinson
Optimized Systems Software
Cupertino, CA

Last month we explored the possibilities inherent in the fact that Atari BASIC supports addressable DATA statements. This month we will tackle a related subject. It will be fairly obvious that the techniques presented in these two articles can help one write a fast yet complex adventure game on the Atari. However, it would be most interesting to me to see what other uses you readers make of these ideas, so start those cards and letters coming!

Nonexistent Subroutines ... Or Having Fun While GOSUBing Nowhere

Purists should probably not read this section: we will plumb depths that advocates of structured programming would never sink to. Just as Atari BASIC allows the code RESTORE ‹expression›, we can also use GOTO ‹expression› and GOSUB ‹expression›. (Don't worry about the notation: ‹anything› just means that "anything" is an English language word instead of a reserved BASIC word.) Allowing GOTO and GOSUB to refer to arbitrary expressions instead of absolute line numbers is unusual in BASICs (and well nigh impossible in most other languages), so perhaps the power of this capability has never been fully realized. We will try to make a few inroads.

Before we get into the more exotic part of our discussion, let us note the most obvious advantage of "line number expressions": self-documenting code. How much more meaningful it is to be able to code GOSUB CALCULATEPAY instead of GOSUB 13250 !! Admittedly, with Atari BASIC one must first have written LET CALCULATE-PAY = 13250 ; but that is a small price to pay for the added readability. Fair warning: there is one drawback to this trick. Atari BASIC allows only 128 different variable names. Normally that is a very big number, but naming every subroutine or section of code can eat up variables in a hurry. Be judicious in your choice of which routines are worth naming.

Now it is time for our main topic. And, thus, time for an example program. Study the example carefully before continuing with this text.

Lines 1000-1030 are simply set-up and initialization, one time operations. Lines 2000-2260 constitute the main loop of the program (in fact, in this simplistic example, this is an endless loop). First, the user is asked for a verb. If the INPUT verb matches one in the DATA list, it is assigned an appropriate verb number. The process is repeated for a noun. In an actual adventure game, the user would presumably be asked for "VERB NOUN" via a single INPUT; the program would then have to parse the request into the separate words.

At line 2200-2260 we exhibit the trick that this article is all about. To understand what happens, let us follow through what would happen if the user had requested "KISS BABY." "KISS" is verb number 2 and "BABY" is noun number 3, so at line 2220 we attempt to GOSUB to line 10000 + 2*1000 + 3*10; that is we attempt to GOSUB 12030. Lo and behold, line 12030 causes the message "YOU HAVE JUST BEEN ELECTED MAYOR" to print out. When the routine RETURNs, we GOTO LOOP and do this all over again.

But wait: suppose the user had typed "LOOK BABY." That phrase evaluates to verb 1 and noun 3, so we try to GOSUB 11030 (10000 + 1*1000 + 3*10). But there is no line 11030! All is not lost: note that on line 2210 we first TRAPped to line 2250. The attempt to GOSUB 11030 will trigger the TRAP and we indeed will continue execution at line 2250. Here, we attempt to GOSUB to line 10000 + 1*1000, effectively ignoring the noun. We succeed in executing line 11000, the "default" routine for the verb "LOOK," and find that the computer sees "NOTHING SPECIAL" about this baby.

The power implicit here is perhaps not obvious. But consider how easy it is to add new verbs and nouns to this program. Consider how easy it is to provide for as many or as few special verb-noun combinations as you wish. And, finally, look how little code is used!

Note that this program expects there will be a routine for each valid verb (it's only sensible: why have a verb in the DATA if it doesn't do anything)? Another TRAP statement, at line 2250, could allow for omitted verbs. By the way, with the program written as it is, there is no way to get to line 2250 with the error TRAP system still active. Atari BASIC always resets any TRAP when it is triggered (this is so that you can't accidentally fall into endless TRAP loops).

The techniques discussed in this and prior articles have actually been used to write a "PICO-ADVENTURE." The most amazing aspect of the program is the speed with which it responds: it seems as fast or faster than even machine language adventures. Try it. Let us know about your efforts.

Unreadable Programs

There follow two program lines that can be added to any existing Atari BASIC program and which, when executed, will make a program virtually unLISTable. The first line simply changes the names of all your variables to RETURN characters, and can be used with or without the second line. The second line actually produces a BASIC SAVEd program that can only be RUN — not LISTed, LOADed, etc.

Atari BASIC version:

```
32766 FOR I = PEEK(130) + 256*PEEK(131) TO
      PEEK(132) + 256*PEEK(133) : POKE I,155 :
      NEXT I
32767 POKE PEEK(138) + 256*PEEK(139) + 2,0 : SAVE
      "‹filename›" : NEW
```

BASIC A + version:

```
32766 for i = dpeek(130) to dpeek(132) : poke i,155 :
      next i
32767 poke dpeek(138) + 2,0 : save "‹filename›"
      : new
```

To use these gems, simply enter them and then type GOTO 32766. The line numbers are not important, but the second line must be the last line of the program. To use the resulting program, simply type RUN "‹filename›". The program should *not* end with STOP or END; instead, it should exit via NEW (yes, "NEW" can be used from within a program). The ‹filename› may be "C:", but CLOAD will not work (you must use RUN"C:").

VARIABLE, VARIBLE, VARABLE

Perhaps one of the more common mistakes when using the long variable names allowed by Atari BASIC is to make a typo when entering the name (I tend to leave off the plural, "s"). How to know you have committed this sin? Try the following program segment:

Using Atari BASIC:

```
32700 I = 0 : FOR J = PEEK(130) + 256*PEEK(131) TO
      PEEK(132) + 256*PEEK(133)-1
32710 IF PEEK(J) < 128 THEN PRINT CHR$(PEEK
      (J)) ; : GOTO 32730
32720 PRINT CHR$(PEEK(J)-128) : I = I + 1
32730 NEXT J : PRINT : PRINT I ; " VARIABLES
      IN USE" : STOP
```

Using BASIC A + :

```
32700 i = 0 : for J = dpeek(130) to dpeek(132 + 1 : print
      chr$(peek(j) & 127) ;
32710 if peek(j) > 128 : print : i = i + 1 : endif : next j
32720 print : print i ; " variables in use" : stop
```

It would be advisable to LIST this program segment to DISK or CASSETTE and then ENTER it into any program that needs it. To use, simply type GOTO 32700 ; all your variables will be listed and a count displayed. Obviously, this output could be sent to the printer by first OPEN #7,8,0, "P:" and then replacing all the PRINTs with PRINT #7. If you do this, it is advisable to CLOSE #7 before the STOP.

```
1000 REM ***** SET UP ********
1010 DIM VERB$(4),NOUN$(4),TEST$(4)
1020 VERBDATA=9000:NOUNDATA=9100
1030 LOOP=2000
2000 REM ***** MAIN LOOP ******
2010 PRINT "GIVE ME A VERB ";:INPUT VERB$
2020 RESTORE VERBDATA:VERB=0
2030 FOR CNT=1 TO 3:REM CHANGE TO MATCH DATA
2040 READ TEST$
2050 IF TEST$=VERB$ THEN VERB=CNT:CNT=99
2060 NEXT CNT
2070 IF  NOT VERB THEN PRINT "INVALID VERB":
     GOTO LOOP
2100 REM VERB DONE, DO NOUN
2110 PRINT "GIVE ME A NOUN ";:INPUT NOUN$
2120 RESTORE NOUNDATA:NOUN=0
2130 FOR CNT=1 TO 3:REM CHANGE TO MATCH DATA
2140 READ TEST$
2150 IF TEST$=NOUN$ THEN NOUN=CNT:CNT=99
2160 NEXT CNT
2170 IF  NOT NOUN THEN PRINT "INVALID NOUN":
     GOTO LOOP
2200 REM ***** THE TRICKY STUFF *****
2210 TRAP 2250
2220 GOSUB 10000+VERB*1000+NOUN*10
2230 GOTO LOOP
2240 REM WE GET TO 2250 ONLY ON TRAP
2250 GOSUB 10000+VERB*1000
2260 GOTO LOOP
9000 REM A LIST OF ALL VERBS
9010 DATA LOOK,KISS,DROP
9100 REM A LIST OF ALL NOUNS
9110 DATA ROOM,BEAR,BABY
10000 REM ***************************
10010 REM *    THE VERB-NOUN ACTION *
10020 REM ***************************
11000 REM >>> LOOK <<<
11001 PRINT "I SEE NOTHING SPECIAL":RETURN
11010 PRINT "I SEE A WINDOW AND A DOOR":RETURN
12000 REM >>> KISS <<<
12001 PRINT "THAT'S SILLY...BUT SMACK":RETURN
12020 PRINT "BEAR BITES OFF YOUR LIPS":RETURN
12030 PRINT "YOU HAVE JUST BEEN ELECTED
      MAYOR":RETURN
13000 REM >>> DROP <<<
13001 PRINT "HOW? I COULDN'T HAVE LIFTED
      THAT.":RETURN
13030 PRINT "IT'S A BOUNCING BABY BOY!!"
      :RETURN
```

©

Overview: "Letter Perfect" Wordprocessing On The Atari

This is the first **COMPUTE!** *Overview. We feel that our readers deserve the most objective, comprehensive analysis of new products possible. We have established review panels, each consisting of three reviewers whose background qualifies them to analyze new items of software or hardware.*

To better prepare you for your buying decisions, and to present fair reviews, we collect the three independent opinions of our panelists into one Overview. Lawyers will test legal packages in the environment of a legal office, doctors will test medical systems, and this issue's Overview panel is made up of professional writers. We believe that you will find **COMPUTE!** *Overviews complete, informative, and balanced. — KEM*

The "LETTER PERFECT" program by LJK Enterprises, Inc. is one of the first word processor programs for Atari. It requires 24K of memory and one disk drive and costs $149.

The program is very well documented, and easily followed, with 85 pages of instruction and indexing. This material is supplied in a handsome binder.

As the manual states, the control functions are relatively simple to remember (Control A for "go to beginning of a line," for example, or Control Z for "go to the end of a line," and control T for "go to the top of the page"). This doesn't mean you can remember all 42 functions after one sitting, of course, but the letters are not just utilized arbitrarily and it doesn't take long to catch on. (See Table 1).

In its present format, Letter Perfect is configured for the Atari 825 Printer and the Epson MX 80.

The Epson will not permit underlining, subscripts or superscripts, but the program does permit use of standard, condensed, and enhanced fonts. On the Atari 825, you may use standard, boldface or proportional fonts, although you will not get right hand justification when you use the proportional font.

Printer Considerations

It seems clear that anyone considering the purchase of a printer would do well to check on just what Word Processing systems are currently available that support the various printers, and what they do. In that context, Ken Leonhardi of LJK Enterprises, who puts out Letter Perfect, says that improvements are planned that include the capability of using Letter Perfect with just about any printer on the market.

It is very flexible in the format of the print and allows the user to set the format several times within the text. This could be used to single space for quotations in a double spaced letter or to indent a list of items within a manuscript.

Other printout features include auto page numbering, header and footer capability, auto line centering and all the standard features such as single/double line spacing and left margin adjust.

An Unusual Disk Capability

Letter Perfect also has its own disk operating system, which for some uses could be a disadvantage. Leonhardi says, however, a utility program will be released that will enable the program to be used with the Atari DOS.

Future disk versions of Letter Perfect will be available at a small additional cost for those who buy the current version.

One small warning: there are some early Atari disk drives on which Letter Perfect won't operate. This is apparently true for only a handful of the first Atari 810s that were released, but, if you've had one for some time, there could be a problem in running this program.

There are good and bad points to the unusual requirement that disks be specially formatted for Letter Perfect text storage. The manual describes "packing" — removing unnecessary spaces (such as the spaces between the period which ends one paragraph and the start of the next paragraph). This permits more text on a given disk. WordPro, for the Commodore machines, for example, saves *every* space to the disk. On the other hand, few of us are novelists, nor would we care very much about saving great amounts of text to individual disks. If you were writing a book, you might find that such packing saved a little time, but normal word processing uses do not benefit much from any special disk storage efficiency. The manual also suggests that such storage prevents problems should Atari change its DOS in the future. The special Letter Perfect formatting takes about 1 minute per disk. However, you usually need to format only rarely in word processing application.

Text Entry

First, some mechanics.

When you first load the program you are given a menu of commands, such as Editor, Load, Save, Printer, Delete, Lock, Unlock, etc. You go to the Editor command to begin writing, and can return to the menu by hitting the escape key.

All of the Letter Perfect processing functions are done by using the Atari control key. This works out quite well, since most Atari users already are practiced at using the up, down, right and left

arrows and the insert and delete functions, and they operate in Letter Perfect just as you would normally use them to write a program on your Atari. There are some things to watch out for, however.

One frequently uses the tab key to move to the middle of a line. If you use the tab key in Letter Perfect on a line that has text on it, the tab will erase as it goes along. This was seen as a serious weakness by two panel members.

There are 42 control key functions and they are listed on a handy card that Letter Perfect provides that fits neatly on top of your Atari. It also lists 13 commands for changing the default format (margin settings and the like) and the commands for subscripts and superscripts.

The functions include scrolling, saving paragraphs in memory, and inserting them afterward, replacing misspelled words automatically, tab setting, etc.

The program does not permit you to scroll upward. If you want to back up and read two paragraphs earlier in your text — you must return to the top of the text. Another time consuming feature is the fact that whenever you go from a main menu function (such as disk access) back to the text mode for writing, you must wait for the entire text to scroll down the screen in front of you. Such features quickly become tiresome when you are forced to wait for them.

The Atari screen permits only ½ page (a standard, typewritten 8½ x 11 sheet) to appear at one time. Though the panel felt that this was not fatal, it was seen as a constraint for many wordprocessing applications. In general, the more text available to you at a time, the easier it is to edit and review.

One of the primary differences between using a typewriter versus a wordprocessor is that, on the latter, you do not hit a carriage return until you reach the end of a *paragraph*. The computer will later break your text into individual lines when printing it on paper. Typing becomes more convenient and faster.

Related to this is a "parsing" feature whereby a word will not break at the rightmost side of the screen. If you type through the right side (your word flows onto the next line), Letter Perfect jumps the entire word down to the start of the next line. Some word processors parse, some do not. It is sometimes thought that parsing makes it easier to read a video text since no words are ever broken in half. The majority of our panelists felt, however, that the jumping action during typing is more distracting than the minor inconvenience created by randomly hyphenated words on screen. What is more, parsing wastes screen space.

After, or during, text entry a process called editing is used to correct errors, change wording, move paragraphs, etc. Editing using "LETTER PERFECT" is a simple process identical to editing BASIC programs. The cursor is moved using both the control key and a cursor direction key. Then characters are inserted/deleted using the control and insert/delete keys.

More sophisticated editing capabilities are also provided:

Search — Automatically searches for a specific text string. Allows easy location of certain area of text.

Search/Replace — Automatically searches for a specific string and replaces it with another string. Very useful for changing misspelled names, etc.

Scroll 1 Page — Automatically displays the next complete page of text. This is more convenient than line scrolling which is also available.

Go To End/Beginning Of Line — Moves the cursor to the end/beginning of the current line. This saves time in moving the cursor.

Also provided is the ability to edit blocks (paragraphs or as much as two pages) of information. This allows moving paragraphs within the text without retyping. This capability is useful.

The General Overview

•**Panelist #1:** "This is my first use of a word processor as a writer. What do I think of it? On that score, the only thing I can say is sensational.

After this first effort, I am fully prepared to kiss those old manual Underwoods and Royals goodbye.

But, remember, I am doing this review from the standpoint of a professional writer, not a businessman or an engineer. For the kind of final polishing I want to do, the editing functions of this program are everything I could ask for. Formatting is not particularly important to me, since most of the writing I do consists of fairly simple blocks of text.

For writing articles, papers, simple business letters, and the usual run of home uses, I find the Letter Perfect program to be quite adequate. It also happens to be fun to use, something which is a lot more important than it sounds.

Like many people who get paid for writing, I find one of the hardest things in the world to do, and one of the easiest things to avoid, is just plain sitting down and writing. Since I've had this program in my home, I've discovered that whenever a new idea comes to me, I can't wait to get at it."

•**Panelist #2:** "I feel that Letter Perfect (with exceptions such as parsing, no upscrolling, etc.) makes extensive and positive use of the computer as a writing tool. I would wonder, though, if the Atari, with its 1/2 printer-page screen limitation, would be the best computer to use for lengthy writing tasks. However for shorter, more common, word processing (essays, letters, mailings and the like), Letter Perfect on the Atari does an admirable job."

•Panelist #3: "Space does not permit a discussion of all the features of "LETTER PERFECT" or a complete description of the keystrokes necessary to accomplish the functions. However, let me say that "LETTER PERFECT" is a very powerful word processor which has been written with ease of operation in mind and should be considered by Atari owners with word processing applications."

Letter Perfect
LJK Enterprises, Inc.,
P.O. Box 10827
St. Louis, MO 63129
$149.00

An Overview of Textwizard is currently being prepared. Watch for it in an upcoming issue of **COMPUTE!**.

Letter Perfect Atari Functions

KEY	FUNCTION
CTRL A	GO TO BEGINNING OF LINE
CTRL B	BOLDFACE TOGGLE
CTRL C	CENTER NEXT LINE
CTRL D	DELIMITING CHARACTER
CTRL E	(END) GO TO END OF TEXT
CTRL F	FORMAT LINE
CTRL G	FOOTER
CTRL H	HEADER
CTRL I	IMPROVE TEXT
CTRL J	(JOIN) ADD TO BUFFER
CTRL K	(KILL) DELETE BUFFER
CTRL L	(LIFT) INSERT FROM BUFFER
CTRL M	MOVE TO BUFFER
CTRL N	(NEXT) DELETE NEXT BLOCK
CTRL O	(ON, ON, ON) CONTINUOUS SCROLL
CTRL P	FORCED END OF PAGE
CTRL Q	SCROLL ONE PAGE FORWARD
CTRL R	(REPLACE) SEARCH AND REPLACE
CTRL S	(SEARCH) SEARCH ONLY
CTRL T	(TOP) GO TOP OF SCREEN
CAPS LOWR	SHIFT LOCK
CTRL U	UNDERLINE TOGGLE
CTRL V	SPECIAL PRINT CHARACTERS
CTRL W	DELETE ALL BEFORE CURSOR
CTRL X	DELETE ALL TEXT
CTRL Y	DELETE ALL AFTER CURSOR
CTRL Z	GO TO END OF LINE
ESC	EXIT EDITOR
CTRL ↑	MOVE CURSOR UP
CTRL ↓	MOVE CURSOR DOWN
CTRL ←	MOVE CURSOR TO LEFT
CTRL →	MOVE CURSOR TO RIGHT
(RETURN)	INSERT CARRIAGE RETURN
sft-DEL	DELETE NEXT LINE
sft-INS	INSERT LINE AT CURSOR
CTRL TAB	CLEAR TAB AT CURSOR
sft-TAB	SET TAB AT CURSOR
CTRL DEL	DELETE A CHARACTER
CTRL INS	INSERT A CHARACTER
sft-CLEAR	GO TO BEGINNING OF TEXT
DEL	DELETE LAST CHARACTER
TAB	TAB TO NEXT TAB STOP

SPECIAL PRINT CHARACTERS

* SUBSCRIPT $n PRINT STRING n
^ SUPERSCRIPT #n PRINT NUMBER n

FORMAT REPRESENTATIONS		DEFAULT VALUES	
d	default values	d	(no number needed)
r	reset standard	r	(no number needed)
t	top margin	t	5 spaces
m	left margin	m	10 spaces
j	justification	j	1 (justify)
w	set line width	w	64 characters
l	line spacing	l	l (single spacing)
p	printed lines/page	p	56 lines
s	stop	s	0 (no stop)
f	set type font	f	0 (10 cpi)
a	margin adjust	a	0 (no adjust)
b	bottom margin	b	5 spaces
n	set page number	n	0 (not printed)

Editor's Note: The manufacturer provided the following updates, now included in the standard Letter Perfect 2.0. Our review panelists did not work with version 2.0. — RTM

Manufacturer's Update

Letter Perfect Version 2.0 differs from the earlier version in the following manner:

1. Fonts — The following enhancements have been made with regard to the two different printers supported by this program.

A. Atari 825 Printer and Centronics 737-739 have the following changes. The PROPORTIONAL FONT of this printer is not the default font in the program. The PROPORTIONAL FONT is right justified as a default value. Font 1 is the condensed font of 16.7 characters per inch and Font 2 is now the 10 character per inch font. All of these fonts are right justified as a default value, and can be used as indicated in the manual. Boldface or expanded print can now be used within the body of a line without regard to other types of fonts also appearing in that line. Right justification will remain in effect. The left margin is now set at a default value of 12 and the width defaults to 78. The use of the adjusted margin may cause minor difficulties. The left justification of the adjusted margin may cause some variance because all spaces in the margin are twelve dots wide. This font allows for variation in the individual character and therefore the alignment may not be as straight as desired. To obtain optimal results it is best to experiment.

B. Epson MX-80, MX-100, GRAFTRAX — These printers may now be used with this program with the following changes noted. If you have the new GRAFTRAX Proms as sold by Epson, you may use the additional characteristics of Italics characters. Italics are turned on by using the superscript command (CTRL-V:) as described in the manual. Italics are turned off by doing a subscript (CTRL-V*). Underlining may be performed on short words by using the underline toggle (CTRL-U). If an excess number of characters are underlined the printer may time out. Boldface may now be used in the main body of the text.

2. Header and Footer Spacing — The spacing between headers and footers and the main body of the text is now variable. The defaults are 4 spaces for headers and 4 spaces for footers. They may be changed by entering a lower case h followed by the spacing for a header in a format line (starting with a CTRL-F) and footer spacing can be changed by using a lower case z in a format line. ©

Atari Disk File Dump

Robert W. Baker
Atco, NJ

Here's another handy utility program for the Atari 800, for anyone with an 810 or 815 disk drive. It provides a hexadecimal dump of *any* disk file along with an ASCII representation of any valid ASCII characters. With this program you can quickly examine how Atari BASIC stores programs and data on diskette. This could be extremely valuable when debugging programs that write or read disk data files.

The program was written to provide only printed output, since most dumps would be too large for the display. Also, the printed output was formatted for the 80-column Atari 825 printer. If you only have the 40-column Atari 820 printer, the program can be easily modified for the shorter line length. Simply shorten the heading lines in program lines 302 and 310, stopping at "7." Then change the loop count in line 600 from 16 to 8 to print eight bytes per line instead of 16. You might also want to shorten the filename printout and remove the CHR$(15) and CHR$(14) from line 300. That should be all you have to change for the 40-column format.

To use the program, enter the filename for the disk file to be dumped, such as: FILEDMP.BAS The drive number always defaults to 1. The filename will be printed at the top of the listing along with a byte count heading. The dump will then follow with 16 bytes per line. At the end of each line is the ASCII representation of any data that is a valid ASCII character. All unprintable characters are printed as periods in the ASCII field.

As the dump is being printed you can press any key on the keyboard to halt the output. Then press "C" to continue the dump, "R" to restart and select another file to be dumped, or "S" to stop the program and terminate the dump.

If the end of the disk file is reached, the program will indicate on the dump the end of file (EOF) was reached. You may want to note the TRAP statement in program line 228. When an error is detected, the program branches to line 900. A PEEK of location 195 checks for error number 136. If an end of file (error #136) was detected, the program returns to ask for another filename. Otherwise, the detected error is indicated on the display and the program terminates after closing all files.

```
10 REM ******************************
25 REM HEX DISK FILE DUMP
35 REM BY: ROBERT W. BAKER
40 REM 15 WINDSOR DR, ATCO NJ 08004
60 REM ******************************
70 GRAPHICS 0
100 DIM H$(16),S$(16),F$(16)
110 H$="0123456789ABCDEF"
130 F$="D1:"
150 OPEN #1,4,0,"K"
200 PRINT CHR$(125);"     H E X   F I L
 E   D U M P   ":? :?
210 PRINT "ENTER DISK FILE NAME"
220 INPUT S$:F$(4,14)=S$
225 IF S$="" THEN 800
228 TRAP 900
230 OPEN #5,4,0,F$
280 OPEN #2,8,0,"P"
290 PRINT CHR$(125);"DEPRESS ANY KEY TO
HALT PRINTER":PRINT #2
300 PRINT #2;CHR$(15);"HEX DUMP OF FILE
-->    ";F$;CHR$(14):PRINT #2
302 PRINT #2;"BYTE#  0  1  2  3  4  5  6
  7  8  9  A  B  C  D  E  F  "
310 PRINT #2;"--------------------------
--------------------------"
320 POKE 764,255
370 V=INT(A/256):GOSUB 1000
375 V=A-(V*256):GOSUB 1000
380 PRINT #2;": ";
400 S$=""
600 FOR X=1 TO 16:GET #5,V
610 GOSUB 1000:PRINT #2;" ";
615 S$(X)=".":IF (V>31) AND (V<123) THEN
 S$(X)=CHR$(V)
620 A=A+1:NEXT X:PRINT #2;"    ";S$
640 IF PEEK(764)=255 THEN 370
650 GET #1,X
700 POKE 752,1:PRINT
705 PRINT "CONTINUE, RESTART, OR STOP (C
,R,S) ?";
710 GET #1,X:IF X=67 THEN 290
730 IF X=82 THEN 990
740 IF X<>83 THEN 710
800 POKE 752,0:CLOSE #1:CLOSE #2:CLOSE #
5:END
900 V=PEEK(195):IF V<>136 THEN PRINT "ER
ROR# ";V:GOTO 800
910 FOR A=X TO 16:PRINT #2;"   ";:NEXT A
920 PRINT #2;"    ";S$:PRINT #2:PRINT #2
;"EOF"
990 CLOSE #2:CLOSE #5:GOTO 200
1000 H=INT(V/16):L=V-(H*16)
1010 PRINT #2;H$(H+1,H+1);H$(L+1,L+1);
1020 RETURN
```

©

Atari Program Library

Ron and Lynn Marcuse
Freehold, NJ

One of the most difficult aspects of owning a home/personal computer is maintaining an accurate catalog of programs and data files. We first attacked this problem through the use of a home-grown data base which handled the program library in addition to other data. But, alas, it became too time consuming to manually update the data base as new programs were added or changes made. The logical extension to this concept was to automate the cataloging process. To accomplish this, one must compare the index of the disk to the program library file, adding or deleting records on the file as the comparison warrants. The ease of OPENing the ATARI's DOS directory from BASIC greatly facilitated the programming, but more on this later.

Besides the automatic cataloging function, other required features were:

1. Listing the directory from the program.

2. Cataloging Non-DOS disks.

3. Inquiry, Browse and formatted Print output of catalog records.

4. Maintaining other data not supplied by the disk directory (source, description, type, and date).

5. Sorting the file on any field (in machine language, for speed).

6. An auto-locate function to RUN any program.

7. Variable search criteria to locate any program.

8. Creating an internal label to

Table 1. List of variables.

NAME	SIZE	DESCRIPTION
AP$	21	"ATARI PROGRAM LIBRARY"
REC$	62	Program Library Record
VOL$	4	Disk (or other) Volume Number
DSN$	12	Data Set Name (Filespec)
DES$	22	Description
TYP$	7	Type (Game, Data, etc.)
SRC$	7	Source
DAT$	6	Date (Entry, Version, etc.)
SEC$	3	Number of Sectors
X$	var	Input String (Main Prog and Sort)
IN$	20	Temporary String Storage
PL$,PC$		Filespecs (D:PROGLIB.DB & D:DISK.CAT)
SV$	20	Search Value
SS,SE		Search (and Sort) Start, End Positions
SK		Search Key
B		Transaction Type
R		Return Line Number from TRAPped Error
ST		IOCB STATUS Value
I,J,L,N		Counting for Loops, Lines, etc.
NP		Number of Files in Directory
IO		Input/Output Type
SEC,BYT		Sector, Byte values for NOTE, POINT
P		Output Type (1=Inqy, 2=Browse, 3=Print)
EOF		Record Counter

Table 2. Key conversion (Epson MX-80 printer).

SYMBOL	ASCII VAL (Dec)	KEY SEQUENCE (ATARI)	DESCRIPTION
[A]	0	CON; , (comma)	Null; End of Tab Set Seq
[B]	9	CON; I	Horizontal Tab
[C]	14	CON; N	Print Double Width Characters
[D]	27,68	ESC;ESC;D	Set Tab (followed by Tab Positions and NULL Char)
[E]	253	ESC;CON;2	Console Bell
[F]	125	ESC;CON;CLR	Clear Screen
[G]	29	ESC;CON; =	Move Cursor Down
[H]	10	CON; J	Line Feed

Note: The string of X's in line 3090 are the printer tabs. Type the ASCII characters for these decimal values: 6,20,39,53,62,70 (i.e. 6 is CON;F)

Program 1.

```
10 DIM AP$(21),PC$(10):AP$="ATARI PROGRAM LIBRARY"
12 REM ** RON MARCUSE, FREEHOLD NJ  **
30 OPEN #2,4,0,"K:":POKE 82,0:? "[E]"
50 DIM REC$(62),VOL$(4),DSN$(12),DES$(22),TYP$(7),SRC$(7),DAT$(6),SEC$(3)
60 DIM X$(500),IN$(20),SV$(20),PL$(12):PC$="D:DISK.CAT":PL$="D:PROGLIB.DB"
100 GRAPHICS 0:POKE 16,64:POKE 53774,64:? :? ,AP$:? ,"[G] CATALOG OPTIONS"
110 ? "[G] 1 AUTO CATALOG        5 UPDATE RECORD"
120 ? " 2 LIST DIRECTORY       6 SORT LIBRARY"
130 ? " 3 ADD DISK (NON DOS)  7 RUN PROGRAM"
140 ? " 4 INQUIRY/LIST         8 END SESSION"
160 GOSUB 6900:TRAP 160:B=VAL(CHR$(A)):TRAP 40000:IF B<1 OR B>8 THEN 160
170 B=B+1:IF B>3 THEN 500
180 ? :IF B=3 THEN 300
190 ? " INSERT DISK TO BE CATALOGED":GOSUB 6910
200 TRAP 240:OPEN #3,4,0,PC$:TRAP 40000:INPUT #3,IN$
210 IF LEN(IN$)<14 OR IN$(1,10)<>PC$ THEN ? " ERROR- ";PC$:GOTO 250
220 B=1:VOL$=IN$(11,14):GOTO 290
240 CLOSE #3:R=200:STATUS #3,ST:IF ST<>170 THEN 9000
250 ? " ENTER DISK # (DNNN) =>";:INPUT VOL$:IF LEN(VOL$)=0 THEN 100
260 R=260:TRAP 9000:OPEN #3,8,0,PC$:? #3;PC$;VOL$
290 CLOSE #3:XIO 35,#3,0,0,PC$
300 R=300:TRAP 9000:OPEN #3,6,0,"D:*.*":L=0
310 TRAP 400:INPUT #3,IN$:TRAP 40000
320 IF LEN(IN$)<17 THEN 400
330 IF B=3 THEN ? ,IN$:L=L+15:GOTO 310
```

automatically identify a disk. All of these goals were achieved in the program which, incidentally, requires at least 24K RAM, one disk drive (810 or 815) and DOS II.

You did notice that there is a "II" after "DOS." ATARI has finally released the new version and, to say the least, it is a vast improvement. Not that we were unhappy with its predecessor, but it did tend to hide whenever one walked by with a can of "RAID" or "BLACK FLAG." Yes, there are bugs in DOS I, one of which leads the NOTE and POINT commands (needed to update any record) somewhere into *the twilight zone*. This program can be modified to work under DOS I, but the explanation would probably take up the rest of the magazine. If you plan to do any serious file processing, it would be advisable to pick up a copy of DOS II. There are other advantages as well, such as less RAM used through the auto-swap feature (the program and DOS share the same area).

The three listings represent the main program, the sort program and the machine language sort routine. The sort program is executed by a RUN statement, allowing the DIMensioning of the rather large string necessary to sort the file in. It loads the file into X$ and calls the machine language sort through the USR function in line 70. You may POKE the routine into storage using the third BASIC program. You must do a BINARY SAVE (DOS II function "K") with AUTORUN.SYS as the file name, 0600 and 066D (hex) as the starting and ending addresses. A possible alternative to this would be to key the FOR/NEXT loop and DATA statements into the BASIC sort program, with the loop as line 14.

The main program begins with the DIMensioning of strings and OPENing of the keyboard in lines 30-60. The strings and other variables are detailed in the accompanying table. Graphics mode 0 is set and the ‹BREAK› key is disabled (POKE 16,64:POKE 53774,64) in line 100. This begins the start of the primary option menu as well. An option (from 1 to 8) is selected and held in variable B. Note that B is increased by 1 in line 170, allowing "Auto-catalog" to split into two functions depending on whether or not the disk has already been cataloged. A "D:DISK.CAT" file, containing the volume number, is written on each as an internal label. If this file is found on any disk, the volume number is retrieved and a "re-catalog" function is performed. If not found, a new disk is assumed and the volume number is requested in line 250. One important note: there is a liberal use of subroutines in the program and, for expediency, we will get into them later.

The options that do not require a DOS directory (B›3) are shunted to line 500. Those that do are sent through the routine (lines 300-430) that OPENs the disk directory (IO = 6) and stores it as substrings of X$. The file name is translated into the more familiar format FILENAME.EXT in lines 340-360. File names equal to DOS.SYS, DUP.SYS, MEM-.SAV or DISK.CAT are dropped at this point (Why catalog them?). Line 500 (ON B GOTO) then transfers program control to the routine that will process the requested option.

The "Re-catalog" function (B = 1, lines 1000-1110), after OPENing the library file for both input and output operation (IO = 12), extracts all records having that volume number. These are then compared to the directory string (X$). Changes and deletions are posted to the file using the NOTE and POINT commands. An asterisk (*) is moved to each directory member that was successfully matched to the library. At the conclusion of this procedure, control is passed to "Auto-catalog."

The "Auto-catalog" function

```
340 FOR I=3 TO 10:IF IN$(I,I)<>" " THEN NEXT I
350 DSN$(1,8)=IN$(3,10):DSN$(9,9)=" ":DSN$(I-1)=IN$(11,13)
360 IF DSN$(I-1)<>"   " THEN DSN$(I-2,I-2)="."
370 IF DSN$="DOS.SYS" OR DSN$="DUP.SYS" OR DSN$="MEM.SAV" OR DSN$="DISK.CAT" THEN 310
380 X$(L+1,L+12)="            ":X$(L+1,L+12)=DSN$
390 X$(L+13,L+15)=IN$(15,17):L=L+15:GOTO 310
400 CLOSE #3:NP=L/15
410 ? :? " FILES FOUND= ";NP;",  FREE SPACE= ";IN$(1,3)
420 IF B<3 THEN ? " DISK IS # ";VOL$:? " INSERT D:PROGLIB.DB"
430 GOSUB 6910:IF NP=0 THEN 100
500 ON B GOTO 1000,1200,100,2000,3000,4000,5000,5500,900
900 GRAPHICS 0:END
950 GOSUB 6250:GOTO 100
960 POP :GOTO 100
990 FOR I=1 TO 300:NEXT I:RETURN
1000 REM ** RE-CATALOG
1010 IO=12:GOSUB 6200:SV$=VOL$:SK=1:SS=1:SE=4:EOF=0:L=0
1020 GOSUB 7000:IF SK=9 THEN 1190
1030 FOR N=1 TO NP:IF X$(N*15-14,N*15-3)=DSN$ THEN 1100
1040 NEXT N:REC$(62)="D":? " ";DSN$;" DELETED ON ";VOL$:GOTO 1110
1100 REC$(59,61)=X$(N*15-2,N*15):X$(N*15,N*15)="*":L=L+1
1110 POINT #3,SEC,BYT:? #3;REC$:GOTO 1020
1190 GOSUB 6250:IF L=NP THEN 100
1200 REM ** AUTO CATALOG
1210 IO=9:GOSUB 6200
1220 FOR N=1 TO NP:IF B=1 AND X$(N*15,N*15)="*" THEN 1240
1230 DSN$=X$(N*15-14,N*15-3):SEC$=X$(N*15-2,N*15):GOSUB 6800
1240 NEXT N:GOTO 950
2000 REM ** MANUAL ADD
2010 IO=9:GOSUB 6200:GOSUB 6800:GOTO 950
3000 ? "[F][G] ";AP$;" - INQY/LIST":N=8:GOSUB 6500:?
3020 ? " OUTPUT:","1. INQUIRY":? ,"2. BROWSE":? ,"3. LISTING"
3040 GOSUB 6900:TRAP 3040:P=VAL(CHR$(A)):TRAP 40000:IF P<1 OR P>3 THEN 3040
3050 L=0:IO=4:GOSUB 6200:EOF=0:IF P<3 THEN 3100
3090 R=3090:TRAP 9020:OPEN #4,8,0,"P:":TRAP 40000:? #4;"[D]XXXXXX[A]"
3100 GOSUB 7000:L=L+1:IF SK>8 THEN 3300
3110 ON P GOTO 3120,3160,3200
3120 GOSUB 7600:?
3130 L=0:? " (E=END) OR";:GOSUB 6910:IF CHR$(A)="E" THEN 950
3140 GOTO 3100
3160 IF L=1 THEN ? "[F][G]VOL  FILE ID       DESCRIP":?
3170 ? VOL$;" ";DSN$;DES$:IF L=19 THEN 3130
3180 GOTO 3100
3200 IF L>1 THEN 3220
3205 ? #4;"[B][C]   ATARI PROGRAM LIBRARY[H]"
3210 ? #4;"DISK#[B]PROG/FILE ID[B] DESCRIPTION[B]TYPE[B]SOURCE[B]DATE  SECTORS[H]"
3220 ? #4;VOL$;"[B]";DSN$;"[B]";DES$;"[B]";TYP$;"[B]";SRC$;"[B]";DAT$;"[B]";SEC$
3230 IF L>55 THEN L=0:? #4;CHR$(12)
3240 GOTO 3100
3300 IF P=3 THEN CLOSE #4
3310 IF P=2 THEN GOSUB 6910
3320 GOTO 950
4000 ? "[F][G] ";AP$;" - RECORD UPDATE":N=8:GOSUB 6500
4010 IO=12:GOSUB 6200:EOF=0
4020 GOSUB 7000:IF SK>8 THEN 950
4030 GOSUB 7600:? :? " TYPE FIELD # TO UPDATE, D TO DELETE"
4050 GOSUB 6910:IF CHR$(A)="D" THEN REC$(62)="D"
4060 TRAP 4300:C=VAL(CHR$(A)):TRAP 40000:IF C<1 OR C>7 THEN 4300
4100 RESTORE 9910:FOR I=1 TO C:READ IN$:NEXT I:?
4110 ? " ENTER NEW":GOSUB 6040+C:GOTO 4030
4300 GOSUB 6100:POINT #3,SEC,BYT:? #3;REC$:GOTO 4020
5000 ? "[F][G] ";AP$;" - SORT/COMPRESS":N=7:GOSUB 6500
```

(lines 1200-1240) OPENs the file for output (append, IO=9) operation. All new entries, as per the X$ directory, are shuffled through the proper subroutines and then added to the file. Option 1 (B=1 or 2) travels this route, with (1) being "Re-cataloged" first. The significance of the asterisk on matched directory members becomes apparent in the bypass on line 1220. The file is then CLOSEd and we return to the option menu.

Option 2 (B=3) consists of displaying the DOS directory and then returning to the menu. The remaining options (B>3), not requiring any help from the directory, go straight into their respective procedures. Option 3 (B=4) handles the manual addition of library records to the file. This could be utilized for Non-DOS disks or even cassette tapes. Line 2010, helped by several subroutines, does all of the processing.

The inquiry and print option (B=5) is handled in lines 3000-3320. The search strategy (GOSUB 6500) and output mode are selected, the file is OPENed for input (IO=4) and, if applicable, the print tabs are set. The file is actually read in the subroutine at line 7000, with only the records matching the search key being passed back. An inquiry goes to line 3120, the browse (19 records per screen) to line 3160 and print (55 records per page) to line 3200. For the multiple record options, the variable L is the line counter. When end of file is reached at line 7010, the search key (SK) is set to "9" indicating completion, the file and printer are closed, and we are back at the menu.

The Update option (B=6, lines 4000-4300) would be used for changing or deleting library records. The search key is selected, file OPENed (IO=12) and records read like the inquiry, but here only the full record is displayed. At this point (line 4030), typing the field number will cause

```
5010 ? " TYPE Y TO SORT ON FIELD # ";SK:GOSUB 6910
5020 IF CHR$(A)<>"Y" THEN 100
5040 POKE 207,SS-1:POKE 208,SE-1
5050 ? " LOADING SORT PROGRAM":RUN "D:PROGSORT"
5500 REM RUN PROG
5510 ? :? " ENTER PROG ID ==>";:INPUT SV$:IO=4:IF LEN(SV$)=0 THEN 100
5520 GOSUB 6200:EOF=0:SS=5:SE=4+LEN(SV$):SK=2:GOSUB 7000:IF SK=9 THEN 950
5530 ? :? " INSERT DISK ";VOL$;" TO RUN ";DSN$
5540 ? :? " TYPE 'Y' TO RUN":GOSUB 6910:IF CHR$(A)<>"Y" THEN 950
5550 IN$(3)=DSN$:IN$(1,2)="D:"
5560 ? :? " LOADING ";IN$:TRAP 5570:RUN IN$:TRAP 40000
5570 ? :? " ";IN$;" NOT ON DISK":GOSUB 990:GOTO 950
6000 ? "[F][G] TO ADD ";DSN$;", ENTER:"
6010 RESTORE 9910:FOR I=1 TO 7:READ IN$
6020 IF B<3 AND (I=1 OR I=2 OR I=7) THEN 6040
6030 ? :GOSUB 6040+I
6040 NEXT I:RETURN
6041 ? "|----| ";IN$:INPUT VOL$:RETURN
6042 ? "|------------| ";IN$:INPUT DSN$:RETURN
6043 ? "|----------------------| ";IN$:INPUT DES$:RETURN
6044 ? "|-------|";IN$:INPUT TYP$:RETURN
6045 ? "|-------| ";IN$:INPUT SRC$:RETURN
6046 ? "|------| ";IN$:INPUT DAT$:RETURN
6047 ? "|---| ";IN$:INPUT SEC$:RETURN
6100 FOR I=1 TO 61:REC$(I,I)=" ":NEXT I
6110 REC$(1,4)=VOL$:REC$(5,16)=DSN$:REC$(17,38)=DES$:REC$(39,45)=TYP$
6120 REC$(46,52)=SRC$:REC$(53,58)=DAT$:REC$(59,61)=SEC$:RETURN
6200 R=6200:TRAP 9000:IF IO<>4 THEN XIO 36,#3,0,0,PL$
6210 OPEN #3,IO,0,PL$:TRAP 40000:RETURN
6250 R=6250:TRAP 9000:CLOSE #3
6260 IF IO<>4 THEN XIO 35,#3,0,0,PL$
6270 TRAP 40000:RETURN
6500 ? "[G] KEY:":RESTORE 9910
6510 FOR I=1 TO N:READ IN$:? ,I;" ";IN$:NEXT I:? ,"E END":GOSUB 6900
6540 TRAP 960:SK=VAL(CHR$(A)):TRAP 40000:IF SK<1 OR SK>N THEN 960
6550 IF SK=8 THEN 6590
6560 RESTORE :FOR I=1 TO SK:READ SS,SE:NEXT I:IF B=7 THEN 6590
6570 ? " ENTER VALUE";:INPUT SV$:IF LEN(SV$)<1 THEN 960
6580 IF SS+LEN(SV$)-1>SE THEN 6570
6585 SE=SS+LEN(SV$)-1
6590 RETURN
6800 GOSUB 6000:GOSUB 6100:REC$(62)="$":GOSUB 7600
6810 ? "[G] TYPE 'Y' IF OK ":GOSUB 6910:IF CHR$(A)<>"Y" THEN 6800
6820 ? #3;REC$:RETURN
6900 ? :? " SELECT OPTION ==>";:GET #2,A:? CHR$(A):RETURN
6910 ? " PRESS ANY KEY TO CONTINUE";:GET #2,A:? CHR$(A):RETURN
7000 IF B=1 OR B=6 THEN NOTE #3,SEC,BYT
7010 TRAP 7060:INPUT #3,REC$:TRAP 40000:IF REC$(62)="D" THEN 7000
7020 IF SK=8 THEN 7040
7030 IF SV$<>REC$(SS,SE) THEN 7000
7040 EOF=EOF+1:VOL$=REC$(1,4):DSN$=REC$(5,16):DES$=REC$(17,38):TYP$=REC$(39,45)
7050 SRC$=REC$(46,52):DAT$=REC$(53,58):SEC$=REC$(59,61):RETURN
7060 SK=9:? :? " RECORDS FOUND= ";EOF:GOSUB 990:RETURN
7600 ? "[F][G]";:RESTORE 9910:FOR I=1 TO 7
7610 READ IN$:? " ";I;" ";IN$;:GOSUB 7610+I:NEXT I:RETURN
7611 ? VOL$:RETURN
7612 ? DSN$:RETURN
7613 ? DES$:RETURN
7614 ? TYP$:RETURN
7615 ? SRC$:RETURN
7616 ? DAT$:RETURN
7617 ? SEC$:RETURN
```

updating of that field and "D" the deletion of the record. Any other character will write (using POINT) the record.

The sort (B = 7, lines 5000-5050), as stated earlier, is executed through a RUN "D:PROGSORT statement. Before this is done, the sort key is selected and its offset (beginning and ending positions) is POKEd into locations 207 and 208 (decimal) for use by the machine language program. These addresses, as well as those on page 6 (1536-1791) containing the sort program, are safe from the ravaging effects of RUN, LOAD and NEW. The RUN PROGRAM option (B = 8, lines 5500-5570) adds a little touch of class to the library. By inserting the program name when requested, it will tell you which disk to load and will then RUN it. Obviously, it will only function with BASIC programs that have been SAVEd on disk. The final option (B = 9) terminates the system. The BREAK key, disabled in line 100, could have disastrous effects on the file if used to end the program at the wrong time.

A few notes on the subroutines that do most of the work in the program. The routine starting at line 6000 is used for the input of data for both adding and updating records. The labels for the individual fields, being DATA statements, are READ during the FOR/NEXT loop in line 6010. The library record (REC$) is built from the individual fields at line 6100. Lines 6200 and 6250 OPENs and CLOSEs the file. The search and sort keys are built at line 6500, using the same DATA statements as above for the headings. The positions of the fields are contained in another DATA statement. Lines 6900 and 6910 are prompts. The library record is READ and moved into its elements in line 7000-7060. The variables SV$, SS, SE and SK are used in the search process. The full screen display of the record is taken care of by 7600-7617, once again using the heading DATA statements. Finally, the error routines for the disk drive and printer are found at lines 9000 and 9020. The variable R is the return line number.

As there are several unprintable characters in the programs, we have substituted others in their place in the BASIC listings. Take a peek at the conversion table before typing. For a better visual effect, the characters enclosed by a box should be typed in reverse video. The two programs call each other by name, so please save them by the names "D:PROGLIB" and "D:PROGSORT". Some last thoughts: The machine language sort routine will not be in storage unless you put it there by either powering up (AUTORUN.SYS will boot in) or doing a DOS binary load. Create the PROGLIB file by typing (in direct mode):

OPEN #3,8,0, "D:PROGLIB.DB": CLOSE #3:XIO 35,#3,0,0, "D:PROGLIB.DB".

Good luck. ©

```
9000 STATUS #3,ST:CLOSE #3:? " CHECK DISK DRIVE ERROR ";ST:GOSUB 6910:GOTO R
9020 STATUS #4,ST:CLOSE #4:? " CHECK PRINTER ERROR ";ST:GOSUB 6910:GOTO R
9900 DATA 1,4,5,16,17,38,39,45,46,52,53,58,59,61
9910 DATA DISK # ,PROG ID ,DESCRIP ,TYPE   ,SOURCE ,DATE   ,SECTORS ,ALL RECORDS
```

Program 2.

```
10 REM ** ATARI PROGRAM LIBRARY SORT **
11 REM ** R MARCUSE
15 A=FRE(0)-800
20 DIM X$(A),REC$(62),AP$(12):AP$="D:PROGLIB.DB":? " LOADING FILE"
30 TRAP 130:OPEN #3,4,0,AP$:N=0
40 TRAP 60:INPUT #3,REC$:TRAP 40000:IF REC$(62)="D" THEN 40
50 N=N+1:X$(N*62-61,N*62)=REC$:GOTO 40
60 CLOSE #3:? " RECORDS LOADED= ";N;", BEGIN SORT"
70 IF N>1 THEN A=USR(1536,ADR(X$),N)
80 ? "[E] SORT FINISHED, SAVING FILE"
90 XIO 36,#3,0,0,AP$:OPEN #3,8,0,AP$
100 FOR I=1 TO N:REC$=X$(I*62-61,I*62):? #3;REC$:NEXT I
110 CLOSE #3:XIO 35,#3,0,0,AP$
120 ? "[E] LOADING PROGLIB":RUN "D:PROGLIB"
130 STATUS #3,ST:CLOSE #3:? " CHECK DISK ERROR ";ST
140 ? " PRESS RETURN TO CONTINUE":INPUT REC$:GOTO 30
```

Program 3.

```
10 REM ** PROGLIB MACHINE LANG SORT  **
11 REM **      R MARCUSE             **
12 REM BINARY SAVE WITH FILESPEC=AUTORUN.SYS
20 FOR I=0 TO 109:READ A:POKE 1536+I,A:NEXT I
100 DATA 104,104,133,216,104,133,215,104,133,213,104,133,212,169,0,133,209,133
110 DATA 214,162,1,165,215,133,205,165,216,133,206,24,165,205,133,203,105,62
120 DATA 133,205,165,206,133,204,105,0,133,206,164,207,177,205,209,203,144,11
130 DATA 240,2,176,28,196,208,176,24,200,144,239,169,1,133,209,160,62,136,177
140 DATA 205,72,177,203,145,205,104,145,203,192,0,208,241,232,224,0,208,2,230
150 DATA 214,228,212,208,188,165,213,197,214,208,182,165,209,201,0,208,160,96
```

MATCH — A Game Of Memory And Timing

Ron Walker
Smithville, Ontario

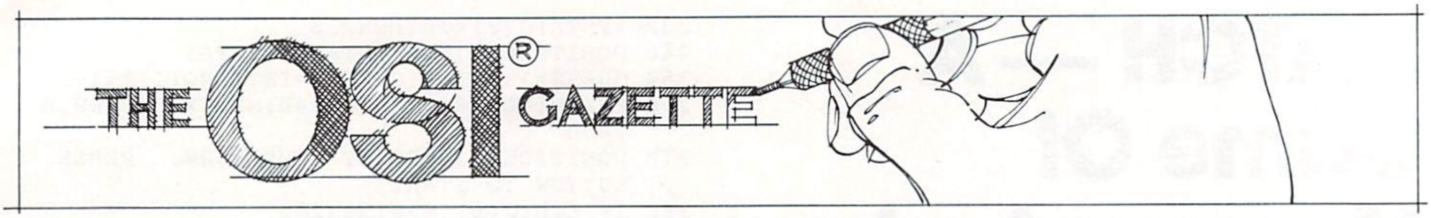
There are nine different skill levels to this game, with level one being the most difficult and nine being the easiest.

The object of the game is to repeat a pattern of musical notes and cursor positions that are randomly selected by the computer. Sounds simple, but at the highest level, the notes occur so rapidly that only the sharpest of minds can cope with it. Try it.

NOTE: Any numbers in brackets in quotes in the program listing are the number of blank spaces to leave. The "rvs" means leave one reverse space. Also, any words on brackets are what they say they are. e.g. PRINT "(clear" means to clear the screen (esc-shift-clr) etc. Good luck.

```
100 OPEN#1,4,0,"K:"
110 GRAPHICS 0:POKE752,1:DIM NT(4),HOR(4)
    ,VIR(4),CHIME(50)
120 DIMA$(40):A$="     (39)       "
130 POSITION15,6:PRINT"MATCH"
140 POSITION10,16:PRINT"ENTER LEVEL (1-9)
    "
150 GET#1,VAR:TRAP510:LEVEL=VAL(CHR$(VAR)
    )*10
160 FORX=1TO4:READ A,B,C:NT(X)=A:HOR(X)=B
    :VIR(X)=C:NEXTX
170 DATA 10,19,4,50,9,12,100,29,12,150,19
    ,21
180 PRINT " (CLEAR)": FORX=5TO20:POSITION
     19,X:PRINT"|":NEXTX
190 FORX=10TO20:POSITIONX,12:PRINT"-":NEX
    TX:POSITION 19,12:PRINT"+"
200 POSITION2,0:PRINT"HERE ARE THE NOTES:
    ":FORDELAY=1TO500:NEXTDELAY:POSI
    TION2,0:PRINTA$
210 FORX=1TO4:POSITION HOR(X),VIR(X):PRIN
    T"RVS":GOSUB470:POSITIONHOR(X),V
    IR(X):PRINT" "
215 NEXT X
220 POSITION2,0:PRINT"PRESS BUTTON TO STA
    RT WITH YOUR FIRST NOTE":SOUND0,
    0,0,0
230 IFSTRIG(0)<>0THEN230
240 POSITION 0,0:PRINTA$:PRINTA$
250 GUESS=1:CHIME(GUESS)=INT(RND(1)*4)+1
260 FORX=1TOGUESS:GOSUB440:NEXTX:SOUND0,0
    ,0,0
270 POSITION1,22:PRINT"YOUR TURN.  PRESS
    BUTTON TO START"
280 IF STRIG(0) <>0THEN280
290 POSITION 1,22:PRINT"(DELETE LINE) ENT
    ER NOTES"
300 FORX=1TOGUESS:POKE77,0
310 TONE=STICK(0):SOUND0,0,0,0
320 IFTONE=14THENTONE=10:GOTO370
330 IFTONE=13THENTONE=150:GOTO370
340 IFTONE=7THENTONE=100:GOTO370
350 IFTONE=11THENTONE=50:GOTO370
360 GOTO310
370 IFNT(CHIME(X))=TONE THEN GOSUB 440:GO
    TO400
380 GOTO410
400 NEXTX:GOTO480
410 PRINT"(CLEAR)NOT QUITE RIGHT.  YOU GO
    T";GUESS;" NOTES IN A ROW":PRINT
    "TRY AGAIN?"
420 GET#1,VAR:IFVAR=89 THEN GRAPHICS 0:EN
    D
430 GOTO180
440 SOUND0,NT(CHIME(X)),10,10
450 POSITIONHOR(CHIME(X)),VIR(CHIME(X)):P
    RINT"RVS":FORDELAY=1TOLEVEL:NEXT
    DELAY
460 POSITIONHOR(CHIME(X)),VIR(CHIME(X)):P
    RINT" ":RETURN
470 SOUND0,NT(X),10,10:FORDELAY=1TO500:NE
    XTDELAY:RETURN
480 GUESS=GUESS+1:SOUND0,0,0,0:POSITION1,
    22:PRINTA$
490 POSITION1,0:PRINT"OK SO FAR.  NOW I A
    DD ANOTHER ONE":CHIME(GUESS)=INT
    (RND(1)*4)+1
500 FORDELAY=1TO500:NEXTDELAY:POSITION0,0
    :PRINTA$:GOTO260
510 PRINT"A NUMBER!":FORDE=1TO500:NEXTDE:
    PRINT"(UP) (14)":GOTO140
```

©



Calling BASIC Commands From Machine Language Routines

William Taylor
Leavittsburg, OH

While working on a tape operating system (TOS) for my OSI C1P and a Stringy Floppy tape drive, many unknown, but desired, features were needed to interface ROM BASIC and the TOS. First, I wanted the TOS to always have command of BASIC's LOAD and SAVE routines. Second, I wanted always to return to the TOS whenever a BASIC program had been loaded into the BASIC workspace. Third, I wanted to go directly from the TOS and RUN a BASIC program that was in the BASIC workspace. In addition, I wished to exit the TOS to the ML Monitor; write a file directory; store the directory on tape; retrieve the directory; and write or load language tapes into the C1P using file marks.

Since the Stringy Floppy tape drives require that all programs stored on tape have file marks or numbers, I needed to free the C1P from ROM BASIC in order to create files on the tape for all programs stored on the tape. The TOS could be written in machine language. The TOS would generate the file numbers under the control of the user, but interfacing the TOS to ROM BASIC was the problem that I faced and pondered for several weeks. How the OSI ROM BASIC and the TOS were interfaced brought several interesting points to light that could be useful in other programming tasks.

Let me summarize. First, calling BASIC commands and executing BASIC programs can be handled from machine language routines. Also, we may LIST, SAVE, LOAD, and exit BASIC to our machine language routines without any USR function call. How these commands can be executed from a machine language routine will become clear with some new knowledge of how BASIC's interpreter works. Let's start with some facts about the BASIC interpreter and how BASIC commands are executed.

Let's look at BASIC's LOAD and SAVE flags and see how they are used to determine if BASIC programs are to be listed to the CRT or to the Cassette port and if the keyboard or the Cassette input port will be the input device.

BASIC's Immediate Mode Commands

BASIC commands are usually executed when input from the keyboard is entered. For example, when you type RUN followed by a carriage return any BASIC program in the workspace will be executed or start to run, starting at the first line of the program. Notice that I said type RUN! This type of command is known as an immediate mode command. If you had typed a number before the RUN command the C1P would have responded with OK. The program would not run but the line of text would have been saved or entered into the program memory. To understand what happens in either the programming mode or the immediate mode we must know how BASIC interprets the code input by the operator. To do this let's look inside BASIC and examine some of what happens during the course of any type of code execution.

At the beginning of system memory is what has become known as zero page. This memory area consists of the first 256 locations of low memory. OSI BASIC uses this area of memory as a scratch pad. OSI BASIC uses page locations $0013 through $005A as what is known as the BASIC Input Line Buffer. What is the Input Line Buffer? This area of low memory is used by BASIC to temporarily store any input code from the user. The code input by the user in the Input Line Buffer will be examined by BASIC to determine what the code's destiny will be. When the user terminates a line of code with a carriage return, the destination of the code input by the operator depends on two factors. First, if the code began with a line number

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

NEW-NEW—NEW TINY COMPILER

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -,=,*,/, (,) , < , >, Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.

TINY COMPILER — $19.95 on tape or disk

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!

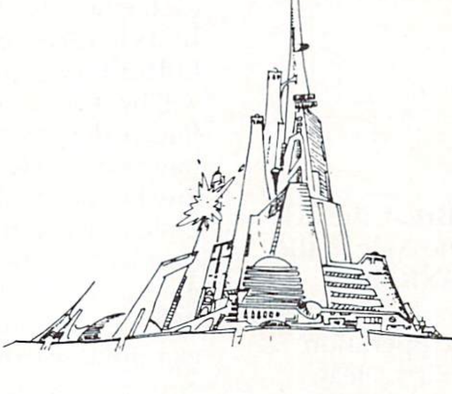
SUPERDISK II

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

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.

NIGHT RIDER - You've seen similar games in the arcades. You see a winding twisting road ahead as you try to make time and stay on the road. NIGHT RIDER uses machine code to generate excellent high speed graphics - by the same author as MINOS.

NIGHT RIDER — $12.95 cassette only

THIEF - Another machine code goody for the C1P cassette only. You must use mobile cannon to protect the valuable jewels in the middle of the screen from increasingly nasty and trigger happy thiefs. Fast action and fun for one or two players. THIEF $13.95 on C1 cassette only!

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.

OSI

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

OSI

this signals BASIC that the code must be saved as a BASIC program line. Second, if the code in the Input Line Buffer does not start with a numeral, then the code represents a BASIC immediate command or some error that the user made while typing at the keyboard. In either of the latter cases, the code will be immediately executed. If the code was a valid command, the command will be executed. If the input was an error, BASIC will respond with Syntax Error.

To demonstrate and reveal the format of the code placed in the Input Line Buffer, please examine the following example of an input line which will be considered a BASIC line of program text: 10 LIST. On examination of the Input Line Buffer, it would reveal the following code if no carriage return were typed after the line of text. Type in the line of code: 10 LIST. Do not enter a carriage return. BREAK the C1P. Call up Monitor Mode by typing M. Call address mode. Call memory location $0013. You will find that the code listed in the next example will reside at memory locations starting at $0013.

```
0013 31 = ASCII 1
0014 30 = ASCII 0
0015 20 = ASCII space
0016 4C = ASCII L
0017 49 = ASCII I
0018 53 = ASCII S
0019 54 = ASCII T
```

On examination of the code in the Input Line Buffer you will find that all the code will be the hexadecimal ASCII equivalent of the text entered at the keyboard.

The code stored in the Input Line Buffer will have a different appearance if you terminate the line with a carriage return. The code will appear in the Input Line Buffer as in the next example.

```
0013 99
0014 00
0015 20
0016 00
0017 49
0018 53
0019 54
```

Try entering the line 10 LIST (CR). Break the computer. Call $0013 and examine the code in the Input Line Buffer. As you can see BASIC has converted its contents.

Now let's try an Immediate Mode operation and examine the Input Line Buffer. First, clear BASIC workspace. Type NEW (CR). Next type LIST (CR). Break the computer and call Monitor Mode. As before, call $0013 and examine the code stored in the buffer. On examination you should find the following code:

```
0013 99
0014 00
0015 53
0016 00
0017 00
```

This data spells out the LIST command. The byte $99 is a Token for the keyword LIST. What is a Token? It is a single byte that represents a command or keyword. OSI BASIC has a Token for all BASIC keywords. Tokens are used by BASIC in immediate Mode or they are stored in all BASIC programs stored in the BASIC program workspace or BASIC source code table. For the sake of this article let's say that a Token describes to BASIC a keyword. A keyword is an indicator to BASIC as to what function BASIC must perform in the case of $99 (LIST) BASIC is told to LIST all the source code in the BASIC workspace.

The point that we have made with the examples indicates that, for BASIC to know what is expected, the proper code must be in the Input Line Buffer starting at $0013. We can use the facts just presented to make BASIC think an operator has entered an Immediate Mode command, but the command can be initiated from a machine language routine as you will see. We are not ready yet to use our new knowledge about the Input Line Buffer and Tokens as commands called from machine language routines. First we must learn some more facts about BASIC.

How does BASIC execute the code for commands in the Input Line Buffer? The code must be read by the BASIC Interpreter. On examination of a Zero page memory map, you will find a machine language routine which starts at $ 00BC. This routine is called a "PARSER." It is used to read a line of code, character by character, stored in the line buffer or code stored in a program line in the BASIC workspace. The Parser routine at $00BC looks at the first character of code in the buffer to see if the character is an ASCII numeral or not. If the first character were a numeral, the Parser tests each character until a non-numeral is found. If the first character is a numeral, the line of code in the buffer is recognized as a line of source code and will be stored in the source code table. When the Parser detects a non-numeral, the Parser routine hands the code to a routine that "Tokenizes" the line before the line is placed in the source code table or back into the input line buffer. If the first character in the buffer is a non-numeral, the parser determines that the input code must be an immediate mode command. If you recall the earlier examples, we demonstrated the keyword LIST entered as a program source line. First we examined the buffer without a carriage return. It was evident that the code was ASCII. Next, we entered a line of text ending with a carriage return and examined the data in the buffer. At this point, we found that the data was in a Tokenized form. As you can see, the BASIC interpreter had, in fact, converted the ASCII to a condensed (or Tokenized) line of code.

To understand how the parser routine interprets the source code (or the code in the Input

Line Buffer) please refer to Listing 1. The machine language parser routine shown in Listing 1 shows that memory locations $00C2, 00C3, and 00C4 contain an LDA direct instruction or AD 13 00. This instruction causes the 6502 accumulator to be loaded with the code at the first address of the Input Line Buffer. On initialization, (BASIC Cold Start) address $00C2, 00C3, and 00C4 will point to $0013 (the beginning of the Input Line Buffer). If you type RUN in Immediate Mode without a program in the BASIC workspace, address $00C2, 00C3, and 00C4 will contain AD 00 03. As you can see, the Parser now points to the beginning of the BASIC workspace.

At this point, enough knowledge about the Input Line Buffer and the parser routine has been presented to allow us to explore the possibility of implementing and executing BASIC Immediate Mode commands called from outside ROM BASIC using machine language routines.

Let us now experiment with the Input Line Buffer and the parser routine to see if we can actually call a BASIC Immediate Mode command from a machine language program. As I mentioned at the beginning of this article, I needed to call BASIC's LOAD and SAVE commands. Let's begin with these. First, let's try the SAVE command to demonstrate how it can be called from a machine language routine.

To use the SAVE command we must learn yet more facts about how BASIC functions. When the user wishes to save a program that is stored in the BASIC workspace, the SAVE command must be used. What happens when you type SAVE? When the command, SAVE, is entered at the keyboard and ended with a carriage return, the code will, of course, be placed in the Input Line Buffer as ASCII. When the carriage return is entered, BASIC examines the code and recognizes that this is an Immediate Mode command. The code in the Input Line Buffer will be Tokenized and placed back in the Input Line Buffer. The Input Line Buffer would not contain:

$0013 94 = TOKEN FOR SAVE
$0014 00 = NULL
$0015 53
$0016 00 = NULL

Now, on examination of the Parser routine at address $00C2, 00C3, and 00C4, you will find that the Parser has read the code located at address $0013 and found a Token for the keyword SAVE, and that BASIC has executed the comand. When the SAVE command was executed, BASIC performed the task of setting what is called the SAVE flag. This flag tells the computer that any data sent from BASIC will be sent to the cassette port and to the screen. The SAVE flag is located at $0205. If the contents of $0205 are set to $00, then output from BASIC will be listed to the screen.

If the SAVE flag contains $01 then the cassette port along with the screen will be activated.

We may use these facts to call the BASIC SAVE command from a machine language routine. Let me demonstrate with an example. Enter the machine language routine (Listing 2) into the computer. Now write a BASIC program into the computer. This program can be any program that you may have on hand, but a single program line will do for the demonstration. Exit BASIC and call the address of the machine language routine of Listing 2. Run the machine language routine. As you can see, the BASIC program that you entered into the computer was LISTed out to the screen of your monitor. Also, the program will be sent to the cassette port.

On examination of the Assembly Listing, notice that we have loaded the Input Line Buffer at $0013 with the Token for LIST ($94). Also notice that, in the Listing, we are setting the SAVE flag at $0205 to the the value of $01. We have set address $00C3 and $00C4 in the Parser routine to point to the beginning of the input line buffer. Finally, we call a routine in the BASIC interpreter located at $A4B5. This routine is called the LIST routine and will execute the LIST command when called by a BASIC program, Immediate Mode, or by a machine language calling routine. As you can see, we have programmed a SAVE and a LIST

Program 1.

```
 10 0000               ;
 20 0000               ;
 25 0000               ;
 30 0000               ;
 40 0000               ;      PARSER CODE
 50 00BC               *=$BC
 60 00BC E6C3          S0 INC $C3   INCREMENT LOW ADDR. BYTE
 70 00BE D002          BNE S1
 80 00C0 E6C4          INC $C4   INCREMENT HIGH ADDR. BYTE
 90 00C2 ADFFFF        S1 LDA $FFFF   LOAD WITH CODE CHARACTER
100 00C5 C93A          CMP #': CHECK FOR COLON (STATEMENT END)
110 00C7 B00A          BCS S2  IF YES BRANCH TO START NEW LINE
120 00C9 C920          CMP #'  ISIT A SPACE
130 00CB F0EF          BEQ S0  IF YES GET NEW CHARACTER
140 00CD 38            SEC     SET CARRY FLAG
150 00CE E930          SBC #$30  SUBTRACT $30
160 00D0 38            SEC     SET CARRY FLAG
170 00D1 E9D0          SBC #$D0  SET C FLAG FOR ASCII NUMBERS
180 00D3 60            S2 RTS  END ROUTINE.  CHARACTER NOW IN A
```

Program 2.

```
 10 0000               ;
 20 0000               ;
 30 0000               ;
 40 0000               ;
 50 0000               ;  BASIC SAVE COMMAND CALL
 60 0000               ;
 70 0000               ;
 80 0000               ;
 90 0000               ;
100 1000               *=$1000
110 1000 A901          START LDA #$01  VALUE SAVE FLAG=ON
120 1002 8D0502        STA $0205    STORE IN SAVE FLAG
130 1005 A999          LDA #$99    TOKEN LIST
140 1007 8513          STA $13    PUT IN LINE BUFFER
150 1009 A900          LDA #$00    NULL
160 100B 8514          STA $14    PUT BUFFER+1
170 100D 8516          STA $16    PUT BUFFER +3
180 100F A953          LDA #$53
190 1011 8515          STA $15    PUT BUFFER+2
200 1013 A914          LDA #$14    PARSER SCAN START LOW BYTE
210 1015 85C3          STA $C3    PUT IN PARSER
230 1017 A900          LDA #$00    PARSER SCAN HIGH BYTE
240 1019 85C4          STA $C4    PUT IN PARSER
250 101B 4CB5A4        JMP $A4B5   GOTO BASIC LIST ROUTINE
```

command into BASIC from outside ROM BASIC and caused its execution.

In a similar manner, let's call and execute a LOAD command from a machine language routine. Enter Listing 3 into the computer. Next bring up BASIC in Warm Start. (Type NEW (CR).) Exit BASIC. Call up the machine language routine for the LOAD command. Place a BASIC program tape into your cassette recorder, execute the machine language routine, and start your recorder on play. Your BASIC program will load into the computer as if called directly under BASIC.

On examination of Listing 3, you will find that the implementation of the LOAD command was very simple. We only need to set the LOAD flag to turn the system on for a BASIC load and jump to the Warm Start of BASIC.

Listing 4 will be used to implement the BASIC RUN command from a machine language program. As before, enter the machine language program into memory and then load a BASIC program into the BASIC workspace. Exit BASIC and call the machine language routine. Start the machine language program. The computer will jump to the BASIC program and run.

On close examination of Listing 4, you will see that we have used the same procedure to force a BASIC RUN command that we used in the SAVE and LOAD routines. We loaded the input line buffer with the Token for RUN, set the Parser scanner to start reading the code in the Input Line Buffer at $0013. With the RUN command it was found that two BASIC interpreter routines were needed to force the computer to execute the RUN command. These were the conversion routine at $A3A6 and the execution routine located at $A5F6.

At the beginning of this article, I said that an executive TOS could be written in machine language that could call BASIC commands. Also, it was mentioned that in order for the TOS to be truly an executive, we must devise some means of exiting BASIC and returning to the TOS. I have shown how BASIC commands could be executed from machine language routines. But, how do we exit BASIC to our machine language routines? At first, it appears that ROM BASIC can only be exited with a BREAK or through a USR function call. This is true unless we can devise some means of patching into BASIC at some point and make BASIC think there is some new form of keyword present in the interpreter. Well, implementing new Keywords is not possible with ROM BASIC, so some other method must be devised.

An article which appeared in *Micro* described interception of BASIC Syntax error codes when printed on the monitor screen. A patch devised to intercept a Syntax error can be utilized to direct an exit from BASIC and force a return to a calling machine language program. The machine language patch routine shown in Listing 5 can be used to force an exit from BASIC during a running BASIC program, and in Immediate Mode or when a BASIC program has finished loading from cassette into the BASIC workspace. Listing 5 is a routine that has been revised for the purpose of exiting BASIC. The routine appeared in an article titled "Stop Those S' Errors" published in the November 1980 issue of Micro Magazine (*Micro*, 30:37).

The patch code for the BASIC exit routine utilizes a vector location in zero page. The vector is located at $03 and $04. Normally, this vector points to the string output routine of the BASIC interpreter at $A8C3. If we replace this jump with a call to our patch routine, we may use the pointer and our patch routine to exit BASIC on command. Listing 5, shows the Exit patch routine that is loaded into memory starting at $0240. To use the patch routine, replace the jump at $03 and $04 with the start of the exit patch routine. That is, load $40 into memory location $03 and $02 into location $04. This can be done in BASIC using the POKE command: POKE 3, 64 : POKE 4, 2. Once the address for the patch code has been loaded into the pointer at $03 and $04 the pointer will not have to be changed unless the computer has been reset.

Program 3.

```
10 0000          ;
20 0000          ;
30 0000          ;
40 0000          ;
50 0000          ; BASIC LOAD COMMAND CALL
60 0000          ;
70 0000          ;
80 0000          ;
90 0000          ;
100 1100                *=$1100
110 1100 A9FF          START LDA #$FF      VALUE LOAD FLAG =ON
120 1102 8D0302        STA $0203        PUT IN LOAD FLAG
130 1105 4C74A2        JMP $A274        GOTO BASIC WARM START
```

Program 4.

```
10 0000          ;
20 0000          ;
30 0000          ;
40 0000          ;
50 0000          ;              ** BASIC RUN COMMAND CALL **
60 0000          ;
70 0000          ;
80 1150          *=$1150
90 1150          ;
100 1150 A952        LDA #$52         GET RUN TOKEN
110 1152 8513        STA $13         PUT IN LINE BUFFER
120 1154 A900        LDA #$00         NULL
130 1156 8514        STA $14         PUT BUFF+1
140 1158 8516        STA $16         PUT BUFF+3
150 115A 85C4        STA $C4         PUT PARSER HIGH BYTE
160 115C A94E        LDA #$4E
170 115E 8515        STA $15         PUT BUFFER+2
190 1160 A913        LDA #$13         GET PARSER START LOW
200 1162 85C3        STA $C3         PUT PARSER LOW
210 1164 20A6A3      JSR $A3A6        GO BASIC CONVERSION RTN.
220 1167 4CF6A5      JMP $A5F6        GO TO BASIC EXECUTION (RUN)
```

Program 5.

```
10 0000               ;
20 0000               ;
30 0000               ;
40 0000               ;
50 0000               ;
60 0000               ;       BASIC EXIT PATCH ROUTINE
70 0000               ;
80 0000               ;
90 00F0                     *=0240
100 00F0              ;
110 00F0 48           PHA          SAVE PRT CHRACTER IN ACC.
120 00F1 AD65D3       LDA $D365        GET CHRACTER FROM SCREEN
130 00F4 C93F         CMP #$3F         TEST FOR ERROR(?)
140 00F6 D008         BNE OUT         NO NOT ERROR GO PRINT CHR.
150 00F8 A900         LDA #$00         YES ERROR GET READY TO EXIT
160 00FA 8D0302       STA $0203        RESET LOAD FLAG
170 00FD 4CFFFF       JMP $FFFF        RETURN TO CALLER($FFFF DUMMY
180 0100 68           OUT PLA          ADDRERESTORE CHARACTER TO ACC
190 0101 4CC3A8       JMP $A8C3        GO PRINT CHR. RETURN TO BASIC
```

The patch routine at $0240 tests memory location $D365 for a question mark ($3F) for each character printed out to the monitor screen. In the event of an error, such as ? Sn Error, the question mark will be loaded into video RAM at $D365. The routine tests $D365 for $3F. If there should be any type of error, the question mark code will appear at $D365. On detection of the error code, the patch routine will cause an exit to your machine language routine. Under normal program execution, the data to be printed is passed to the string printing routine of BASIC as if the patch routine did not exist.

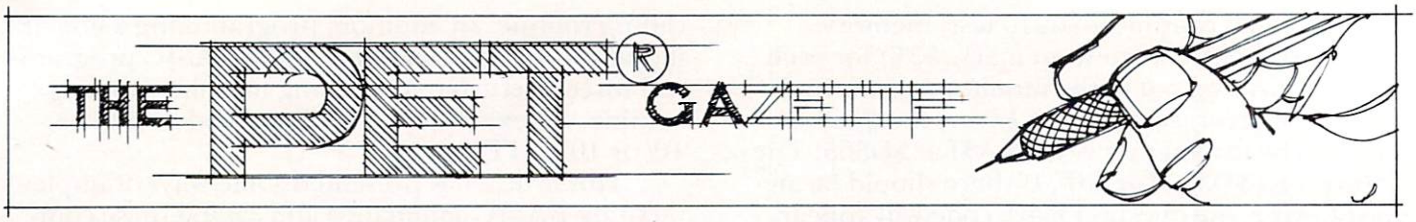
The exit patch code routine was implemented into my TOS to detect an error at the end of a program loading from tape. My Stringy Floppy tape unit sends $8F when all the program on tape has been sent to the C1P. This hex byte, when seen by BASIC, will send back a Syntax error which will be detected by the patch routine causing an automatic exit to the TOS. While in BASIC, if the user types any key followed by a carriage return. It will cause a Syntax error and force a return to any calling routine. In addition, programming a line of illegal code at the exit point of the BASIC program will force a return to the calling machine language routine. An example line of illegal code could be: 10/ or 10 EXIT etc...

This article has presented some ways of implementing BASIC commands and calling these commands from machine language programs. Through these efforts, I have further expanded the ways in which we may use OSI BASIC and machine language programs as a means of system development. In my case, I have a TOS that functions like a disk operating system (DOS). With the information presented in this article, you may also be inspired to develop new programming techniques. Although this article was developed around OSI 6502 BASIC, the concepts should apply to other systems using similar BASIC such as, PET, and APPLE. Of course, tokens and interpreter routine addresses may need changing but the basic principles still apply.

References:

OSI BASIC In ROM, Edward H. Carlson

"Stop Those S' Errors," *Micro* Magazine, November 1980. ©



Practical Pet Printing Primer For Perplexed Programmers

Ron Gunn
Livermore, CA

When you first connect a printer to your PET, it is for one primary purpose: to obtain program listings so you can see enough code at one time to rescue yourself from the latest paradox in your programming. As eminently useful as this is, it soon becomes but a step in the utilization of the printer for the output of data from your programs in an organized and permanent form.

Organizing data on a printed sheet implies just that; organizing. You don't want it scattered randomly and capriciously all over the page. Columns with labels are often desired, and so you find that the convenient TAB functions for the PET screen don't work on the printer. It is then that you perhaps start to reduce your data to strings, add blanks, and then partition these longer strings with the LEFT$, RIGHT$, and MID$ functions to make neatly justified columns.

Universal Printing

We are assuming that you are interested in programming that will allow you to print on any printer that will give a listing, a PET, Epson, or whatever. This kind of coding will be the subject of this article. Many printers, including Commodore's, have proprietary ways of handling this problem and I say more about the PET printer's formatting later. These proprietary methods will normally work only on that one make of printer.

You will be able to take the kind of code we are discussing here over to Joe's house and it will work on his PET, Anadex or Kung-Fu papergobbler.

The Answer That Creates A Problem

Here is a sample of code that outputs 3 variables in neat columns on any printer that will LIST. It is intended to illustrate how the string technique mentioned in paragraph two works in practice. There is a problem with this, especially in PET ROM sets earlier than 4.0, and it is not recommended for large data bases or more than a page of print for reasons we will get into.

```
2200 FOR I=1 TO 30
2210P$=""
2220Q$="        "+STR$(K%(I))
2230P$=RIGHT$(Q$,8)
2240Q$="        "+STR$(L(I))
2250P$=P$+RIGHT$(Q$,8)
2260Q$="        "+STR$(M%(I))
2270P$=P$+RIGHT$(Q$,8)
2280PRINT#4,P$
2290NEXT
```

This code is reasonably concise and straightforward and will produce neat columns of numbers of varying length, all nicely right-justified. What, then, is the problem? It is a lot like watching a centipede on a treadmill. There is an awful lot going on, but not much progress. Concantenating strings and then printing them out is potentially very slow. As you increase the number of columns, the string handling becomes appreciably slower than the printer.

When you add to this the fact that enormous numbers of throwaway strings are created, and that pre-4.0 PETs can take many seconds to collect garbage, you have a serious problem. (See Butterfield, "Learning About Garbage Collection," **COMPUTE!** #10.) If you have not experienced this yet, try the experimental listing at the end of this article for an eye opener.

This code example is print test #1 in the sample listing, and 100 seconds and more were required on my 32N to run it. This is admittedly an extreme example, but points up the trouble you can get into with a program that is large for your machine when you get heavily into string handling.

A Partial Answer

As the referenced article discussed, you can reduce the garbage collection, and you can get the printer to print while the computer is computing, with the following changes. If you have existing print routines that are structured like the above, and which could use some speeding up, then the following substitute coding will help:

```
2300 FOR I=1 TO 30
2310Q$="        "+STR$(K%(I))
2320PRINT#4,RIGHT$(Q$,8);
2330Q$="        "+STR$(L(I))
2340PRINT#4,RIGHT$(Q$,8);
2350Q$="        "+STR$(M%(I))
2360PRINT#4,RIGHT$(Q$,8)
2370NEXT
```

The change, of course, is that an intermediate variable is not created and then concantenated. The variable is created as a string and is packed with blanks to the maximum needed length on the same line. It is then immediately printed using the appropriate string handling function to make it the correct number of spaces to fit the column you are creating.

You have saved in two ways. The forever concantenated P$ is not created; an advantage as pointed out by the redoubtable Mr. Butterfield. In addition, the printer can process each output while the next gets a head start.

This is print test #2 in the sample listing. Run times on my machine ran some 50 seconds, about half the total time required for test #1.

Columns Of Names

These examples have been right justified. If you need to print a column of names, you will want to left justify it. This is not necessarily because it is better, it is just that it will be considered a bug or a product of indifference if you do not. Sample coding to do this is:

```
2800P$=N$+"                    "
2810PRINT#4,LEFT$(P$,20)
```

In this case, the necessary blanks have been added after the name, and the resulting oversized string has been printed out starting with the left-most character using the LEFT$ function. Always add enough blanks to insure that the space is filled even if N$ turns out to be a null string. This insures that other columns will not be disturbed if that happens.

When you have strings to print, you have to print strings. You can avoid extra string creation as shown by tacking on the necessary blanks at once. Then print immediately using the LEFT$ function in the print statement.

Hints And Kinks

If you have a lot of fill blanks to add, it saves memory to create space variables as a substitue for the blanks-within-quotes as shown on line 2220. Create these handy variables when the program initializes:

```
6000S4$="    ":S5$="     ":S6$="      "
6010S7$="       ":S8$="        "
6020RETURN
```

Use a gosub to do this when the program runs. GOSUB6000 in the first line of a program would set up the space variables shown above for use later.

The eight blanks in a line like 2310 could then be added to each string as:

```
2310P$=S8$+STR$(K%(I))
```

Another point; you already know that the TAB function doesn't work on the printer. Well, the SPC function does. You may have wondered what the differences between these two seemingly similar functions are. This is one of those differences. We will see an example of use later.

WHAT #0$*! APPROACH WORKS?

That second word was copied verbatim from the cover of **COMPUTE!** #7. Anyhow, there is a way to get whole numbers into printed columns without having to cope with slow string handling. It may sound silly, but the way to do it is to keep it as data (numerical data that is). We will now examine the techniques that may allow you to avoid string renderings of data altogether. You will then put the data out as fast as the printer can handle it; all day if necessary and always in neat columns.

The Basic Idea

Let's say you want to print out a number that will range between 0 and 999 in a column eight spaces wide. Using the integer variable K%, we might write:

```
2900IFK%=0THENPRINT#4,S8$;:GOTO2950
2910IFK%<10THENPRINT#4,S5$K%;:GOTO2950
2910IFK%<100THENPRINT#4,S4$K%;:GOTO2950
2930 PRINT#4,S3$K%;
2950REM NEXT COLUMN
```

Here we have covered the full range from 0 to 999. If the number goes over 999 it will disturb the columnation unless another line of code is added to catch numbers less than 9999 to realign it. Only one of these print statements will actually execute. If you expect a lot of larger numbers then this routine would go faster with the larger numbers first, as fewer comparisons would take place before the correct line was found and executed. This code will place one column on the sheet. The next column would require a repeat of the whole routine, so coding this way could get long. It is fast, however, and it is effective and straightforward if only a few columns are needed.

In this example, zero is not printed, eight spaces are printed instead. A single digit, including zero if you want it to print, takes three spaces. The <10 or single digit line (line 2910) has 5 spaces placed first, then the sign space, the numeral itself, then a space after. If you want 0 to print, then put in REM at the start of line 2900 to neutralize it. (Don't eliminate it as it is a GOSUB target line).

Note that the carriage return suppress semi-colon is used on all of these lines. This leaves the printer poised on the start of the next column on the same line. A PRINT# statement must be added at the end of the line to go smoothly to the next line, as we will see in the next example. It is like bowling, where the tenth frame is handled differently to wrap things up.

The GOSUB Variable Width Column Maker

Now that the principle has been covered, let's go to some code that will produce an unlimited number of columns of varying width using a reusable subroutine for each column desired. This routine will compete in size with string handling code and is as fast as you'll want.

```
2400FORI=1 TO 30
2410P=K%(I):S=6:GOSUB2460
2420P=L(I):GOSUB2460
2430P=M%(I):S=8:GOSUB2460
2440PRINT#4
2450NEXT
2460IFP=0THENPRINT#4,SPC(S);:GOTO2510:
REM BLANK WHEN P=0
2470IFP<10THENPRINT#4,SPC(S-3)P;:GOTO2510
2480IFP<100THENPRINT#4,SPC(S-4)P;:GOTO2510
2490IFP<1000THENPRINT#4,SPC(S-5)P;:GOTO2510
2500PRINT#4,SPC(S-6)P;
2510RETURN
```

Any practical number of columns can be created by reusing the GOSUB at 2460 over and over. The width of each separate column is determined by the variable S, set at line 2410 for six spaces for K%(I) and L(I), and then set again for eight spaces for the variable M%(I).

When the last column has been printed out, the carriage return suppress is cleared with a final PRINT# statement on line 2440. Any additional columns would be added after line 2430 and before 2440, so the PRINT# statement would still be last.

This is print test #3. It prints in about 16 seconds with no pauses. My 80 cps printer is apparently running at its full speed through the whole test.

Check the data that you are printing to make sure that there is a printing line for each possible number that will be sent to the printing subroutine. Any number that is out of range will skew the columnation. You may have to exclude numbers that are too small or too large, or which contain negative or decimal quantities.

Lines, sometimes a number of them, may have to be carefully added to care for everything that you want to include in your printout. Remember the words of Gerald Weinberg in *The Psychology of Computer Programming*: "To detect errors, the programmer must have a conniving mind, one that delights in uncovering flaws where beauty and perfection were once thought to lie."

The Commodore Answer

I have not seen an in-depth review on the PET 2022 printer, but I have worked with a number of them. Commodore has provided a neat way to format printed information using a format file which you instruct as needed to produce columns, with many useful options. This works only with PET printers and then with reservations, as explained below.

There are two ROM sets for the 2022. The early set is noteworthy for the fact that it supplies a carriage return for every linefeed. When it pages or passes blank lines, the print head moves clear across for each line. That is the bad news. The good news is that the formatting works, and works well.

The newer ROM set will give linefeeds without the time-consuming full carriage scan. There is a bug, however, that causes the machine to go into permanent lower-case mode when more than a few columns are sent to the format file and then printed to. This is a fatal error, as nothing short of a power down will restore normal operation. Complex formatted output that prints perfectly on the older ROM set cannot be made to run on the new set.

I have not been able to locate a corrected ROM, although it has been on the horizon for a year. I have not yet had a chance to try the new PET/MX-80 printer.

Epson

The standard MX-80 was built for use with the TRS-80 and contains the Radio Shack character set to prove it. It does not have the same logic regarding carriage return/linefeed that the PET printer does. With the switch set for listings, KEYPRINT (**COMPUTE!** #7) will not work. Wordpro 3 required an added lf1 (linefeed) in every file header, and so on. Hoo Boy!

There is a switch that adds linefeeds, but it is buried inside the machine. I almost had my cover trained to jump off when I snapped my finger, when I finally added an external SPST switch in parallel with switch 2-3. When that SPST wears out I will have to replace it.

David Lein's late 1980 manual does a good job of explaining and demonstrating printer features. Appendix D on using the unit with other than the TRS-80 is very brief and touches on the Apple only.

The important features like double width letters, variable line spacing, and double strike work from the keyboard and under program control if you leave those pernicious JPET controls on the IEEE interface board OFF.

I hope that these samples and the discussion will help you to avoid some of the long hours otherwise required to find out how to efficiently use your printer. Some of these problems had me almost carrying my head around under my arm a few times. Remember Weinberg's words; "Any fool without the ability to share a laugh on himself will be unable to tolerate programming for long."

The Demonstration Program Listing.

```
1000 M=PEEK(51000):J=140
1450 IFM=0THENL=PEEK(135):POKE135,26:GOTO2000
1500 L=PEEK(53)
1510 POKE53,26
2000 DIMA$(J),B$(J),C$(J),D$(J),K%(30),L(30),M%(30)
2020 FORI=1TOJ
2030 A$(I)="TRY":B$(I)=A$(I)+"THIS":C$(I)="OUT":D$
     (I)=C$(I)+"TO SEE"
2040 NEXT
2050 FORI=1TO30
2060 K%(I)=RND(1)*9
2070 L(I)=INT((K%(I))↑2)
2080 M%(I)=(L(I))↑2
2090 NEXT
2100 OPEN4,4
2110 T1=TI
```

```
2120 PRINT"PRINT TEST NUMBER (1, 2, OR 3)"
2130 INPUTB:ONBGOSUB2200,2300,2400
2135 T=(TI-T1)/60
2140 PRINT#4,T
2180 CLOSE4
2190 PRINT"THAT TOOK"T"SECONDS TO PRINT"
2192 PRINTFRE(0)
2193 IFM=0THENPOKE135,L:END
2195 POKE53,L:END
2200 FORI=1TO30
2210 P$=""
2220 Q$="          "+STR$(K%(I))
2230 P$=RIGHT$(Q$,8)
2240 Q$="          "+STR$(L(I))
2250 P$=P$+RIGHT$(Q$,8)
2260 Q$="          "+STR$(M%(I))
2270 P$=P$+RIGHT$(Q$,8)
2280 PRINT#4,P$
2290 NEXT:RETURN
2300 FORI=1TO30
2310 Q$="          "+STR$(K%(I))
2320 PRINT#4,RIGHT$(Q$,8);
2330 Q$="          "+STR$(L(I))
2340 PRINT#4,RIGHT$(Q$,8);
2350 Q$="          "+STR$(M%(I))
2360 PRINT#4,RIGHT$(Q$,8)
2370 NEXT:RETURN
2400 FORI=1TO30                                        13
2410 P=K%(I):S=6:GOSUB2460
2420 P=L(I):GOSUB2460
2430 P=M%(I):S=8:GOSUB2460
2440 PRINT#4
2450 NEXT:RETURN
2460 IFP=0THENPRINT#4,SPC(S);:GOTO2510:REM THIS
     LINE LEAVES BLANK SPACE ON 0
2470 IFP<10THENPRINT#4,SPC(S-3)P;:GOTO2510
2480 IFP<100THENPRINT#4,SPC(S-4)P;:GOTO2510
2490 IFP<1000THENPRINT#4,SPC(S-5)P;:GOTO2510
2500 PRINT#4,SPC(S-6)P;
2510 RETURN
READY.
```

©

Odds And Ends:

A Fat Forty Bug

Gordon Campbell, Willowdale, Ont.

Some machine language programs which work just fine in PETs with BASIC 4.0 and 9-inch screens will yield odd results on the new 12-inch, 40-column machines. Occasionally, it will appear as if the program has responded inaccurately to the key which was pressed.

This is due to a very subtle difference in the ROM. If the machine language program happens to be in decimal mode when an interrupt occurs, the keyboard decode is completely inaccurate. Even if a key is still pressed from a previous character, it will be decoded as a different character, so, instead of getting one character, you get three (the original correct one, one that is wrong, and probably the original one again).

The problem is well illustrated by the program listing attached. If you assemble the program as shown, then from the monitor type: G 0800 you can type on the keyboard and see what character the PET thought was pressed. By the way, to get out of the program, press the RVS key. Now change the SED (set decimal mode) to a NOP. The program gives accurate keyboard decode. Now the STOP key will get you out.

The way to get around this bug, if you must use decimal mode, is to precede the routine with an SEI (disable interrupts) instruction, and follow it with a CLI (enable interrupts).

```
                                   .BA $0800
                                   .OS
                  ; TEST KEYBOARD DECODE
                  ; WITH DECIMAL MODE SET!
                  GET              .DE $FFE4
                  PRINT            .DE $FFD2
                  ;
0800- F8          STRT             SED
                  ;
0801- CA          DLLOOP           DEX
0802- D0 FD                        BNE DLLOOP
0804- 88                           DEY
0805- D0 FA                        BNE DLLOOP
                  ;
0807- D8                           CLD
0808- 20 E4 FF                     JSR GET
080B- C9 00                        CMP #0
080D- F0 F1                        BEQ STRT
080F- C9 03                        CMP #3
0811- F0 06                        BEQ STOP
0813- 20 D2 FF                     JSR PRINT
0816- 4C 00 08                     JMP STRT
0819- 00          STOP             BRK
                                   .EN

LABEL FILE:  [ / = EXTERNAL ]

/GET=FFE4            /PRINT=FFD2     STRT=0800
DLLOOP=0801          STOP=0819
//0000,081A,081A
```

©

Machine Language: What's Your Sign?

Jim Butterfield
Toronto, Canada

Beginning programmers learn very quickly that a memory location has eight bits capacity, so that it may hold a number from zero to 255, or 00 to FF in hexadecimal. That's the range of values that you are allowed to POKE and that you will see with a PEEK. These numbers all seem to be positive. Why, then, do some of them set the N (negative) flag when loaded? More generally — how do you handle signed numbers?

As Humpty Dumpty almost said: "When I use a number, it means just what I choose it to mean — neither more nor less". As programmers, we can choose to treat a memory location value as if it were unsigned, a number from 0 to 255; or signed, a number from -128 to +127. We can view the value in many other ways, too: as part of a bigger number, as an ASCII character, as two or more small numbers packed together, and so on. For the moment, let's concentrate on the signed number aspect.

Sign Posts

The N (negative) flag is tied to the most significant bit of the value under consideration. For example, if we load into a register a value of hexadecimal C8 which would PEEK as 200 decimal, the N flag is turned on. Why? If we write C8 in binary we get 11001000. The first bit (called the high-order bit or the most significant bit) is one, and that's what kicks the N flag on. If we mean the number to be unsigned, we may ignore the N flag; but if we mean the number to be signed, the N flag tells us that it is negative. We can tell what negative value is represented in an 8-bit location by subtracting its unsigned value from 256, so that C8 (unsigned 200) has a signed value of -(256-200) or -56.

This method of representing signed numbers is called twos-complement and it works well once you get used to it. There are a few special rules to keep in mind when you add, subtract, multiply, and compare, but most things are quite straightforward. You'll quickly learn that FF or decimal 255 has a signed value of -1; that the highest 8 bit signed value is hex 7F or +127 and the lowest is hex 80, or -128.

Overflow: My Byte Runneth Over...

When we add unsigned numbers, we need to watch for a leftover Carry (C flag) after the addition is complete. If the C flag is on, it means that the addition has generated a result that is too large to fit the space available. Similarly, when we subtract unsigned numbers, we look for the inverse: the C flag being off means that we have tried to subtract a bigger value from a smaller one — and that's illegal if we want unsigned results.

The rules are different when we add and subtract signed numbers. The problem we must look for here is a "sign switch": for example, adding +100 to +100, two positive numbers, will generate a value of 200 — which is a negative value if placed in a single byte. This type of error is called "overflow," and the 6502 conveniently provides us with an overflow indication (the V flag) to warn us of difficulty in signed addition or subtraction. A BVS (Branch Overflow Set) will detect the fault and allow us to code an appropriate error or warning routine.

Remember that both Carry and Overflow are set with each Add and Subtract command you execute. It's up to you to choose which flags are important: you know which numbers are signed and which are not.

Multiplication: Sign Of The Times?

General multiplication of signed numbers calls for careful testing of both signs and quite a bit of work. For the moment, we'll concentrate on simple multiplication routines: multiplying by a fixed value of say four or five.

We multiply a number by two by using an ASL command. If we were doubling an unsigned number, we once again test the C flag to make sure that the new value fits into the space provided. For signed numbers, it's a little more work: we must make sure that doubling the number hasn't caused the sign to change. The overflow flag won't help us here (I wish it did) since it is unaffected by Rotates and Shifts. The usual coding method is to check that the C flag, which holds the previous sign, matches the N flag, which holds the current sign.

To multiply by four, we use two ASL commands, and we must carefully check for errors after each one. If we wish to multiply by five, we multiply by four and then add the original value — hopefully stored somewhere — and make the final overflow check on the addition.

Comparison: Getting the High Sign...

Comparing signed numbers can't be done with a single flag or a single test. The C flag gives you a valid comparison if the two signs are the same, but not if they differ. You could pre-check the signs: for example LDA VALUE1 : EOR VALUE2 would

mighty wings

UPGRADE-SYSTEM 3064 4064 8064

UPGRADE SYSTEM

(UPSYS) is an intelligent RAM-Expansion for the COMMODORE COMPUTER. With UPSYS you can add 256K-, 128K- or 64K-Byte RAM to your CBM-Computer.
UPSYS is available for the Commodore Computers with BASIC 2.0 and for BASIC 4.0 i.e. 4032N, 4032B or 8032.

ORGANIZATION

UPSYS is organized in banks, 32K RAM, each, addressed from 0000 to 7FFF (0–32767). The OPERATING-SYSTEM (OS) is located in E900-EFFF and is installed in EPROM inside the UPGRADE SYSTEM.
With these locations it will not conflict with any other EPROM (VISICALC, WORDPRO, CREATE-A-BASE...)

WHAT can I do with UPGRADE SYSTEM?

You can use the different banks of UPSYS for programs (Basic or Machine Code) and/or for datafiles. The OS (Operating System) of UPSYS will allow bankswitch (jump from one 32K bank to another 32K bank), by a simple SYS-command. SYS60419 #1 means to jump from one bank into bank# 2.
You can switch to another bank with or without sharing the variables by OVERLAY. This allows a fast calculation from programs needing more than 32K of RAM area without reloading from floppy (and destroying the first program in the memory), for you are able to keep two, three or four, depending on the type of UPSYS you use, in the different 32K banks.

DATAFILES with UPGRADE SYSTEM

You can also set up one or more of the 32K banks for datafile banks.
The OS from UPSYS will allow up to 6 different datafiles in each 32K bank. A total of up to 31 different datafiles in various banks will be allowed. Maximum string length is 255 bytes. It is marvellous to access from a Basic program to any datafile bank containing, for example, 30K bytes of data in direct access with RAM-SPEED without lowering the RAM area in the user's Basic program.

Syntax of read-access:

SYSrd# fn;sn,r$

That means: read in datafile # *fn* the string with the # *sn* into the variable *r$*.

Another advantage of UPGRADE SYSTEM is that there is no restriction in the number of *open-files* at the same time.
You can access up to 31 various datafiles without opening or closing them!!!
Included in the OS of UPSYS is a command allowing you to find, with lightning speed, a specified string matching with a preset string.
This command and all other OS-commands are written in the machine-code of the 6502.
COMMAND SYNTAX:

SYS fi#fn,sp,r$,sn

this will look for a match with r$ in the specified datafile # *fn* (for example, in a customer list or in inventory list) beginning in the *sp*-string position (i.e. mid$ (x$,sp, len(r$)), starting with the *sn*-th string in the datafile. If a match occurs, the position number of the match string will be placed in *sn*.
The advantage of this command is its high speed matching string in every given string-position, without presorting the strings. With this fast access you do not have to keep a lot of different indices of your datafiles to find a specified string. You are able to define your search- or match-field with complete freedom!!!
All these functions will not lower your program space in memory, for the data-files are in another bank in the UPSYS.

Programs with UPSYS

Another advantage of UPSYS is to keep different programs in different banks (you could use a few PETs with various programs). Of course, all of these programs are able to access to one or more of the defined datafiles.
To switch into another program-bank only needs a simple SYS command.
SYNTAX:

SYS jp#bn

That means: SYS 60419 (jp-for jump is set in your program to 60419) into bank # *bn*. With UPSYS 8256 you will have up to 8 additional programbanks plus one bank from your Commodore Computer itself. With UP 8064 you will have up to two additional 32K banks plus one bank from the PET itself.
The command above will switch into the specified bank without sharing the variables. If you want to switch in another bank with the OVERLAY of the variables you use:

SYS jp#bn,val

with this command the variables from the first program are *overlayed* into the bank # *bn*.

If you switch out of a program, the program will remain in the statement following the SYS command, as if "frozen."

If you switch back, your program will "wake up" and continue with the statement following the switch command.
Of course: you are able to switch back with or without *overlaying* the variables.

UPGRADE SYSTEM comes complete in a separate cabinet with all necessary cables, connectors, OPERATING SYSTEM in EPROM (inside the UPSYS cabinet). With the installation and operation manual, UPGRADE SYSTEM is easy to install.

PRICING:

UPGRADE SYSTEM with 64K	usd 792.00
UPGRADE SYSTEM with 128K	usd 1184.00
UPGRADE SYSTEM with 192K	usd 1390.00
UPGRADE SYSTEM with 256K	usd 1576.00
Manual separate	usd 10.00

Please specify which Commodore Computer (BASIC 2 or BASIC 4) you have. Delivery is from stock.

DEALER INQUIRIES are invited from all countries.

UPGRADE SYSTEMs are available from spima computer in WEST GERMANY. For more information, contact:

Bernd H. Ruf
201 Hackberry 1213
CLUTE, TX 77531
(713) 265-5625

Contract Trade Ltd.
788-790 Finchley Rd.
LONDON NW 11 7UR
ENGLAND

spima COMPUTER

Turbinenstr. 4
6800 Mannheim 31
West Germany
Tel. ℘ (06 21) 72 15 15
Telex 4 63 708 spima d

turn the N flag off if the signs were the same, or on if they were different. If the signs were identical, a normal Compare would settle the matter; if not, the positive number must be the biggest one.

Frank Covitz offers the following elegant signed compare:

```
TEST1:  SEC
        LDA  N1      ;Get 1st number
        SBC  N2      ;Compare vs. 2nd number
        BVC  TEST2   ;Branch on overflow clear
        BIT  N1      ;Else test sign of 1st number
TEST2:  BPL  GTE     ;If plus, branch to Greater Than
                        or Equal
        ..           (code for less than)
        ..
GTE:    ..           (code for greater than or equal)
```

Note that two tests are performed: the SBC (which is used to set flags rather than calculate results) and the BIT test. The N flag is brought into play here — it's unusual to see it doing a useful job in a comparison situation.

Odd Signs

When signs are used as part of large numbers, the sign bit appears only in the most significant byte. So if you allocate 32 bits (four bytes) to a value, only one — the highest order — gives the sign.

It's possible to sign decimal numbers. If the numbers are held in BCD for decimal addition and subtraction, the sign works out rather oddly. The high bit is still the most convenient to use — but this causes positive values to be those starting with the digits 0 through 7, and negative values to begin with digits 8 and 9 only. This "unbalanced" arrangement of numbers is often satisfactory and allows the N and V flags to perform their proper roles. If, on the other hand, you need to balance the range of positive and negative decimal values, you'll want positive numbers to start with digits 0 through 4 and negative values with 5 through 9. In this case, you have to do most of your own sign work. As a last resort, you can keep the sign as a completely separate flag — but beware of additions and subtractions that cross the positive-negative boundary.

Signing Off...

Most machine language work is in unsigned integers: you'll need to deal with signed numbers only rarely. But when you do, it's essential to know how to handle them ... you might say that it's one of the signs of good programming. ©

Train Your Pet To Run VIC Programs

Lyle Jordan
Maple Grove, MN

Have you already wished for the capability to renumber your VIC program, or to get a print-out, or to save it on a disk?

This could be especially frustrating if you have a PET, a printer, and a disk sitting idly nearby.

Or how about satisfying the desire to "upload" your VIC program into a PET? This article will give you a couple of quick and easy ways to do just that.

The PET BASIC programs start to occupy memory at location 1025 decimal or $0401 hex. For the VIC, programs will start at 4097 decimal or $1001 hex. To make things compatible, start by putting a one line program into your PET (example: 1 REM). Now load your VIC program by typing "load". The VIC program will load just fine, but will have a starting location of $1001 hex and if you do a LIST, it won't show up at all. You will see only your one line program, 1 REM.

To get to the VIC program, you will need to change the forward linking pointers. This can be accomplished by doing a SYS 54386 (to get into the machine language monitor) and then by changing two memory locations.

First look at memory locations $0400 to $0407, by typing:

```
M 0400,0407
```

The PET will display the following:

```
.M 0400,0407
.: 0400 00 07 04 01 00 8F 00 00
```

Next list memory $1000 to $1007 and see:

```
.M 1000,1007
.: 1000 AA 18 10 0A 00 99 22 56
```

This display will vary depending on the first line of your VIC program. My first line was 10 PRINT "VIC-20".

Now you can change the '07' and the '04' at locations $0401 and $0402. You want this to point to the location of the first forward pointer of the VIC program, so the '07' becomes '01' and the '04' is changed to '10'. Make the changes, press RETURN, and cursor down to the last line displayed, type 'x', and then press return again. When the PET gives the "READY", you are back in BASIC and can do a LIST. What appears is the one line, 1 REM, followed by the VIC program.

Having served its purpose, line one can now be removed, and the VIC program will be copied into the normal start of PET BASIC at location 1025 decimal or $0401 hex.

If you have The BASIC Programmer's Toolkit from Palo Alto IC's, this entire procedure can be replaced by simply activating the Toolkit, and typing "APPEND".

I will have a lot of use for both of these procedures. Some that come to mind immediately are such things as getting a VIC program listing on a PET printer, renumbering a VIC program, and compacting a program so as to make the best possible use of the VIC's 3.5K of memory. I hope that this simple procedure will prove useful to others. ©

Converting To Fat-40

Joe Ferrari
Commodore Canada

The addition of some new features to the 40 column PET has brought about some problems with program compatibility between the 4032 ten inch and 4032 twelve inch CRT display machines. In some cases, the program changes required for proper operation on the "Fat Forty" may be trivial and in other cases the conversion may be next to impossible. I will attempt to cover areas where failure can occur and what changes need implementing.

Level 1
Programs Loading Below BASIC ($0400)

Standard BASIC programs should work without any modification, unless they employ PEEKs and POKEs or if the program loads into memory below BASIC. The latter problem can be a bit tricky to spot unless you know specifically what to look for. If the program does load below BASIC (say $033A), but does not use locations $03E9-03f9, one method that will correct the problem is:

1) get the program into memory (don't execute)
2) SYS to the monitor (SYS 4)
3) display hex $03E9-03F9
4) modify these registers with the following values

[$03E9: 10,10,09,10,00,00,00,00,00,00,00,00,00,00,00,00,00]

5) resave the program via the monitor

Tape Unit #2

Another area where the standard BASIC program can fail is in the utilization of the second cassette unit for sequential file access. If any program calls files from tape unit #2, unpredictable effects can result depending on the data coming into the buffer. In this case, nothing can be done to resolve the problem. The 12 inch 4032 has rendered the operation of the second tape unit virtually useless.

PEEKs And POKEs

Decimal location 151, often used to check if a particular key has been pressed, is still the same on the 12 inch, but the value of the keys has changed and therefore expected values for certain keys will return false information. The following table will assist in the conversion of a program with this problem.

KEY	OLDV	NEWV	KEY	OLDV	NEWV
@	15	64	A	48	65
C	31	67	D	47	68
F	39	70	G	46	71
I	53	73	J	45	74
L	44	76	M	29	77
O	60	79	P	52	80
R	55	82	S	40	83
U	61	85	V	23	86
X	24	88	Y	54	89
1	26	49	2	18	50
4	42	52	5	34	53
7	58	55	8	50	56
0	10	48			

KEY	OLDV	NEWV
B	30	66
E	63	69
H	38	72
K	37	75
N	22	78
Q	64	81
T	62	84
W	56	87
Z	32	90
3	25	51
6	41	54
9	57	57

When POKEs to this problem area are used for saving byte variables (whatever purpose), they must be moved to a free spot elsewhere in memory. If a space is free just below $03E9, this could be a good area for relocating the byte variables.

Level 2
BASIC Programs With Machine Language Utilities

BASIC programs using machine language utilities that reside in the second cassette buffer can work properly provided they don't use the taboo area of the buffer (namely decimal 1001-1017). Again, if the utility uses this area, the space must be relinquished to the PET operating system in order to obtain successful execution of the program. Usually, in the case of small machine language utilities, it shouldn't be too difficult to understand and relocate it to an area of memory that is free.

Level 3
Machine Language Programs

This will be the most difficult area to troubleshoot. If you are going to attempt to modify this type of program, be prepared to spend a good deal of time. Making the necessary changes to get the program working will most likely require a considerable amount of effort — which I personally don't recommend.

If you are really desperate, here are a few helpful hints that may assist you:

CBM/PET INTERFACES

RS-232 SERIAL PRINTER INTERFACE – addressable – baud rates to 9600 – switch selectable upper/lower, lower/upper case – works with WORDPRO, BASIC and other software – includes case and power supply.
MODEL – ADA1450 149.00

CENTRONICS/NEC PARALLEL INTERFACE - addressable – high speed – switch selectable upper/lower, lower/upper case – works with WORDPRO, BASIC and other software – has Centronics 36 pin ribbon connector at end of cable.
MODEL – ADA1600 129.00

CENTRONICS 730/737 PARALLEL INTERFACE – as above but with Centronics card edge connector at end of cable.
MODEL – ADA730 129.00

COMMUNICATIONS INTERFACE WITH SERIAL AND PARALLEL PORTS – addressable – software driven – true ASCII conversion – selectable reversal of upper-lower case – baud rates to 9600 – half or full duplex – X-ON, X-OFF – selectable carriage return delay – 32 character buffer – centronics compatible — much more.
MODEL – SADI 295.00

ANALOG TO DIGITAL CONVERTER – 16 channels – 0 to 5.12 volt input voltage range – resolution is 20 millivolts per count – conversion time is less than 100 microseconds per channel.
MODEL – PETSET1 295.00

REMOTE CONTROLLER WITH CLOCK/CALENDAR – controls up to 256 devices using the BSR X10 remote control receivers – 8 digital inputs, TTL levels or switch closure — 8 digital outputs, TTL levels.
MODEL – PETSET2 295.00

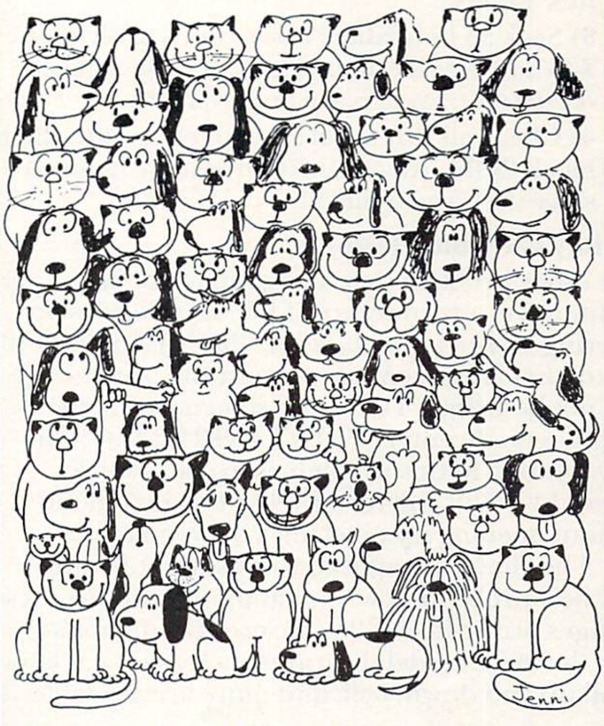
All prices are in US dollars for 120VAC.
Prices on 220 VAC slightly higher.
Allow $5.00 shipping & handling, foreign orders add 10% for AIR postage.
Connecticut residents add 7½% sales tax.
All prices and specifications subject to change without notice.
Our 30 day money back trial period applies.
MASTER CHARGE/VISA accepted.
MENTION THIS MAGAZINE WITH YOUR ORDER AND DEDUCT 5% FROM TOTAL.

IN CANADA order from: Batteries Included, Ltd., 71 McCaul Street, F6 Toronto, Canada M5T2X1, (416)596-1405.
IN THE USA order from your local dealer or direct: Connecticut microComputer, Inc., 34 Del Mar Drive, Brookfield, CT 06804, (203)775-4595.

Dealer inquiries invited.

Connecticut microComputer, Inc.
34 Del Mar Drive, Brookfield, CT 06804
203 775-4595 TWX: 710 456-0052

Every PET Needs a Friend.



CURSOR is the best friend your Commodore PET will ever have. Since July, 1978 we have published 150 of the most user-friendly programs for the PET available anywhere. When we write or edit a program, we spend lots of time fussing about how it will treat you. We pay attention to lots of little things that help make using a computer a pleasure instead of a pain.

Naturally, **CURSOR** programs are technically excellent. Each program that we purchase is extensively edited or re-written by a professional programmer. But *imagination* is just as important as being user-friendly and technically good! We delight in bringing you off-beat, unusual programs that "show off" the abilities of your PET or CBM.

CURSOR is user-friendly, technically great and full of imaginative programs. And every issue of **CURSOR** is still available! We continue to upgrade previously published programs so that they'll work on the three varieties of Commodore ROM's (Old, New, and 4.0). New issues also work on the 80 column CBM.

For only $4.95 you can buy a sample issue and judge for yourself. Or send $18 for a four-issue subscription. Each **CURSOR** comes to you as a C-30 cassette with five programs and a graphic Front Cover, ready to LOAD and RUN on your PET.

Who knows? After your PET meets **CURSOR**, things may never be the same!

AUTHORIZED DISTRIBUTORS:

Great Britain
AUDIOGENIC, Ltd.
P.O. Box 88
Reading, Berkshire

Holland
COPYTRONICS
Bergemeester
Van Suchtelenstraat 46
7413 XP Deventer

Japan
SYSTEMS FORMULATE CORP.
Shin-Makicho Bldg. 1-8-17
Yaesu, Chuo-ku, Tokyo 103

Australasia
MICROCOMPUTER HOUSE, LTD.
133 Regent Street
Chippendale, Sydney
N.S.W. Australia 2008

THE CODE WORKS
Box 550
Goleta, CA 93116
805-683-1585

1) Use Supermon or Extramon to locate any absolute occurrences of memory addresses from \$03E9-\$03f9 and reassign new values.

2) Check hi-low tables for references to the same address locations and, if any, reassign new values.

3) Seek all immediate operations involving hex \$03 and \$e9-f9...if any, look at code where occurrence takes place and evaluate.

4) Check all JSR & JMP occurrences into the E block ROM. All other ROMs can be ignored since they are identical.

Factory CRT Setup (Size)

One other problem that may or may not be encountered is screen setup. If the user decides on entering screen text mode via "PRINT CHR$(14)", the top line of screen may run off the upper edge and not be visible. To restore screen to normal graphics mode enter "PRINT CHR$(142)". One easy solution to this problem is to use "POKE 59468,14". This will turn the PET to text mode without opening up pixel lines between text.

The changes required to existing software may be a problem now, but, at the same time, these changes bring the 4032 to a closer compatibility with the 8032 model. Features such as repeat keys, scroll up and down, bell, and more are available. ©

High Resolution Bar Graphs For The Pet

David C. Swaim
Atlanta, GA

One of the reasons I chose the PET when I bought my computer was its excellent graphics capabilities. The PET/CBM graphics character set can be used to make "double density" or "high resolution" graphs without resorting to additional hardware. The highest resolution can be achieved when using bar graphs. Bar graphs can be found regularly in the financial section of the newspaper and are used to illustrate everything from the yearly rise in prices to the yearly production of wheat. You can draw bar graphs on your PET easily. What I want to describe in this article is a method I use.

How It Was Done

The first program I wrote to draw a bar graph used the reversed space to draw the bar. These bars looked fine but there were only about 20 steps vertically or 35 horizontally (allowing some space for labels). I wanted higher resolution than that for my graphs. It is possible to increase the resolution by a factor of eight either horizontally or vertically. All I had to do was use the other characters needed along with my reversed spaces. I put the needed graphics characters in an array. First I needed to plot the proper number of whole spaces for the bar. This was done by dividing the value to be plotted by the value I chose to be the maximum allowable value (the value that would give me a complete solid bar). This gave me a fraction that I multiplied by 20 (my vertical bar is 20 lines tall) and the result was truncated to an integer value, i.e., the fractional part was dropped (in my "low resolution" program the result was rounded to the nearest integer). This is an important difference. This integer corresponded to the number of times I would print a reversed space. The graphics character to be used for the last space was found by taking the fractional part from the original calculation of length and multiplying it by eight. This number was rounded to the nearest integer value and used as the index of my character array. Thus the proper character was printed to finish the bar.

The Program

The program listed here is a general purpose bar graphing program illustrating the high resolution bars. The bar graph is loaded into an array first before it is plotted on the screen. This is done for flexibility. To switch from vertical to horizontal graphs requires rewriting only the print routine (lines 1000 and up) and putting the appropriate graphics characters in the array X$ (lines 110 and 115). The rest of the program remains the same. You can change the data plotted by retyping the DATA statement (line 150) and changing the heading in K$ (line 160). You do not need to worry about the maximum value of the data. The program calculates (on line 280) a maximum bar length so that the largest data value is plotted nine-tenths of the maximum allowable bar length. If you prefer line 280 can be changed to set a definite ceiling. Be careful though. The program is not protected against data larger than the maximum bar length.

Figure 1 shows the bar graph displayed on the screen by the program. This particular graph shows the seasonal or monthly variation of my natural gas bills last year. During September and October my securing deposit was refunded as a credit on my bills so I paid no bills for those months. I put a dollar in September and a dollar and a half in October to illustrate the high resolution available with this program. This is a resolution of about half of a percent (maximum bar length would represent around $100). Figure 2 illustrates the same data printed with a horizontal format. Try it and I'm sure you'll never be satisfied with "low resolution" graphs again.

```
10 REM *************************
60 REM *************************
100 DIM M$(11,20),Y(12),X$(8)
105 REM X$ CONTAINS GRAPHICS CHARACTERS
106 REM FOR MAKING THE BAR
110 X$(1)="$":X$(2)="o":X$(3)="y":X$(
    4)="b":X$(5)="{REV}x{OFF}":X$(6)=
    "{REV}w{OFF}"
115 X$(7)="{REV}c{OFF}":X$(8)="{REV}{OFF}"
119 REM MOVE CURSOR DOWN CHARACTERS
120 P$="{24 DOWN}"
125 REM HOME AND UP, BACK CHARACTERS
130 H$="{HOME}":U$="{UP}{LEFT}"
140 REM DATA TO BE PLOTTED ON THE BAR
150 DATA 75.36,91.53,61.29,39.56,21.7
    8,11.4,11.39,10,1,1.5,24.69,35.67
155 REM K$ IS THE TITLE OF THE GRAPH
160 K$="NATURAL GAS BILLS"
190 REM READ THE VALUES TO BE PLOTTED
200 FOR X=1 TO 12
210 READ Y(X)
220 NEXT X
230 Y(0)=Y(1)
250 FOR X=2 TO 12
260 IF Y(X)>Y(0) THEN Y(0)=Y(X)
270 NEXT X
275 REM SET MAXIMUM BAR LENGTH, Y(0),
276 REM TO BE 10/9 TIMES LARGEST VALUE
280 Y(0)=10*Y(0)/9
300 FOR M=1 TO 12
301 N=M-1
305 REM CALCULATE THE NUMBER OF PRINT
306 REM LINES LONG THE BAR  WILL BE
310 Z=Y(M)*20/Y(0)
315 REM Z% IS NUMBER OF WHOLE LINES
316 REM TALL THEN BAR IS
320 Z%=INT(Z)
325 REM R% IS NUMBER OF RASTER LINES
326 REM TO BE PRINTED ON THE Z%+1
```

```
330 R%=INT((Z-Z%)*8)
336 IF Z%=0 THEN 530
500 FOR P=1 TO Z%
510 M$(N,P)=X$(8)
520 NEXT P
530 IF R%=0 THEN 550
540 M$(N,Z%+1)=X$(R%)
550 NEXT M
1000 REM PRINT THE BAR GRAPH
1004 PRINT"{CLEAR}"
1005 REM WRITE THE GRAPH TITLE
1006 PRINT SPC(20-LEN(K$)/2);K$
1009 REM PUT BAR IDENT. AT BOTTOME OF SCREEN
1010 PRINT "{HOME}";P$;"   J  F  M  A~
    M  J  J  A  S  O  N  D";
1020 FOR M=1 TO 12
1021 N=M-1
1025 REM POSITION CURSOR READY TO DRAW
1026 REM THE NEXT BAR
1030 PRINT H$;LEFT$(P$,21);SPC(3*M);
1035 REM DRAW THE ENTIRE BAR INCLUDING
     BLANKS
1040 FOR B=1 TO 20
1050 PRINT M$(N,B);U$;
1060 NEXT B
1070 NEXT M
1080 GETA$:IFA$=""THEN1080
1100 END
READY.
```

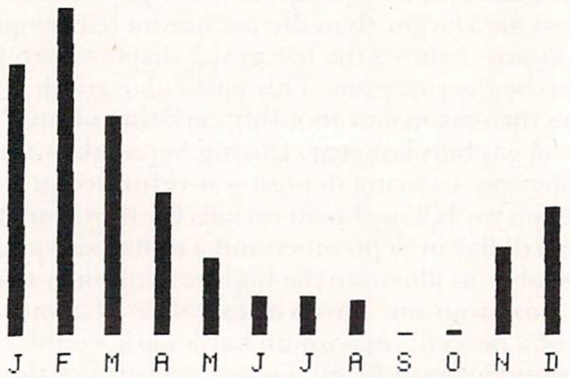


Figure 1: Natural gas bills are plotted to illustrate vertical bars. Note the fine resolution between September and October.

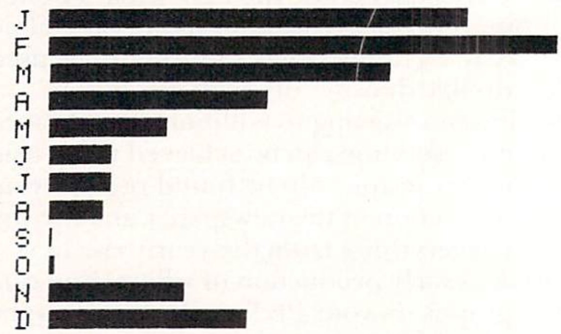


Figure 2: The same data as in Figure 1 is plotted horizontally. The only program changes made were to lines 110 and 115 and the print routine beginning on line 1000. ©

Waking Up The PET Screen

Hal Bredbenner
Raleigh, NC

An active screen display is always an asset to any business or recreational program and there are many ways to animate or otherwise liven-up a drab display of information. Most active screens make use of special graphic characters, machine code routines, or special hardware functions such as a character blink mode generated by a video processor. This article describes a software routine that provides reverse field blinking anywhere on the screen at variable blink rates. The routine uses BASIC only and no special hardware is required. The PET jiffy counter, TI, is referred to by the program, however its contents are not modified. The TI jiffy counter is standard in the PET (all versions, all ROMs) and if another system is used, any Real Time Clock (RTC) can be used as a reference.

The Blinking GET Routine, shown in Listing 1, is useful for prompting the system operator during program execution. When called, the subroutine will display, at the bottom center of the screen, the words "HIT ANY KEY." The displayed words will then blink from normal to reverse video at a rate programmed in line 160. Each jiffy in the PET is 1/60th of a second and since line 160 waits for 15 jiffies the display will blink every quarter of a second. If you don't yet understand how this timing works, I will decribe it further in the next paragraph. The subroutine continues blinking the words until a key is depressed, at which time the routine returns to the main calling program with string variable S$ equal to the input key. The routine is very straightforward in its operation and, although it is not very classical in its construction, it is efficient and can be easily understood and adapted for use in other programs.

This paragraph gives a line-by-line description of the actual operation of the routine and, if you have advanced in your BASIC programming skills, perhaps you will find it a little boring. When you are just getting started though, a line-by-line description can really be a help. Line 120 places the cursor at the bottom of the screen, however this could be modified with the cursor movement characters to place the cursor at any screen location. X$, in lines 130 and 140, is used as a switch to alternately hold the REVERSE FIELD ON or REVERSE FIELD OFF character. Initially X$ holds the REVERSE FIELD ON character, line 130, and a jump to line 150 is made. After the cursor is moved over thirteen spaces by the TAB function, the reverse field is either turned on or off by printing X$. Since X$ initially holds the REVERSE FIELD ON character the reverse field function is turned on and then the words "HIT ANY KEY" are printed. A cursor up character is also printed to leave the cursor on the line just printed. The last task done in line 150 is to set variable H equal to the current value of TI, the jiffy counter. In line 160 a comparison is made to see if 15 jiffies have elapsed since the words were printed on the screen. This is done by comparing the number of jiffies in TI at the time the words were printed, which we called H, to the present value of TI. Any time a program refers to TI, the current value is returned and this is what makes a realtime clock a very nice feature. If the time elapsed is less than 15 jiffies then line 170 checks to see if a key is depressed. If any key is pressed its value is assigned to the string variable S$ and a return to the calling program is made. If no key is pressed line 180 redirects the routine back to line 160. This loop continually checks the elapsed time and the state of the keyboard. After 15 jiffies have elapsed line 160 will redirect the program to line 190. If X$ is equal to the REVERSE FIELD ON character a branch is made to line 140 that changes X$ to the REVERSE FIELD OFF character. If X$ was equal to the REVERSE FIELD OFF character then it alternates to the REVERSE FIELD ON character by jumping to line 130 from line 200. After this, the words "HIT ANY KEY" are printed again with the REVERSE FIELD being opposite from the previous time, causing the words to blink. The program continues in this loop until a key is depressed and a return is made through line 170.

Listing 2 is a simple demonstration program using the blink routine in a slightly different manner. The three Blinking Message Subroutines are called by a main program that is printing the current time at the top of the screen. Each subroutine checks to see if it is time for it to blink and, if so, it reverses the field of its message. If the blink time has not been reached, then the subroutine immediately returns to the main program. Notice that the GET statements have been dropped and that each routine must reposition the cursor before printing its message. Each subroutine prints a different message in a different screen location and blinks at a different rate in this DEMO program. Notice also that the blink rate (25 in line 270) could be made a variable that could be modified elsewhere in a program to increase or decrease the blink rate of certain messages. A very large number could be used to turn the blink function

on or off if desired.

The blinking routines shown can be used in many applications. They give the impression that the computer is doing more than one operation at a time. Let these short routines tell your secretary to turn on the printer, or let them highlight when an inventory item is about depleted, or, most importantly, let them warn of unseen Klingons and those inoperative phasor banks!

Program 1.

```
 100 REM  BLINKING GET ROUTINE
 110 REM
 120 PRINT"SQQQQQQQQQQQQQQQQQQQQQ"
 130 X$="R": GOTO150
 140 X$="■"
 150 PRINTTAB(13);X$;"HIT ANY KEY⌝":H=TI
 160 IFTI>H+15THEN190
 170 GETS$:IFS$<>""THENRETURN
 180 GOTO160
 190 IFX$="R"THEN140
 200 GOTO130
READY.
```

Program 2.

```
 10 PRINT"⌝"
 20 PRINT"STHE TIME IS ";LEFT$(TI$,2);":
   ";MID$(TI$,3,2);":";RIGHT$(TI$,2)
 30 GOSUB100:GOSUB200:GOSUB300
 40 GOTO20
 90 REM
 100 REM  BLINKING MESSAGE SUBROUTINES
 110 REM
 120 IFH1<>0THEN170
 130 X1$="R":GOTO150
 140 X1$="■"
 150 PRINT"SQQQQQQQQQQQQQQQQQQQQQ"
 160 PRINTX1$;"TIME IS PASSING⌝":H1=TI
 170 IFTI<H1+15THENRETURN
 180 IFX1$="R"THEN140
 190 GOTO130
 200 REM
 210 REM
 220 IFH2<>0THEN270
 230 X2$="R":GOTO250
 240 X2$="■"
 250 PRINT"SQQQQQQQQQQQQQQQQ"
 260 PRINTTAB(15)X2$;"TIME HAS PASSED⌝":H2=TI
 270 IFTI<H2+25THENRETURN
 280 IFX2$="R"THEN240
 290 GOTO230
 300 REM
 310 REM
 320 IFH3<>0THEN370
 330 X3$="R":GOTO350
 340 X3$="■"
 350 PRINT"SQQQQQQQQQ"
 360 PRINTTAB(25)X3$;"TIME IS GONE⌝":H3=TI
 370 IFTI<H3+45THENRETURN
 380 IFX3$="R"THEN340
 390 GOTO330
READY.
```

©

Interfacing A BSR X-10 AC Remote Control System To Your PET

C. W. Ward
Hazelwood, MO

The January 1980 issue of BYTE magazine published an article by Steve Ciarcia entitled "Computerize a Home." In this article Mr. Ciarcia described a method of expanding his home security system to include control of all the lights and AC outlets in the house. In his words, "*This expansion seems to be a contradiction considering previous concern over wiring costs. It would appear that every AC outlet would have to be directly wired to the computer through relays or some remote control capability would have to be added to each light and appliance.*"

It is this latter approach that is the basis for this and Mr. Ciarcia's article: more specifically, interfacing a BSR X-10 AC Remote Control System to a computer. His approach differs however, in that it proposes building a relatively simple, but not so inexpensive, hardware interface between his computer (a Radio Shack TRS-80 Model I) and the BSR X-10 command console. This article proposes a software oriented solution. In terms of hardware, it requires no more than the connection of an ultrasonic transducer (2 wires) to the appropriate pins of the PET's user port, thus resulting in an even lower cost. It should be noted that the concept applies to any system with an available output port (one line). The machine language program would require rework depending upon the speed of the system's clock (the PET has a 1 megahertz clock) and the number of cycles per instruction.

The BSR X-10 system consists of a group of electronic devices that control the electrical environment inside and outside of of your house. Lights can be turned on and off, dimmed or brightened. Plug-in appliances, television sets, and stereos can be turned on or off. No special wiring or complex installations are involved. The system is expandable by adding more modules, a cordless controller or other command consoles. The system is shown in photo 1 and photo 2. The components are: The command console, the wall switch module, the appliance module, and the lamp module. A cordless controller (not shown) is available, but is not an integral part of the system being presented in this article.

The BSR X-10 system is marketed by Sears, Penneys and Radio Shack, to mention just a few, under slightly different product names. The components of one vendor's system are interchangeable with those of another, with one very notable exception. There are two different types of command consoles: one model (X10-014301) that can be remotely controlled with a hand held cordless controller, and one model that cannot! The command console that can be controlled by the hand held cordless controller is a requirement. The buyer should make certain that the BSR X-10 system purchased includes this particular model.

Photo 1: The BSR X-10 command console, appliance module and lamp module.

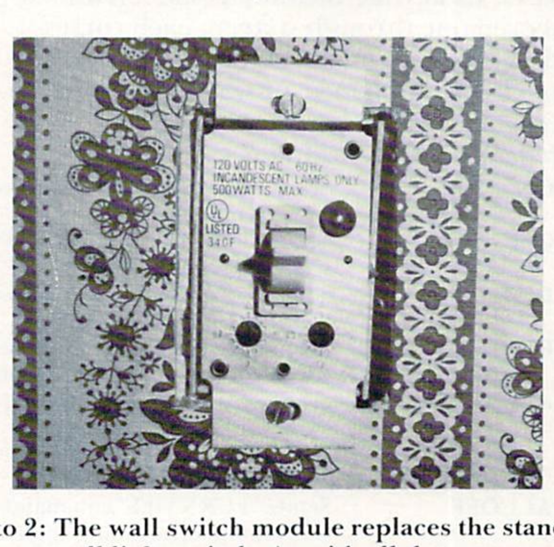


Photo 2: The wall switch module replaces the standard wall light switch. As with all the remote modules, it can be locally activated without the command console.

Photo 3: The 40K Hertz transducer, shielded cable and receptacle for the PET's parallel user port.

The BSR X-10 components are quite inexpensive when compared to the cost of the alternatives. The command console sells for $39 while each remote module sells for $15.

The command console of the system operates by sending coded signals through the house wiring to the lamp, appliance, and wall switch modules. Each remote module monitors these transmissions and responds only when its particular code is sent. The coded signals sent by the command console can be initiated by physically pressing a key on the console or by the receipt of a series of tone bursts in the proper sequence through the ultrasonic receiver section. This series of tone bursts would typically be transmitted by the hand-held cordless controller as a result of pressing one of its twenty-two keys. Its keyboard contains sixteen unit keys, numbered one through sixteen, each corresponding to the number set (or dialed) on the remote lamp, appliance, or wall switch module. The keyboard also contains six command keys labeled and defined as follows:

ON	Sends "TURN ON" command to selected module.
OFF	Sends "TURN OFF" command to selected module.
DIM	Sends "DIM" command to selected lamp or wall switch module.
BRIGHT	Sends "BRIGHTEN" command to selected lamp or wall switch module.
ALL LIGHTS ON	Sends "TURN ON" command to all lamp and wall switch modules simultaneously. (Does not affect appliance modules.)
ALL OFF	Sends "TURN OFF" command to every module, including appliance modules.

The user then selects a given lamp, appliance or wall switch module by pressing the appropriately numbered unit key and initiates the desired function by pressing the appropriate command key. This action actually transmits two separate messages to the command console.

Figure 1 describes the format of the coded messages (tone bursts) sent by the cordless controller to the command console via ultrasonnic communication. Each of the twenty-two keys on the controller has a unique 5-bit binary code (see table 1). A single message is made up of a start-of-message (SOM) code, one 5-bit binary code, the logical inversion of that 5-bit binary code, and an end of message (EOM) code. One message is approximately 100 ms in length and is composed of thirteen segments. Each segment is 8 ms in length. The start of message segment consists of a 4 ms 40K hertz tone followed by a 4 ms period of silence. Each segment of the data (5-bit binary code or inverted 5-bit binary code) consists of a 4 ms 40K hertz tone for a logic 1 or a 1.2 ms 40K hertz tone for a logic 0, followed by a silent period of the appropriate length. The end of message code consists of two 8 ms segments, each containing a 40K hertz tone for the complete duration. All messages use exactly the same format; only the 5-bit binary data codes vary.

Safety is the primary consideration. There is no hazard in using the controller or any of the remote modules as long as their cases remain intact. The BSR X-10 is Underwriters Laboratories listed. The PET must remain electrically isolated from the command console at all times. This is accomplished with communication in the form of ultrasonic sound transmitted through space by the transducer attached to the PET's user port. In essence, the PET software will simulate the activity of the hand-held cordless controller.

The hardware task consists of soldering the two wires of an output transducer to pin C (PAO) and pin A (GND) of a receptacle for the PET's parallel

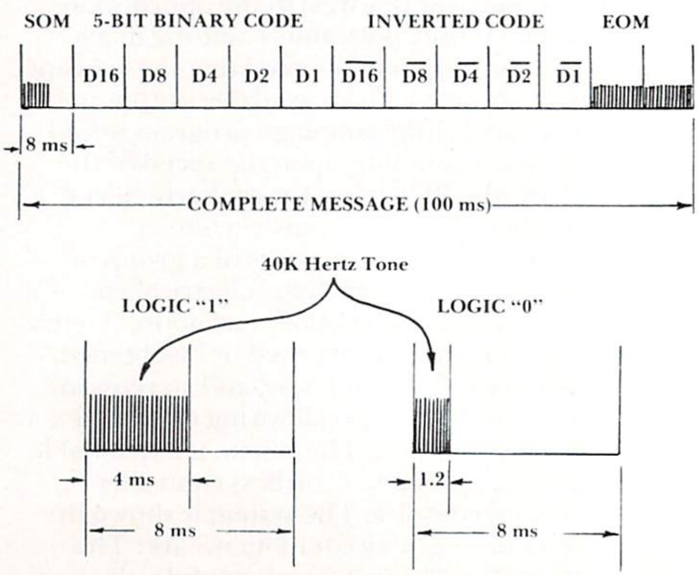


Figure 1: Description of coded message sent from cordless controller to the command console.

user port. The cable between the transducer and the receptacle is typically shielded (see photo 3).

Several sources for obtaining the 40K hertz transducer follow:

MASSA Laboratories Inc.
Hingham, MA
Part number TR-89

The Micromint
917 Midway
Woodmere, NY 11598
Part number 1002

The transducer shown in photo 2 was obtained from a local electronics surplus store, giving reason to believe that the 40K Hertz transducer is relatively common. Another possible source would be the service department of a local TV dealer. Most remote TV controllers today make use of ultrasonic transducers.

The effective range of the transducer is approximately twenty feet. When testing, various distances from the command module should be tried. Experience with several different transducers has shown that some failed to activate the console if positioned extremely close while others worked only in such a position.

Program 1 shows a simple BASIC driver program that serves only to demonstrate how the machine language sub-program should be used. Line 100 requests the user to enter the decimal data code for a given remote unit or a given function. If valid, the value is poked into location zero and the machine language program is executed. This is repeated several times to ensure success. During testing, this loop count should be increased. The REM statements in the program merely document the unit and function codes. If the program is listed just prior to running, the codes will be readily available (displayed on the PET screen) during the demonstration.

Program 2 is an assembled version of the 6502 machine language program required to generate the thirteen 8 ms segments that make up a single message. The routine begins by initializing the data direction register ($E843_{16}$) for VIA data output port A ($E84F_{16}$). PA0 is defined as an output line and PA1 through PA7, although not actually used, are defined as input lines. The remainder of the mainline routine is subdivided into segments containing calls to appropriate subroutines to produce the 8 ms segment in question. Labels on each of the subdivisions correspond to the field labels in figure 1 describing the complete message. The 40K hertz tone on the transducer is produced by alternating the logic value (0 or 1) of pin C (PA0) of the user port at a rate of 40,000 cycles per second. This is accomplished by the machine language subroutine labeled "X40KHZ". The period of time that a given 40K hertz tone is produced is determined by the number of times the subroutine continues to loop. The calling program sets register X accordingly. The periods of silence are accomplished by calling the machine language subroutine labeled "DELAY". Again, the

UNIT CODE	5-BIT BINARY CODE D16	D8	D4	D2	D1	DECIMAL EQUIVALENT
1	0	1	1	0	0	12
2	1	1	1	0	0	28
3	0	0	1	0	0	04
4	1	0	1	0	0	20
5	0	0	0	1	0	02
6	1	0	0	1	0	18
7	0	1	0	1	0	10
8	1	1	0	1	0	26
9	0	1	1	1	0	14
10	1	1	1	1	0	30
11	0	0	1	1	0	06
12	1	0	1	1	0	22
13	0	0	0	0	0	00
14	1	0	0	0	0	16
15	0	1	0	0	0	08
16	1	1	0	0	0	24

COMMAND	5-BIT BINARY CODE D16	D8	D4	D2	D1	DECIMAL EQUIVALENT
ALL OFF	0	0	0	0	1	01
ALL LIGHTS ON	0	0	0	1	1	03
ON	0	0	1	0	1	05
OFF	0	0	1	1	1	07
DIM	0	1	0	0	1	09
BRIGHT	0	1	0	1	1	11

Table 1: Cordless controller push-button codes and decimal equivalents.

```
100 INPUT"?ENTER CODE";C
110 IFC<0ORC>30THENPRINT"INVALID CODE"
    :GOTO100
120 FORI=1TO3:POKE0,C:SYS(31744):NEXT
140 GOTO100
197 REM ------------------------------
198 REM ******************************
199 REM ------------------------------
200 REM   UNIT     CODE     UNIT     CODE
201 REM ------------------------------
202 REM    1        12        9        14
203 REM    2        28       10        30
204 REM    3         4       11         6
205 REM    4        20       12        22
206 REM    5         2       13         0
207 REM    6        18       14        16
208 REM    7        10       15         8
209 REM    8        26       16        24
210 REM ------------------------------
211 REM FUNCTION  CODE   FUNCTION CODE
212 REM ------------------------------
213 REM  ALL OFF     1  ALL LITES ON  3
214 REM    ON        5     OFF        7
215 REM   DIM        9   BRIGHT      11
216 REM ------------------------------
217 REM ******************************
218 REM ------------------------------
READY.
```

period of time is determined by the number of loops. The instruction timings are given in the comments field of the assembled instructions in each subdivision.

The machine language program, as assembled, will load into the high address end of RAM ($1C00_{16}$). It should be loaded and BASIC's upper memory limit reset prior to loading and executing the BASIC driver program (see a previous **COMPUTE!** article).

The approach used to time the various functions (ie. the program loops and sequences of instructions to produce the desired periods) does not interfere with the PET's clock nor does the clock interrupt handling software produce any undesired effects on these timing sequences. Turning on a table lamp, then, is as simple as poking a value and executing a machine language program. A sophisticated BASIC program can now be developed using the PET's time-of-day clock. Remote BSR X-10 modules can be placed around the home to control a variety of appliances and lights, and all at a much lower cost!

```
                      ORG    X'0000'
            * PAGE ZERO WORKING STORAGE
0000        CODE      DS     1X       CALLING PARAMETER
0001        ONE       DS     1X
0002        TEMP      DS     1X
            * SYSTEM EQUATES
            DDREG     EQU    X'E843'  DATA DIRECTION REGISTER
            PORT      EQU    X'E84F'  VIA DATA OUT PORT A
                      ORG    X'1C00'
1C00 A901   START     LDX    #1
1C02 8D43E8           STA    DDREG
1C05 8501             STA    ONE
            *--------------------------SEGMENT 1-------------
1C07 A2A0   SOM       LDX    #160
1C09 20631D           JSR    X40KHZ   4000,8
1C0C A24F             LDX    #79        2
1C0E 20791D           JSR    DELAY    6,3962
1C11 0502             ORA    TEMP     3
            *--------------------------SEGMENT 2--------------
1C13 A500   D16       LDA    CODE     3
1C15 2910             AND    $10      2
1C17 F00D             BEQ    OVER1    3*
1C19 A2A0             LDX    #160       2
1C1B 20631D           JSR    X40KHZ   6,4,4000,8
1C1E A24F             LDX    #79        2
1C20 20791D           JSR    DELAY    6,3962
1C23 4C331C           JMP    D8       3
1C26 A230   OVER1     LDX    #48        2
1C28 20631D           JSR    X40KHZ   6,4,1200,8
1C2B A287             LDX    #135       2
1C2D 20791D           JSR    DELAY    6,6762
1C30 4C331C           JMP    D8       3
            *--------------------------SEGMENT 3--------------
1C33 A500   D8        LDA    CODE
1C35 2908             AND    $08      (TIMINGS SAME AS ABOVE)
1C37 F00D             BEQ    OVER2
1C39 A2A0             LDX    #160
1C3B 20631D           JSR    X40KHZ
1C3E A24F             LDX    #79
1C40 20791D           JSR    DELAY
1C43 4C531C           JMP    D4
1C46 A230   OVER2     LDX    #48
1C48 20631D           JSR    X40KHZ
1C4B A287             LDX    #135
1C4D 20791D           JSR    DELAY
```

```
1C50 4C531C            JMP   D4
          *----------------------------SEGMENT 4--------------
1C53 A500   D4         LDA   CODE
1C55 2904              AND   $04        (TIMINGS SAME AS ABOVE)
1C57 F00D              BEQ   OVER3
1C59 A2A0              LDX   #160
1C5B 20631D            JSR   X40KHZ
1C5E A24F              LDX   #79
1C60 20791D            JSR   DELAY
1C63 4C731C            JMP   D2
1C66 A230   OVER3      LDX   #48
1C68 20631D            JSR   X40KHZ
1C6B A287              LDX   #135
1C6D 20791D            JSR   DELAY
1C70 4C731C            JMP   D2
          *----------------------------SEGMENT 5--------------
1C73 A500   D2         LDA   CODE
1C75 2902              AND   $02        (TIMINGS SAME AS ABOVE)
1C77 F00D              BEQ   OVER4
1C79 A2A0              LDX   #160
1C7B 20631D            JSR   X40KHZ
1C7E A24F              LDX   #79
1C80 20791D            JSR   DELAY
1C83 4C931C            JMP   D1
1C86 A230   OVER4      LDX   #48
1C88 20631D            JSR   X40KHZ
1C8B A287              LDX   #135
1C8D 20791D            JSR   DELAY
1C90 4C931C            JMP   D1
          *----------------------------SEGMENT 6--------------
1C93 A500   D1         LDA   CODE
1C95 2901              AND   $01        (TIMINGS SAME AS ABOVE)
1C97 F00D              BEQ   OVER5
1C99 A2A0              LDX   #160
1C9B 20631D            JSR   X40KHZ
1C9E A24F              LDX   #79
1CA0 20791D            JSR   DELAY
1CA3 4CB31C            JMP   NOTD16
1CA6 A230   OVER5      LDX   #48
1CA8 20631D            JSR   X40KHZ
1CAB A287              LDX   #135
1CAD 20791D            JSR   DELAY
1CB0 4CB31C            JMP   NOTD16
          *----------------------------SEGMENT 7--------------
1CB3 A500   NOTD16     LDA   CODE       3
1CB5 2910              AND   $10        2
1CB7 D00D              BNE   OVER6      3*
1CB9 A2A0              LDX   #160        2
1CBB 20631D            JSR   X40KHZ     6,4,4000,8
1CBE A24F              LDX   #79         2
1CC0 20791D            JSR   DELAY      6,3962
1CC3 4CD31C            JMP   NOTD8      3
1CC6 A230   OVER6      LDX   #48         2
1CC8 20631D            JSR   X40KHZ     6,4,1200,8
1CCB A287              LDX   #135        2
1CCD 20791D            JSR   DELAY      6,6762
1CD0 4CD31C            JMP   NOTD8      3
          *----------------------------SEGMENT 8--------------
1CD3 A500   NOTD8      LDA   CODE
1CD5 2908              AND   $08        (TIMINGS SAME AS ABOVE)
```

```
1CD7 D00D                 BNE    OVER7
1CD9 A2A0                 LDX    #160
1CDB 20631D               JSR    X40KHZ
1CDE A24F                 LDX    #79
1CE0 20791D               JSR    DELAY
1CE3 4CF31C               JMP    NOTD4
1CE6 A230     OVER7       LDX    #48
1CE8 20631D               JSR    X40KHZ
1CEB A287                 LDX    #135
1CED 20791D               JSR    DELAY
1CF0 4CF31C               JMP    NOTD4
              *-------------------------SEGMENT 9-------------
1CF3 A500     NOTD4       LDA    CODE
1CF5 2904                 AND    $04         (TIMINGS  SAME AS ABOVE)
1CF7 D00D                 BNE    OVER8
1CF9 A2A0                 LDX    #160
1CFB 20631D               JSR    X40KHZ
1CFE A24F                 LDX    #79
1D00 20791D               JSR    DELAY
1D03 4C131D               JMP    NOTD2
1D06 A230     OVER8       LDX    #48
1D08 20631D               JSR    X40KHZ
1D0B A287                 LDX    #135
1D0D 20791D               JSR    DELAY
1D10 4C131D               JMP    NOTD2
              *-------------------------SEGMENT 10------------
1D13 A500     NOTD2       LDA    CODE
1D15 2902                 AND    $02         (TIMINGS SAME AS ABOVE)
1D17 D00D                 BNE    OVER9
1D19 A2A0                 LDX    #160
1D1B 20631D               JSR    X40KHZ
1D1E A24F                 LDX    #79
1D20 20791D               JSR    DELAY
1D23 4C331D               JMP    NOTD1
1D26 A230     OVER9       LDX    #48
1D28 20631D               JSR    X40KHZ
1D2B A287                 LDX    #135
1D2D 20791D               JSR    DELAY
1D30 4C331D               JMP    NOTD1
              *-------------------------SEGMENT 11------------
1D33 A500     NOTD1       LDA    CODE
1D35 2901                 AND    $01         (TIMINGS SAME AS ABOVE)
1D37 D00D                 BNE    OVER10
1D39 A2A0                 LDX    #160
1D3B 20631D               JSR    X40KHZ
1D3E A24F                 LDX    #79
1D40 20791D               JSR    DELAY
1D43 4C531D               JMP    EOM
1D46 A230     OVER10      LDX    #48
1D48 20631D               JSR    X40KHZ
1D4B A287                 LDX    #135
1D4D 20791D               JSR    DELAY
1D50 4C531D               JMP    EOM
              *-------------------------SEGMENT 12 & 13-------
1D53 A2FF     EOM         LDX    #255
1D55 20631D               JSR    X40KHZ
1D58 A2FF                 LDX    #255
1D5A 20631D               JSR    X40KHZ
1D5D A282                 LDX    #130
```

```
1D5F 20631D              JSR    X40KHZ
1D62 60                  RTS
             * GENERATE A 40K HERTZ FREQUENCY ON VIA PORT A BIT 0
             * FOR A LENGTH OF TIME DETERMINED BY REGISTER X
1D63 A900    X40KHZ      LDX    #00        2
1D65 EA                  NOP               2
1D66 8D4FE8 LOOP1        STA    PORT       4 ╮
1D69 6901                ADC    #1         2 |
1D6B D8                  CLD               2 |13
1D6C D8                  CLD               2 |
1D6D 8502                STA    TEMP       3 ╯
1D6F 8D4FE8              STA    PORT       4 ╮
1D72 6501                ADC    ONE        3 |12
1D74 CA                  DEX               2 |
1D75 D0EF                BNE    LOOP1      3*╯
1D77 EA                  NOP               2
1D78 60                  RTS               6
             * DELAY FOR A LENGTH OF TIME DETERMINED BY
             * THE VALUE  IN REGISTER X
1D79 0502    DELAY       ORA    TEMP       3
1D7B 0D0200              ORA    TEMP       4
1D7E 20901D LOOP2        JSR    RETURN     6,6╮
1D81 20901D              JSR    RETURN     6,6|
1D84 20901D              JSR    RETURN     6,6|
1D87 0502                ORA    TEMP       3   |50
1D89 0D0200              ORA    TEMP       4   |
1D8C EA                  NOP               2   |
1D8D CA                  DEX               2   |
1D8E D0EE                BNE    LOOP2      3*  ╯
1D90 60      RETURN      RTS               6
1D91                     END    START------------------------------------
```

©

Using Non-Pin-Feed Forms In The 2022 Tractor Printer

Rev. Jack Weaver
Homestead FL

Most of the forms we use in our operation are punched for our 2022 Tractor Feed CBM Printer. However there are some things we need to print out on our printer, things for which we could not justify the high cost of having printed. We would use relatively few over a year's time. Two examples that come to mind are Bank Deposit Slips and payroll checks. In my son's business we write three payroll checks each week and we have about 50 to 75 checks to deposit in the bank each week.

We have been using a very easy and unique way to print on standard checks and standard bank deposit forms using our 2022 printer. We have found it very exact and that it can be perfectly registered each time the forms are used.

Basically the method we use is as follows: We take one or more sheets of blank white fan-fold pin feed paper and very carefully lay the proposed forms out on the paper — tracing the outline of each one on the pin-feed paper. The corners of each form are clearly marked in black ink.

The pin-feed paper is then taken and laid on a flat surface outside and carefully sprayed with #1301 KRYLON brand Crystal Clear Acrylic spray coating. We mention this brand because we have found that it will be less likely to crack or peel. After the one side is completely dry — turn it over and spray the back side. Repeat for a total of four or five coats — allowing time for each coat to fully dry.

When the paper is completely dry you will in effect have a plastic sheet — perforated for your pin feed printer.

It is very important that you use a transparent type opaque cellophane tape and tape the tear perforations so it will not separate as it is being used. The tape should also be taped over the side and top margins of each form outline — so each form can be taped to the page and then removed without doing damage to the plasticized paper.

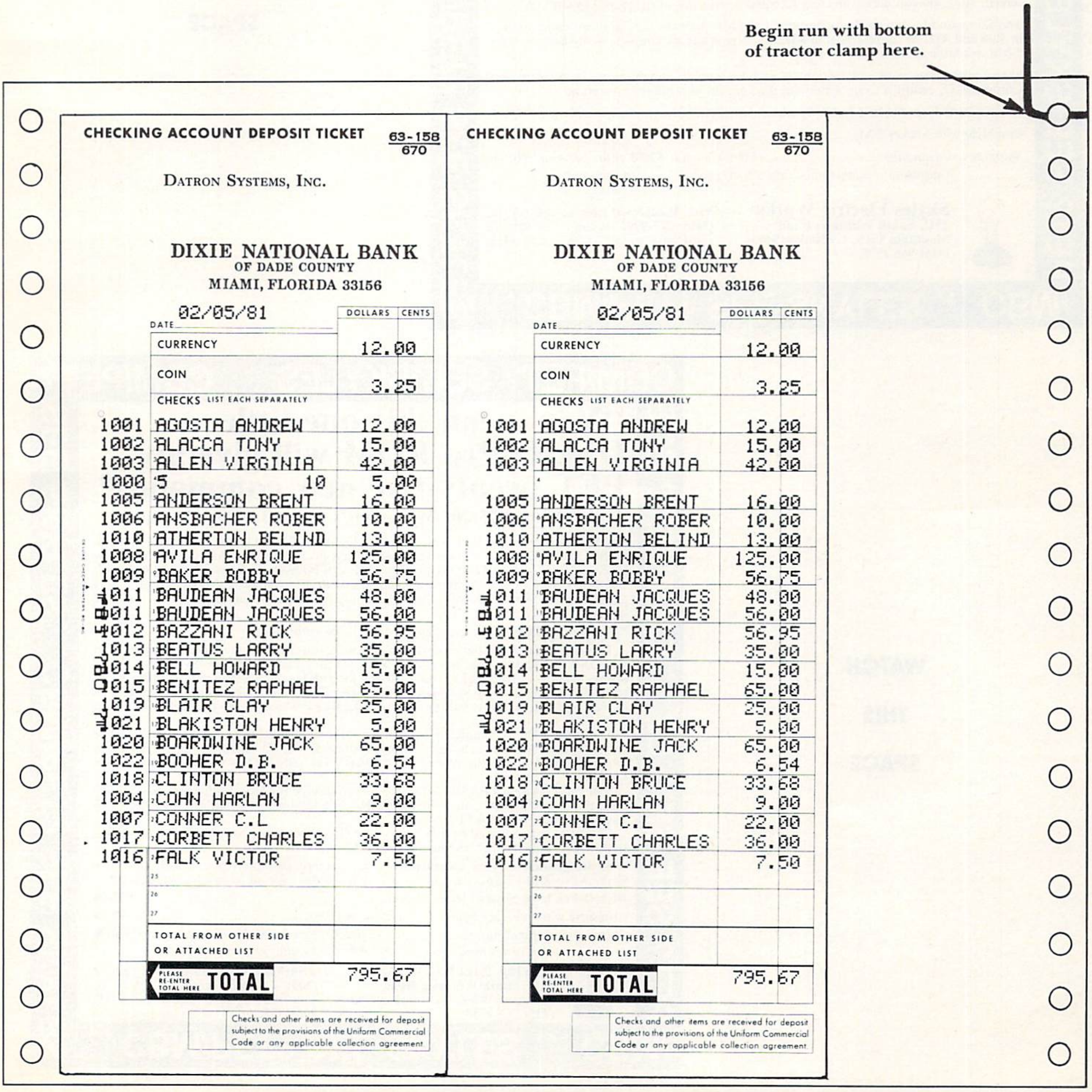
If you are very careful in drawing your outline of the forms — and careful in the format programming of the printer — you can make each copy exactly like the previous one.

Very important — It is necessary to draw a line on the plasticized sheet at the bottom of the lever on the right hand pin-feed mechanism. This will let you know where to start your printing to get perfect registration each time.

The tape on each form is almost unnoticeable since we use only a square one half-inch square at top and bottom of each form — which, when removed from the plasticized paper form holder, is simply folded over to the back side.

Our bank tellers were impressed and could not figure out how we had done it. When making Bank Deposits we use the form in conjunction with a program which credits the check to the customer while at the same time makes a very neat Bank Deposit Slip.

The possibilities for this are unlimited. We have been using one sheet for 6 months and it is still going strong — maybe for another 6 months!!



CHECKING ACCOUNT DEPOSIT TICKET 63-158 / 670

DATRON SYSTEMS, INC.

DIXIE NATIONAL BANK
OF DADE COUNTY
MIAMI, FLORIDA 33156

DATE 02/05/81

		DOLLARS CENTS
CURRENCY		12.00
COIN		3.25
CHECKS LIST EACH SEPARATELY		
1001	AGOSTA ANDREW	12.00
1002	ALACCA TONY	15.00
1003	ALLEN VIRGINIA	42.00
1000	5 10	5.00
1005	ANDERSON BRENT	16.00
1006	ANSBACHER ROBER	10.00
1010	ATHERTON BELIND	13.00
1008	AVILA ENRIQUE	125.00
1009	BAKER BOBBY	56.75
1011	BAUDEAN JACQUES	48.00
1011	BAUDEAN JACQUES	56.00
1012	BAZZANI RICK	56.95
1013	BEATUS LARRY	35.00
1014	BELL HOWARD	15.00
1015	BENITEZ RAPHAEL	65.00
1019	BLAIR CLAY	25.00
1021	BLAKISTON HENRY	5.00
1020	BOARDWINE JACK	65.00
1022	BOOHER D.B.	6.54
1018	CLINTON BRUCE	33.68
1004	COHN HARLAN	9.00
1007	CONNER C.L	22.00
1017	CORBETT CHARLES	36.00
1016	FALK VICTOR	7.50
25		
26		
27		
TOTAL FROM OTHER SIDE OR ATTACHED LIST		
PLEASE RE-ENTER TOTAL HERE **TOTAL**		795.67

Checks and other items are received for deposit subject to the provisions of the Uniform Commercial Code or any applicable collection agreement

CHECKING ACCOUNT DEPOSIT TICKET 63-158 / 670

DATRON SYSTEMS, INC.

DIXIE NATIONAL BANK
OF DADE COUNTY
MIAMI, FLORIDA 33156

DATE 02/05/81

		DOLLARS CENTS
CURRENCY		12.00
COIN		3.25
CHECKS LIST EACH SEPARATELY		
1001	AGOSTA ANDREW	12.00
1002	ALACCA TONY	15.00
1003	ALLEN VIRGINIA	42.00
1005	ANDERSON BRENT	16.00
1006	ANSBACHER ROBER	10.00
1010	ATHERTON BELIND	13.00
1008	AVILA ENRIQUE	125.00
1009	BAKER BOBBY	56.75
1011	BAUDEAN JACQUES	48.00
1011	BAUDEAN JACQUES	56.00
1012	BAZZANI RICK	56.95
1013	BEATUS LARRY	35.00
1014	BELL HOWARD	15.00
1015	BENITEZ RAPHAEL	65.00
1019	BLAIR CLAY	25.00
1021	BLAKISTON HENRY	5.00
1020	BOARDWINE JACK	65.00
1022	BOOHER D.B.	6.54
1018	CLINTON BRUCE	33.68
1004	COHN HARLAN	9.00
1007	CONNER C.L	22.00
1017	CORBETT CHARLES	36.00
1016	FALK VICTOR	7.50
25		
26		
27		
TOTAL FROM OTHER SIDE OR ATTACHED LIST		
PLEASE RE-ENTER TOTAL HERE **TOTAL**		795.67

Checks and other items are received for deposit subject to the provisions of the Uniform Commercial Code or any applicable collection agreement

How And Why You Should Use PEEK (155) Instead Of GET

David M. Miller, Howard Beach, NY

In many two player action games, one player has a keyboard-screen control advantage over the other. By this I mean if player A holds down the "2" key for instance, and player B presses the "W" key, the PET will only acknowledge the depression of the "2." But if player B holds down the "W" first and player A presses the "2," the PET will again only be aware of the depression of the "2." This problem arises when you use the GET command. To illustrate this problem more clearly for those of you who are not familiar with it, type in the following program:

```
10 GET A$:IF A$ = "" THEN 10
20 PRINT A$: GOTO 10
```

If the "2" and "W" are pressed together (in action games it is likely that two keys may at the same time be depressed) the PET prints only the "2." Even if the "W" is pressed first, and then the "2," in the end, when both are depressed the PET will only print the "2." That, of course, gives player A a large advantage in the game.

The easiest way to overcome this unfairness is to use the memory content of PEEK(155) in Upgrade ROMs. I think it is PEEK (512) in Original ROMs, but I did not test it. Using this instead of GET gives both players an equal keyboard-screen control over his "man."

The only drawback using this method is that each player has control over only two keys each, so this feature would only be advantageous in certain games.

The left player must use the RVS key and SPACE key; the right player must use the left carat ‹, and left bracket [.

Memory 155 has a value of 255 unless one or more of the four keys mentioned above are depressed. When 2, 3, or all 4 keys are depressed independently or simultaneously, memory 155 has a special value which is given in the chart below.

Key(s) Depressed	Value of PEEK(155)
NONE	255
RVS	254
SPACE	251
‹	247
[	253
RVS, SPACE	250
RVS, [	252
RVS, ‹	246
SPACE, [	249
SPACE, ‹	243
[, ‹	254
RVS, SPACE, [	242
RVS, SPACE, ‹	241
RVS, SPACE, [, ‹	240

Note: PEEK(155) never has a value of 244.

Using these values in your program you can branch off to a step which will carry out the required function. To illustrate how you can use this in a game situation, consider a game in which two players have two guns each. Part of the program may look as follows:

```
10 IF PEEK(155)=254 THEN REM SHOOT TOP
LEFT GUN
20 IF PEEK(155)=249 THEN REM SHOOT TOP
RIGHT AND BOTTOM LEFT GUN
30 IF PEEK(155)=240 THEN REM SHOOT ALL
FOUR GUNS
```

Using PEEK(155) in place of GET makes the keyboard-screen control, and, in effect, the entire game, fair and equal for both players. I hope you can take advantage of this feature in programming your next interactive action game, or revising an old one. If you have any comments on this idea, please send them directly to me.

David Miller
149-45 83 St.
Howard Beach, NY 11414

©

THE SBC GAZETTE

AIM 65 BASIC Floating-Point Arithmetic From Machine Language

Paul Beasley,
Mobile, AL

Writing floating-point operations in machine language on a microprocessor is a "messy" proposition. I avoid it like the plague unless I absolutely must do it. But I have discovered how to use the floating-point routines in the AIM 65 BASIC ROM's. It's so easy even I do not mind floating-point applications any more.

AIM 65 BASIC Floating-Point Numbers

For those who are unfamiliar with floating-point numbers, particularly on the AIM 65, I'll describe the floating-point number format. Floating-point representations are similar to scientific notation. An example of a number written in normalized, scientific notation is $.27 \times 10^2$ (=27). Computers commonly use a similar scheme except instead of 10 as a base, the base 2 is used (e.g., 27 = .84375 $\times 2^5$). By storing the sign, the exponent of 2, and the mantissa of the number, a broad range of values can be efficiently represented. In the AIM 65, this is accomplished by storing each floating-point number in five consecutive bytes as follows:

1	2	3	4	5
E	M3	M2	M1	M0

S = bit 7

E = exponent
M3,M2,M1,M0 = mantissa
S = sign

Note: Bits in a byte are numbered 0 (LSB) to 7 (MSB).

The exponent, E, is a power of 2 and is biased so that E = $80 actually corresponds to a power of 0, E = $7F corresponds to -1, E = $81 corresponds to +1, etc. When a floating-point number is normalized, the mantissa is shifted so that the first 1 bit of the mantissa falls in bit position 7 of M3. This means that bit 7 of M3 will always be 1 and the exponent reflects the number of bits that the mantissa was shifted in order to have the implied decimal in front of the first 1 bit. E = $80 means no shifts were required; E = $81 means the mantissa was shifted right one bit; E = $7F means the mantissa was shifted left one bit; etc.

Since bit 7 of M3 is always 1 using the above method, it is stripped off and restored only when performing arithmetic operations (this process is explained later). So, when a number is stored in memory, this bit position is used to store the sign of the number — 0 for positive and 1 for negative. (Incidentally, the floating-point representation of 0 is all five bytes equal $00.) My previous example of the number 27 would be stored in memory as follows:

85 58 00 00 00

AIM 65 BASIC Floating-Point Accumulators

In order to use floating-point numbers in arithmetic operations, BASIC reserves twelve bytes in Page 0 to provide two floating-point accumulators. Accumulator 1 (FPAC1) is in locations $A9 through $AE and accumulator 2 (FPAC2) is in locations $B1 through $B6. Each accumulator spans six bytes and has the following format:

1	2	3	4	5	6
E	M_3	M_2	M_1	M_0	S

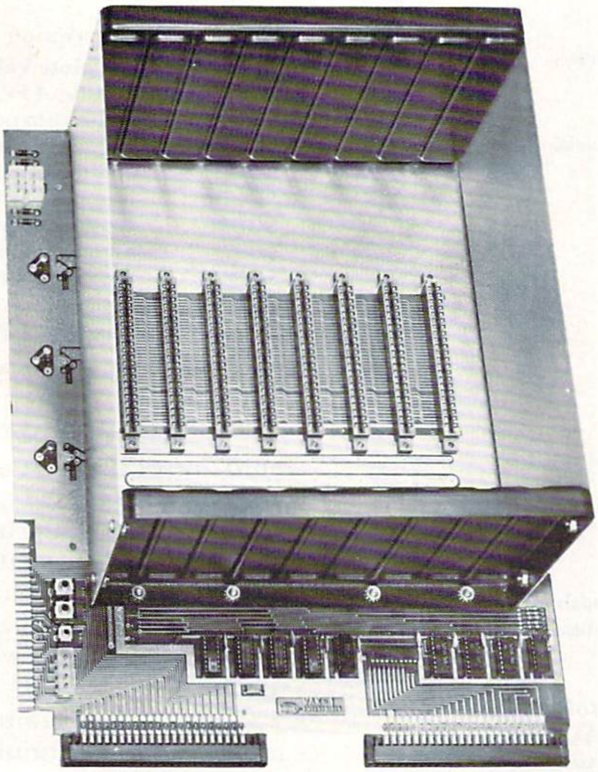
S = $00 for +
$FF for -

As I mentioned earlier, when numbers are stored into memory, the sign is put into bit 7 of M3. Technically, this is accomplished as follows:

($M_3 \wedge$ $7F) — strips off most significant bit of M3
(S $\wedge$ $80) — strips off all bits except leftmost bit

$\vee$ denotes logical OR
$\wedge$ denotes logical AND

VAK-1 MOTHERBOARD



We also carry:

SYM-1	$229.00
AIM-65 w/1K	389.00
AIM-65 w/4K	439.00

We also do custom hardware and software for the 6502 microprocessor

Special price for prepaid customers only (Including Master Card and Visa. Exp. 11/30 $119.00

PRICE: ~~$189.00~~

Call or write for shipping charges and our complete catalog.

The VAK-1 was specifically designed for use with the KIM-1, SYM-1 and the AIM 65 Microcomputer Systems. The VAK-1 uses the KIM-4* Bus Structure, because it is the only popular Multi-Sourced bus whose expansion boards were designed specifically for the 6502 Microprocessor.

SPECIFICATIONS:

- Complete with rigid CARD-CAGE
- Assembled (except for card-cage). Burned in and tested.
- All IC's are in sockets
- Fully buffered address and data bus
- Uses the KIM-4* Bus (both electrical Pin-out and card size) for expansion board slots
- Provides 8 slots for expansion boards on 1″ centers to allow for wire-wrap boards
- Designed for use with a Regulated Power Supply (such as our VAK-EPS) but has provisions for adding regulators for use with an unregulated power supply.
- Provides separate jacks for one audio-cassette, TTY and Power Supply.
- Board size: 14.5 in. Long x 11.5 in. Wide x 8 in. High
- Power requirements; 5V.DC @ 0.2 Amps.

**KIM-4 is a product of MOS Technology/C.B.M.*

RNB ENTERPRISES INCORPORATED

4030 N. 27th Avenue
Suite D.
Phoenix, AZ 85017

master charge VISA

Table 1. Calling sequences for floating-point operations.

OPERATION	CALLING SEQUENCE		
1. Load FPAC1	LDA	AL	source address
	LDY	AH	
	JSR	$C8E1	
2. Load FPAC2	LDA	AL	source address
	LDY	AH	
	JSR	$C7CB	
3. Store FPAC1	LDX	AL	destination address
	LDY	AH	
	JSR	$C913	
4. Copy FPAC1 to FPAC2	JSR	$C94B	
5. Copy FPAC2 to FPAC1	JSR	$C93B	
6. Convert fixed-point to floating-point	LDY	IL	
	LDA	IH	
		(result in FPAC1)	
7. Convert floating-point to fixed-point	Load FPAC1 with floating-point value		
	JSR	$C536	
	(result right-justified in M3-M0 of FPAC1		
8. Addition	Load FPAC1 with operand 1		
	LDA	AL	source address for minuend
	LDY	AH	
	JSR	$C58F	
	(Addressed value is loaded into FPAC2, FPAC1 is subtracted from FPAC2 and result in FPAC1; FPAC2 unchanged.)		
10. Multiplication	Load FPAC1 with operand 1		
	Load FPAC2 with operand 2		
	JSR	$C76F	
	(result in FPAC1; FPAC2 unchanged.)		
11. Division	Load FPAC1 with divisor		
	Load FPAC2 with dividend		
	JSR	$C851	
	(result in FPAC1; FPAC2 unchanged.)		
12. Power operation	Load FPAC1 with exponent		
	Load FPAC2 with base number		
	JSR	$CC7F	
	(FPAC2 is raised to the power in FPAC1; result in FPAC1; FPAC2 unchanged.)		
13. Multiply FPAC1 by 10;	JSR	$C821	
14. Divide FPAC1 by 10	JSR	$C83D	
15. Add .5 to FPAC1	JSR	$C588	
16. Convert floating-point number to ASCII string	Load FPAC1 with number		
	JSR	$CB1C	
	(result at $0200)		

Note: Resulting ASCII string starts at location $0200. The first character is a space, followed by the ASCII digits and ended with a $00 byte.

17. Compare FPAC1 to memory			
	LDA	AL	source address of number in memory
	LDY	AH	
Branch to xxxx if:	JSR	$C99A	
memory<FPAC1	BCC	xxxx	
memory=FPAC1	BEQ	xxxx	
	BEQ	LABEL	
memory>FPAC1	BCS	xxxx	
LABEL	.		
	.		
	.		

Table 2. Intrinsic Function Subroutine Addresses

Basic Function	Address	Description
ABS	$C997	Absolute Value of FPAC1
COS	$CDD2	Cosine of FPAC1
EXP	$CCF1	Raises e to power in FPAC1
INT	$CA0B	Integer portion of FPAC1
LOG	$C729	Natural logarithm of FPAC1
NEG	$CCB8	Negation of FPAC1
RND	$CD96	Generates random number
SGN	$C978	Sign function of FPAC1
SIN	$CDD9	Sine of FPAC1
SQR	$CC75	Square root of FPAC1
TAN	$CE22	Tangent of FPAC1

The logical OR places the sign bit into M3.

When a number is loaded into one of the accumulators, the sign bit is separated out and made the sixth byte of the accumulator (as shown above) so that bit 7 of M3 can be restored to 1. This makes arithmetic operations easier and explains why the accumulators are six bytes each. My example of the number 27 would appear in an accumulator as:

85 D8 00 00 00 00

In addition to the accumulators, there are two other bytes in Page 0 that you should know about. These are the overflow (at $B0) and underflow (at $B8) bytes. The underflow byte is used for rounding M0 of FPAC1. The overflow byte becomes non-zero when a computational result becomes too large. It is important that these two bytes be initialized to zero before the first floating-point operation is performed. In relation to this, I must give a word of caution. The BASIC floating-point routines still "think" they are operating in the context of a BASIC program. This means that any computation error (e.g., overflow) which is normally trapped by BASIC will still be caught and your program terminated. The termination message may look peculiar since the BASIC statement and variable pointers in Page 0 probably contain meaningless values.

Performing The Floating-Point Operations

I have prepared Table 1 as a

reference for the fundamental floating-point operations along with their appropriate machine language calling sequences. All operations are executed with the subroutine jump instruction (JSR) plus minimal parameter set-up. In preparing the table I used the following notation:

AL – Address Low; the least significant 8 bits of the source or destination memory address.

AH – Address High; the most significant 8 bits of the source or destination memory address.

IL – Integer Low; the memory address of the least significant 8 bits of a 2-byte integer value.

IH – Integer High; the memory address of the most significant 8 bits of a 2-byte integer value.

FPAC1 – Floating Point Accumulator 1.

FPAC2 – Floating Point Accumulator 2.

In addition to the fundamental operations in Table 1, the BASIC intrinsic functions may also be used. The common calling sequence for these functions is as follows:

> **load FPAC1 with the argument value**
> **JSR $xxxx (select function address from Table 2)**
> **(result in FPAC1)**

The entry point address for each of the functions is given in Table 2.

Sample Program

In order to illustrate what I have just described, I have included the following sample program. It is a very simple calculation of the volume of a cylinder using the formula $V = \pi r^2h$, where r = radius and h = height. I know that r^2 can be computed as r times r very efficiently, but I used the power function to illustrate its use. When the program finishes (successfully), it will display V = 88357.2935. Another tidbit I'll point out is that the floating-point representation for 2 π is at location $CE53 of the BASIC ROM's.

Sample Program: Calculate Volume of Cylinder ($V = \pi r^2h$)

```
*=$0220
COMIN=$R1A1                         ; monitor entry for command input
EQUAL=$E7D8                         ; output "=" to display/printer
OUTPUT=$E97A                        ; output char. in A to diaplay/printer
CRLOW=$EA13                         ; output CR & LF to display/printer
FMUL=$C76F                          ; floating-point multiply
CONVIF=$C0D1                        ; convert fixed-point to floating-point
CONVFA=$CB1C                        ; convert floating-point to ASCII string
FST1=$C913                          ; store FPAC1
FLD2=$C7CB                          ; load FPAC2
CPY12=$C94B                         ; copy FPAC1 to FPAC2
FDIV=$C851                          ; division
FPWR=$CC7F                          ; power operation
PI2=$CE53                           ; 2*
START    LDY    R                   ; get radius
         LDA    #0
         STA    $B8                 ; initialize underflow
         STA    $B0                 ; and overflow bytes
         JSR    CONVIF
         LDX    #<TEMP
         LDY    #>TEMP
         JSR    FST1                ; store R in TEMP
         LDY    #2
         LDA    #0
         JSR    CONVIF              ; exponent 2 in FPAC1
         LDA    #<TEMP
         LDY    #>TEMP
         JSR    FLD2                ; load R in FPAC2
         JSR    FPWR                ; raise R to power 2
         LDX    #<TEMP
         LDY    #>TEMP
         JSR    FST1                ; store R squared in TEMP
         LDY    H
         LDA    #0
         JSR    CONVIF              ; height H in FPAC1
         LDA    #<TEMP
         LDY    #>TEMP
         JSR    FLD2                ; load FPAC2 with R squared
         JSR    FMUL                ; FPAC1=H times R squared
         LDA    #<PI2
         LDY    #>PI2
         JSR    FLD2                ; load 2*    innto FPAC2
         JSR    FMUL                ; FPAC1=H times R squared times 2
         JSR    CPY12               ; save FPAC1 in FPAC2
         LDY    #2
         LDA    #0
         JSR    CONVIF              ; FPAC1=2
         JSR    FDIV                ; divide by 2
         JSR    CONVFA              ; resulting volume in FPAC1
         JSR    CRLOW
         LDA    #'V'
         JSR    OUTPUT              ; display 'V'
         JSR    EQUAL               ; display '='
         LDX    #0
LABEL1   LDA    $0200,X             ; fetch & display ASCII digits
         BEQ    LABEL2
         JSR    OUTPUT
         INX
         JMP    LABEL1
LABEL2   JSR    CRLOW
         JMP    COMIN
R        .BYTE  25                  ; radius=25
H        .BYTE  45                  ; height=45
TEMP     .BYTE  0,0,0,0,0
         .END
```

©

A General Purpose BCD-To-Binary Routine

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO

A number of routines have been published [1,2,3] that will convert either a two-digit number or a four-digit number in BCD code to a binary number, and Butterfield[4] has published a routine to handle a six-digit BCD number. The routine described here can be easily modified to handle any number of BCD digits. It is a 6502 assembly language interpretation of an algorithm found in Peatman's[5] book. The BCD-to-binary routine assumes its importance from the fact that human beings usually input numbers to a computer in a decimal representation. A number of scientific instruments have BCD outputs that may be interfaced to a microcomputer, requiring some kind of conversion routine before the data from such a device can be processed. Finally, if you want to interface some of the calculator chips to a microprocessor in order to do more complex arithmetic, you will very likely need a BCD-to-binary routine somewhere in your software. A 6502 assembly language routine to go the other way (binary-to-BCD) can be found as a subroutine in reference six at the end of this article.

The BCD-to-binary routine is based on a familiar technique for converting a base-ten number to a base-two number. The decimal number is successively divided by two, and the remainders are noted as either a one or a zero. Each division gives the next more significant binary digit or bit. Example 1 illustrates the process.

Example 1. Convert 59_{ten} to a binary number.

Solution: Successively divide 59_{ten} by two, with the divisions beginning from the right and proceeding to the left.

0	1	3	7	14	29
2) 1	2) 3	2) 7	2) 14	2) 29	2) 59
0	2	6	14	28	58
1	1	1	0	1	1

$59_{ten} = 111011_{two}$

Referring to Example 1 it can be seen that the algorithm requires that the BCD number be successively divided by two and the remainders are saved to become the binary number. The first division remainder is the least significant bit, while the remainder from the last division is the most significant bit. If in Example 1 we wanted to convert 59_{ten} to an eight-bit binary number, namely 0011 1011, we would simply perform two more divisions than shown, providing the two leading zeros in the eight-bit representation.

If you are mildly familiar with BCD numbers you will recall that each digit requires four bits (or one nibble). So an eight-digit decimal number requires four memory locations. Conversely, four memory locations can represent a decimal number as large as 99999999, which is more easily expressed as 10^8-1. Question: How many bits are needed to represent a given number of decimal digits? Let N be the largest number of decimal digits that we need for our particular application, so the largest decimal number is (10^N-1). Let n be the number of binary digits (bits) needed to represent the same number. By analogy, the largest binary number that can be represented by n bits is (2^n-1). Since we wish to represent the same number, we may equate (10^N-1) and (2^n-1) and then solve for n. Thus, with some mathematical magic, the answer to the question posed above is

$$N = N/\log 2 = N/0.30103$$

where a base ten logarithm is implied.

If $N = 8$ then $n = 26.6$ which becomes $n = 27$ when rounded upward (fractional numbers of bits are not allowed as answers for this problem). Twenty-seven bits can be handled quite nicely by four bytes, *but please do not create* your own theorem that the number of memory locations needed to represent a number in binary is equal to the number of memory locations to represent the same number in binary-coded decimal (BCD). Use the equation, and be sure to allocate enough memory to handle the number in either binary or BCD representations. Note that, in the program described by Listing 1, we assume an eight-digit decimal number is being converted to a binary number that will also be stored in four memory locations. The program is easily modified to handle situations where the number of memory locations needed for the BCD number is *different* than the number of memory locations needed for the binary number. Using the immortal words of many authors, "we leave this problem for the student."

So we know how many memory locations to assign to represent the number, and we have a simple algorithm (divide by two and store the remainder) to perform the conversion. Enter some corollary to Murphy's Laws: "nothing is as simple as it seems." Dividing by two is neat and easy for a binary number: successive shifts to the right (LSR or ROR) give successive divisions by two. Dividing by two is considerably more complex for a BCD number. Fortunately, Peatman[5] has pointed out a few tricks that accomplish division-by-two for a BCD number.

The eight bit "weights" in a byte of memory that represent a binary number are 1, 2, 4, 8, 16, 32, 64, and 128, proceeding from the right-most bit to the left-most bit. Clearly, shifting the number to the right divides each bit weight by two. That is why an LSR or an ROR instruction may be used to divide a binary number by two. However, if the same memory location represents a BCD number, then the bit weights are 1, 2, 4, 8, 10, 20, 40, 80. consequently, a shift-right or a rotate-right instruction results in division-by-two only for bits zero, one, two, three, five, six, and seven. Shifting bit four (with a weight of ten) to the right changes its weight to eight. Eight is three more than five, the number you usually get when you divide ten by two. So, the trick to dividing a BCD number by two is to shift right or rotate right as usual, but if a one is shifted from bit four to bit three, then you must subtract three from the shifted-right result to get the correct answer. That's it folks. I wish I could say it was my idea, but I found it in Peatman's[5] book.

```
BCDNUM  = $0000;  Base address of the BCD number to be converted to binary. The
                  most-significant digit of an N digit BCD number is in the high-
                  order nibble of BCDNUM.
BINUM   = $0010;  Base address of the binary number whose most-significant byte
                  will be in BINUM.
BYTE    = $FC;    Twos complement of the number of bytes needed to hold
                  the BCD number; in this program four bytes ($0000 - $0003)
                  are used.

$ 0D00 D8        START   CLD                 Clear decimal mode.
  0D01 A9 00             LDA #00             Clear locations that will
  0D03 A2 FC             LDX #BYTE           hold the binary number.
  0D05 95 14     BACK    STA BINUM + 4,X
  0D07 E8                INX
  0D08 D0 FB             BNE BACK            Locations have been
  0D0A 38                SEC                 cleared.
  0D0B A2 FC     THERE   LDX #BYTE           Rotate the binary number
  0D0D 76 14     RETURN  ROR BINUM + 4,X     right, moving the remainder
  0D0F E8                INX                 from the BCD division into
  0D10 D0 FB             BNE RETURN          the binary number.
  0D12 B0 2B             BCS OUT             If the carry is set, the conver-
  0D14 A2 FC             LDX #BYTE           sion is complete.
  0D16 76 04     AGAIN   ROR BCDNUM + 4,X    Start the division-by-two by
  0D18 E8                INX                 shifting BCD number right.
  0D19 D0 FB             BNE AGAIN           Remainder will be in carry
  0D1B 08                PHP                 flag so save it on the stack.
  0D1C A2 FC             LDX #BYTE           Test bit three of each byte to
  0D1E 38                SEC                 see if a one was shifted in.
  0D1F B5 04     LAKE    LDA BCDNUM + 4,X
  0D21 29 08             AND #08             If so, subtract three.
  0D23 F0 06             BEQ FORWD           If not, no correction needed,
  0D25 B5 04             LDA BCDNUM + 4,X    so test bit seven of each byte
  0D27 E9 03             SBC #03             to see if a one was shifted in.
  0D29 95 04             STA BCDNUM + 4,X
  0D2B B5 04     FORWD   LDA BCDNUM + 4,X    Here bit seven is checked.
  0D2D 29 80             AND #$80
  0D2F F0 06             BEQ ARND            No correction.
  0D31 B5 04             LDA BCDNUM + 4,X    Correction: subtract 30.
  0D33 E9 30             SBC #$30
  0D35 95 04             STA BCDNUM + 4,X
  0D37 E8        ARND    INX
  0D38 D0 E5             BNE LAKE            Repeat for all N bytes.
  0D3A 28                PLP                 Get the carry back because it
  0D3B B0 CE             BCC THERE           held the remainder.
  0D3D 90 CC             BCC THERE           Go back and put it in the
  0D3F 60        OUT     RTS                 binary number. Then finish.
```

If the BCD number is to be represented by several bytes, an added complication occurs. Bit seven in the least-significant byte has a weight of 80. Bit zero in the next most significant byte has a weight of 100. Clearly, shifting a one from bit zero of this byte to bit seven of the least-significant byte does not result in a division-by-two because 100/2 is not 80. However, if we subtract 30 after the shift we do get the correct answer. When performing a divide-by-two operation on a multi-byte BCD number, each byte in the number must be tested to see if a one was shifted into either bit three or bit seven, and then the appropriate remedies must be applied if the tests are positive. In short, if a one is shifted into the most-significant bit position of any of the N nibbles used to represent the N digits in BCD, then the nibble must be corrected by subtracting three.

One other point remains to be made. From Example 1 it is clear that we are interested in the remainder after division-by-two. When dividing by two, the remainder is either zero (even dividend) or one (odd dividend). The remainder will be found in the carry flag after a shift-right operation.

Although the comments should make most of the routine understandable, a brief explanation follows. The first instruction in Listing 1 is not needed if the program is already in the binary mode, in which case the routine starts by clearing the locations used to store the binary number. Rather than inserting some kind of loop counter to keep track of the number of divisions-by-two (refer to Example 1), the carry is set by the instruction location at $0D0A, and the remainders are rotated into the binary number until the carry bit that was initially set has rotated through the binary number and into the carry flag once more. Thus, the conversion stops at the BCS OUT instruction at location $0D12. The division-by-two routine takes up the remainder of the program. Note that the carry flag holds the remainder, and it is stored on the stack while the division-by-two routine is finished, after which it is rotated into the binary number in the RETURN loop.

Very likely some improvements in the speed of the routine could be made. In most cases the number of bytes needed for the binary number will be sufficiently close to the number of bytes occupied by the BCD number that no modification will be needed on that score. Remember, BYTE is the *twos complement* of the number of bytes used to represent the BCD number and the binary number. The BCD number is stored in locations $0000 - $0003 in Listing 1, in the sense that the least-significant digit is in the low-order nibble of location $0003 and the most-significant digit is in the high-order nibble of location $0000. These locations must be filled before the routine is called.

REFERENCES

1. **Programming and Interfacing the 6502, With Experiments**, De Jong, Marvin L., Howard W. Sams & Co., Inc., Indianapolis, 1980, p 129.
2. **6502 Assembly Language Programming**, Leventhal, Lance A., Osborne/McGraw-Hill, Inc., Berkeley, 1979, p 7-9.
3. **6502 Software Design**, Scanlon, Leo J., Howard W. Sams & Co., Inc., Indianapolis, 1980, p 147.
4. "Multi-Mode Adder," Butterfield, Jim, **6502 User Notes**, No. 13, p 23.
5. **Microcomputer-Based Design**, Peatman, John B., McGraw Hill, New York, 1977, p 400.
6. "A Floating-Point Binary to BCD Routine," De Jong, Marvin L., **COMPUTE!**, 1981, in press.

©

CAPUTE!
Corrections And Clarifications

COMPUTE! #14, pg. 68 — To clarify Mr. Victor's comments on conversions:

> In Applesoft BASIC, "DIM A$(40)" means that memory should be reserved for the string variables A$(0), A$(1),...,A$(40). (Note: this is 41 variables, not 40, a quirk of BASIC.) The variable "A$" is another variable (the 42nd) used to store the various strings discussed in the article. The statement "DIM A$(40)" at the start of the programs in the article does not mean that A$ will be 40 characters long. The DIM statement does not refer to A$, it refers to the subscripted variable A$(i).

COMPUTE! #15, pg. 80 — the Editor's Note on halting the dynamic mode should have read POKE 842, 12.

COMPUTE! #15, pg. 64 — at the bottom of the page, following the words "clears the flag," this paragraph was omitted:

> But is this efficient? Of course! How could you possibly reduce such a trivial, two-line program segment? Well, maybe you can! Consider the following alternative:

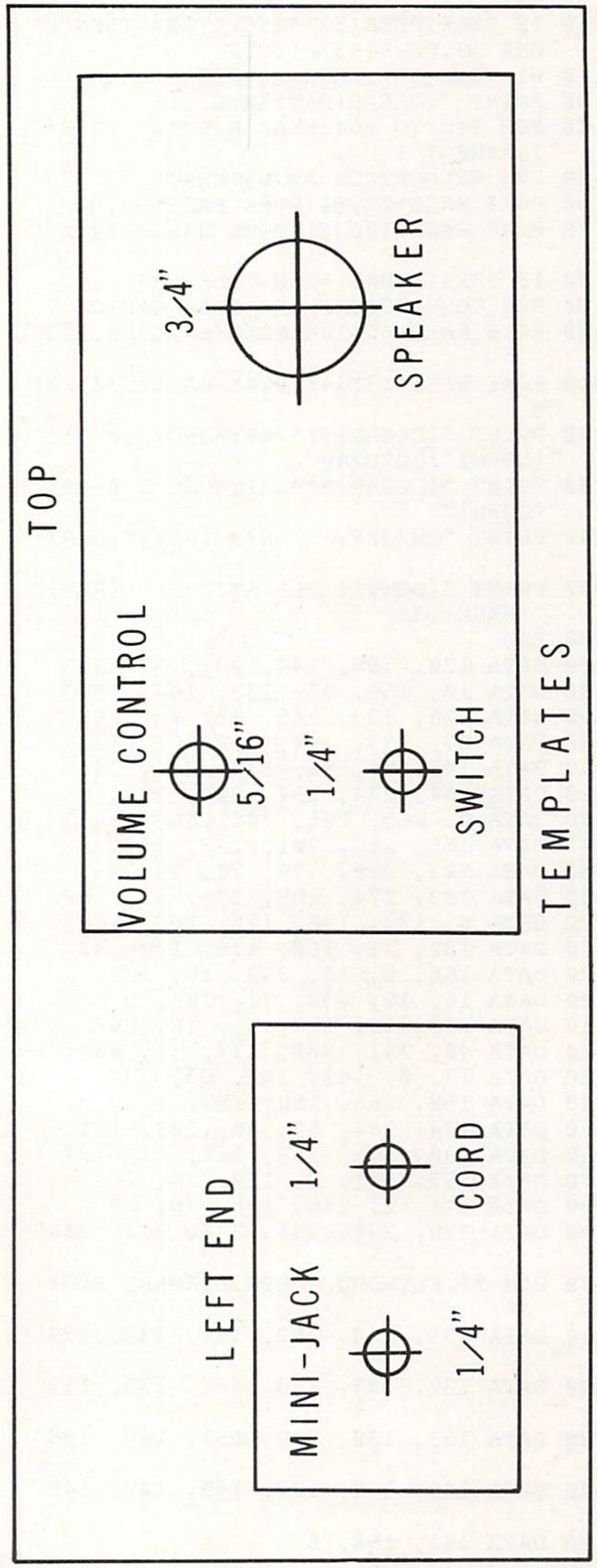
```
        SEC
        ROR ALFALK        ;Enable Alpha-Lock Mode
sets the flag, and
        LSR ALFALK        ;Disable Alpha-Lock Mode
clears the flag.
```

COMPUTE! #15, pg. 99 — the following program was not printed:

```
10000 REM DEFINE A LINE
10010 GRAPHICS 8+16:SETCOLOR 4,5,10
10020 POKE 204,COLUML:POKE 205,COLUMH
10030 FOR I=1600 TO 1636
10040 READ X:POKE I,X:NEXT I
10050 X=USR(1600)
10060 DATA 104,120,166,205,141,10,212,14
1,24,208,200,177,204,141,23,208,200,208,
2,230,205,173,11
10070 DATA 212,201,16,16,4,160,0,134,205
,177,204,24,144,223
```

COMPUTE! #16, pg. 86 — To permit "SHOOT" to run on the new (Revision B) ROMs, change line 201 to: BUF$(589,589) =CHR$(95):BUF$ (590,590)=CHR$(228). To permit it to run on black and white TVs, change line 202 to: BUF$ (97,97)=CHR$(0):BUF$ (98,98)=CHR$(0). The source code correction is: 124B 4C 5F e4 JMP SYS VBV.

COMPUTE! #15, pg. 79 — these templates should have appeared full-size:

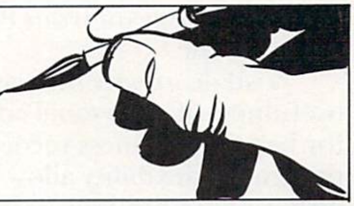


COMPUTE! #15, pg. 120 — Charles Brannon has improved his "Keyword" program to cancel the keyword action in the quote mode and to permit it to be deactivated.

```
100 REM KEYWORD LOADER SEPTEMBER 8, 1~
   ~981
110 IF PEEK(PEEK(53)*256)<>120 THEN P~
   ~OKE 53,PEEK(53)-1:CLR
120 HI=PEEK(53):BASE=HI*256
130 PRINT "{CLEAR}PATIENCE..."
140 FOR I=0 TO 164:READ A:POKE  BASE+~
   ~I,A:NEXT I
150 REM RELOCATION ADJUSTMENTS
160 POKE BASE+22,HI:POKE BASE+58,HI
170 POKE BASE+100,HI:POKE BASE+110,HI~
   ~
180 IF PEEK(50003)=160 THEN 230
190 REM CONVERSIONS FOR 3.0  BASIC
200 POKE BASE+65,146:POKE BASE+69,192~
   ~
210 POKE BASE+131,46:POKE BASE+132,23~
   ~0
220 PRINT "{CLEAR}*** KEYWORD 3.0 ***~
   ~{DOWN}":GOTO240
230 PRINT "{CLEAR}*** KEYWORD 4.0 ***~
   ~{DOWN}"
240 PRINT "ON/OFF:    SYS {REV}";BASE~
   ~
250 PRINT "{DOWN}TABLE AT:       {REV}~
   ~";BASE+138
260 END
270 DATA 120, 165, 144, 201, 37, 208
280 DATA 10, 169, 85, 133, 144, 169
290 DATA 228, 133, 145, 88, 96, 169
300 DATA 37, 133, 144, 169, 63, 133
310 DATA 145, 88, 96, 234, 234, 234
320 DATA 234, 234, 234, 234, 234, 0
330 DATA 0, 165, 205, 208, 89, 234
340 DATA 165, 217, 201, 193, 144, 82
350 DATA 201, 219, 176, 78, 56, 233
360 DATA 193, 170, 189, 138, 63, 162
370 DATA 0, 134, 158, 170, 160, 178
380 DATA 132, 31, 160, 176, 132, 32
390 DATA 160, 0, 10, 240, 16, 202
400 DATA 16, 12, 230, 31, 208, 2
410 DATA 230, 32, 177, 31, 16, 246
420 DATA 48, 241, 200, 177, 31, 48
430 DATA 17, 8, 142, 164, 63, 230
440 DATA 158, 166, 158, 157, 111, 2
450 DATA 174, 164, 63, 40, 208, 234
460 DATA 230, 158, 166, 158, 41, 127
470 DATA 157, 111, 2, 169, 20, 141
480 DATA 111, 2, 230, 158, 76, 85
490 DATA 228, 234, 234, 234, 234, 234~
   ~
500 REM ** KEYWORD TOKEN LIBRARY HERE~
   ~ **
510 DATA 153, 194, 152, 157, 132, 129~
   ~
520 DATA 137, 200, 133, 161, 135, 155~
   ~
530 DATA 163, 130, 159, 151, 160, 138~
   ~
540 DATA 148, 167, 187, 149, 147, 148~
   ~
550 DATA 141, 158, 0
READY.
```

©

New Products



AUTO-CAT ... The Automatic Answer Direct Connect Modem

An auto answer, FCC approved, direct connect 300 baud modem, is now available from Novation, Inc., Tarzana, California.

The Bell 103 compatible AUTO-CAT will communicate at 300 baud over dial-up telephone lines using a standard modular jack. It has three data modes: automatic answer, manual answer, and manual originate. It will operate in either full or half-duplex, and features both local and remote-loopback test functions. The interface between computer and modem is the EIA RS-232.

Pressure sensitive switches on the end of the AUTO-CAT case select its answer or originate functions. LED's give a constant indication of the unit's operational status. The compact 10", 4.7", 1.2" modem uses a separate AC power supply that eliminates heat and voltage hazards.

In addition to communication over standard telephone lines, the AUTO-CAT will also automatically answer each call. This feature offers users the benefits of unattended operation for personal and business computers and terminals. Data can be made available 24 hours-a-day. Using AUTO-CAT, executives can communicate with their office computers in the evenings, on week-ends, even on holidays or vacations, and hobbyists can access their home computers from just about any location with a phone.

The AUTO-CAT is priced at $249.00 for quantity one and is available throughout the country

from authorized Novation distributors, dealers, computer stores, and retail electronic outlets. The AUTO-CAT provides the user with access to data banks, and the ability to swap personal programs with other computer users. The AUTO-CAT is designed to maximize the potential of communications with any computer or terminal.

For further information contact Novation, Inc., 18664 Oxnard St., Tarzana, CA 91356. Telephone 213-996-5060.

TV/Monitor Stand For Atari And Apple II

BYTM Systems, Inc. announces the introduction of a new TV/Monitor stand for ATARI 400/800™ and Apple II™ consoles.

The stand is made of clear polished high-impact acrylic. It securely holds a TV/Monitor up to 15 inches (diagonal) in screen size. Its open design allows maximum air flow around the console and provides easy access to top, sides and rear for cable or ROM cartridge insertion.

The use of the stand reduces workspace crowding while raising the TV/Monitor to eye level.

For ATARI 400/800 owners with ATARI 810™ Disk Drives, it also provides a simple means of minimizing the potential problem identified by ATARI that:

> Placing the ATARI 810 Disk Drive within 12 inches of a television set or monitor causes alteration of the data. Use the disk drive a minimum of 12 inches from the TV or monitor.

The CLEAR STAND raises the TV/Monitor 10 inches above the table level and makes leaving a diagonal space of 12 inches from the disk drive an easy problem to solve.

CLEAR STAND is available for $59.95 from BYTM Systems, Inc., 389 Fifth Avenue (Suite 400), New York, New York 10016.

VisiFile Program Simplifies Electronic Filing

Sunnyvale, CA — September 1, 1981 — File management on a personal computer — record filing, searching, sorting, printing reports and mailing labels — is fast and simple with the new

VisiFile™ program from Personal Software Inc.

VisiFile greatly increases the usefulness of a personal computer for keeping business records. The program's flexibility allows many different applications — inventory, client lists and records, sales information, medical records and other word or numerical data — to be stored, sorted and printed in a variety of formats.

"The 'human interface' of the program makes it extremely easy to use," said Ed Esber, director of merchandising. "Even people who are unfamiliar with computers will be able to master the program and begin to use it right away. Instructions are simple and direct and are selected from an easy-to-understand 'menu' system."

The VisiFile FlexFormat™ feature makes it easy to change, rearrange and add unforeseen information to records, or combine business records into new

files. This means that record keeping can adapt quickly as business information needs change. For example, the change from a five-digit to nine-digit zip code could be handled without rekeying all the data. Users may also creata a "partial file definition" for extremely fast data entry of specific portions of records.

The ability of VisiFile to communicate with other Personal Software products (such as the VisiCalc®, VisiPlot™ and VisiTrend™/VistPlot™ programs) makes it part of an information management software system for personal computers. This ability to exchange data reinforces the power of the total software system; information created or stored by one program need not be rekeyed in order to be used by another.

VisiFile records (and information stored by the other "Visi" products) may be transferred over phone lines by Personal Software's VisiTerm™ program.

Moving Cursor Menu

VisiFile is controlled by a "moving cursor menu" with prompting, similar to the kind of user interface in other Personal Software products. Information entry is simple and allows the user to custom design a "form-like" format on screen for input. Machine language sorting routines and multiple keyed-field indexes assure information retrieval within three seconds.

Mailing labels up to five across and reports may be printed with VisiFile. When ready to print a report, the user may use a simple "row-and-column" format, or develop a custom format area six rows deep by the width of the paper. Elements of each record may be arranged in that area to suit the user. Records may be selected for printing (in addition to being sorted or indexed): for example, all customers from Kansas with orders greater than $10,000 may be selected at the time the report is printed.

All reports can be easily set up, saved with the program. Arithmetic calculations, including column totals, also can be handled in reports.

VisiFile runs on the Apple® II (with language card or Applesoft Basic card) and Apple II Plus personal computers with 48K minimum memory and one disk drive; two disk drives and a printer are recommended. Suggested retail price of the program is $250 (U.S. price only). It will be

available in October.

Atari Sponsors Research Efforts In Education

Sunnyvale, CA — The founding of the Atari Institute for Educational Action Research, an organization that will foster the innovative, yet practical, use of personal computers in education, was announced today by Raymond E. Kassar, chairman and chief executive officer of Atari®, Inc.

The Institute will provide grants of Atari computer products and/or cash stipends to selected institutions, individuals or organizations able to develop and promulgate new uses for computers in education, whether that usage will take place in established institutions, in community programs or in the home.

Dr. Ted M. Kahn, 32, formerly an education consultant to the computer division of Atari, has been named executive director of the institute, which will be located in the company's headquarters here. Kahn has been active in research and development in the use of computers in education for more than ten years.

Grants totaling more than $250,000 in cash and equipment will be given during the Institute's first year of operation. Initially, all support will be for domestic U.S. programs, and not for those from overseas.

"The use of personal computers is fast becoming an integral part of the mainstream of American culture," Kassar said. "We feel we should take an active role in supporting those who are working to push the computer to its full potential as a learning tool. We are looking for applications which will appeal to and serve broad sectors of society."

"Atari has, within the past year, already given major cash and equipment grants to projects at the Lawrence Hall of Science Computer Education Project (University of California at Berkeley); the future center at the Capital Children's Museum (Washington, D.C.); and the I.E.C. Mobile Computer Van (Santa Clara County, co-sponsored by the Industry Education Council and the Computer-Using Educators of California)," Kassar added. "The Institute will give us a formal channel through which to focus our efforts in this area."

One key program is the support of a small number of model schools or alternative learning centers to act as "centers of excellence: to illustrate various uses of computers in education."

For further information contact J. Peter Nelson at Atari, Inc., 1265 Borregas Ave., P.O. Box 427, Sunnyvale, CA 94086.

LO•BALL COMPUTERS

LO•BALL COMPUTE

NECC-82 Call For Participation

The 1982 National Educational Computing Conference (NECC-82) will be held in Kansas City, Missouri, June 28-30, 1982. The purpose of the conference is to provide a broad and rich forum for discussion among individuals interested in educational computing. Based on previous conferences, approximately 1000 people from institutions at all levels are expected to attend.

Authors are invited to submit papers describing actual experiences with computer use in the classroom or the consequences of concrete results or be survey or tutorial papers which include a synthesis and thorough evaluation. Generally, papers that describe projects presented at previous conferences are not considered unless substantial new information can be reported. In this case provide a brief synopsis of the earlier paper clearly indicating the new information. It is expected that most papers will report on specific accomplishment. Papers reporting negative results are also encouraged, especially when the results could have a profound effect in the way educational computing should be viewed. The deadline for submission of papers is Jan. 15, 1982.

Specifications for paper submittal may be obtained from:

Gerald L. Engel
Computer Science Dept.
Christopher Newport College
50 Shoe Lane
Newport News, VA 23606

or from:

E. Michael Staman
NECC-82 General Chairman
Campus Computing Services
University of Missouri-Columbia
305 Jesse Hall
Columbia, MO 65211

Individuals are also invited to submit proposals for special sessions, exhibits, project presentations, and "Birds of a Feather" sessions. For further information write to E. Michael Staman.

New Book For OSI C1P/C4P Users

Los Alamos, NM — TIS, Inc., the company that for the past two years has provided workbooks for the Ohio Scientific C1P computer, announces a new book. "Understanding Your C1P/C4P: A Woorkbook of BASIC Exercises" is a 112 page softcover book that is designed specifically for the Ohio Scientific C1P, SUPERBOARD II, and Challenger C4P. It introduces the fundamentals of OSI BASIC: calculator and program mode, input and output, data representation, and program storage on cassette. The book describes OSI control and logic including testing and branching, subroutine use, and logical operations. It covers character strings and array handling. Understanding Your C1P/C4P contains many exercises and sample programs. It is available from your Ohio Scientific dealer or by writing TIS, Inc., P.O. Box 921, Los Alamos, NM 87544. Price is $7.95 plus $2.00 shipping and handling.

Computer Graphics Spotlighted At National Conference On Visual Communications

National Conference on Computer Graphics: Tools for Productivity, December 7–9, 1981, Washington, D.C. The Conference will spotlight trends in usage and applications of computer graphics, as well as perspectives in emerging services and management needs for visual information. Computer graphics equipment and services demonstrations and corporate and government case histories will be included for an overall look at effectively integrating visual and textual information. Contact U.S. Professional Development Institute, 12611 Davan

Drive, Silver Spring, MD 20904. Telephone: (301) 622-0066.

Software Package For Small Manufacturers

COMPUMAX ASSOCIATES, Inc., of Palo Alto, CA, announces the newest addition to its MicroBiz product line of Accounting Software, with the release of MICROBOMP. A 'bill of manufacturing/materials program' for the microcomputer.

MICROBOMP is a software package specifically for the small manufacturer. This package is designed to handle inventory control, materials requirement planning and it will also 'explode' your bills of material. The program sets-up and maintains three files; 1) a stock file that contains the inventory master information; 2) a bill of materials file that defines the product structures and 3) a schedule file that contains production schedule data.

In addition a data base manager allows the operator to input and update information in these files. The bill of materials processor generates the 'product tree' up to six levels deep. Shortages are flagged at run time. Seperate reports provide stock valuation, ABC analysis, materials requirement listings and a 'MaxMin' report.

The program will, finally, create a journal file automatically, for those using MICROBOMP interactively, either with COMPUMAX's MICROLEDGER program, or with MAXILEDGER, the new Compumax program for divisional accounting.

MICROBOMP is written in Microsoft Basic, requires 48K and retails for $350.00. For further information contact:

COMPUMAX ASSOCIATES, Inc.
P.O. Box 1139
Palo Alto, CA, 94302
415/321-2881 X56

Software Development System For CBM, Apple, Atari

Eastern House Software has announced the release of their unbundled MAE Software Development System for the 2001/4001/8032 Commodore, Apple II 3.2/3.3 DOS, and Atari 400 or 800 computers.

If a software developer has a microcomputer in his laboratory or office, he can transform it into a development system via purchase of the MAE software package.

A main advantage of this software is that it is available for several different microcomputers. When a software developer moves from a PET to an APPLE to an ATARI etc., he need not be concerned with the time consuming task of having to relearn a new set of syntaxes, commands, and other peculiarities because this software works similarly on each of these microcomputers.

This software was designed for ease of use and to aid programmer productivity via its extensive text editing capabilities and numerous other programmer development utilities included in the package.

Since the Assembler/Editor are co-resident, continual manual loading of the editor, then assembler, then editor, etc. is not necessary. The main features include: Macros, Conditional Assembly, Interactive Assembly, up to 31 characters/label, string search and/or replace, plus numerous other editing facilities. Even a word processor is included to aid in the development of program descriptions, manuals, and other text. A number of other utilities are also provided — depending on the version, they include Word Processor, forward/reverse scrolling, tape interface, machine language macro library, library of disk driver subroutines, source of information file, relocating loader, etc.

A cross-assembler version which assembles 6800 source code is available and other cross assemblers are in the works. A detailed spec sheet is available to anyone interested in this software. Price $169.95. Contact:

Eastern House Software
3239 Linda Drive
Winston-Salem, NC 27106

Utility Package For PET/CBM Users

Professional Software Inc. (PSI) of Needham, MA has introduced POWER, a programmer's utility package in a 4K ROM designed for use with Commodore CBM/PET computers.

POWER is for any CBM/PET user who would benefit by reduced program entry time, debugging tools, and easy, quick modifications and updates.

POWER contains a series of new commands and utilities which are added to the BASIC Interpreter. Designed for the user of CBM/PET BASIC, POWER also contains special editing, programming, and debugging features. Included are special keyboard "instant action" features which make up for and go beyond the limitations of CBM/PET BASIC. POWER is sold with complete documentation written by Jim Butterfield of Commodore fame who has been working with preliminary versions of POWER for over a year.

To make POWER even more "user-friendly", PSI has included new "stick-on" keycap labels (which denote POWER's most commonly used features) in every program.

There are currently three (3) versions of POWER, one each for:

40 column CBM/PET with 3.0 BASIC;

40 column (9" or 12") CBM/PET with 4.0 BASIC;

and CBM 8032 (4.0 BASIC) ©

PRICE BREAKTHROUGH

We've done it again! As a leader in small computer sales we must constantly find new ways to save customers money while offering quality products. Our most popular seller, the Apple II Plus, is now available with 64K of user RAM at the unbeatable price of only $1249. This is possible because we manufacture the 16K RAM Card that expands the factory Apple II 48K to it's maximum capacity of 64K. The Ram Card allows use of Integer and Applesoft Basic and other languages like Pascal. It's a must for large data bases, Visicalc, and the Z-80 cards. The card is made from high quality components and has a full one year warranty.

64K APPLE II PLUS*

48 K Apple II Plus with 16K Ram Card

ONLY $1249

FREE SHIPPING*
*on all pre-paid cash orders

16K ONLY **$1025** **48K** ONLY **$1089**

DISK II DRIVE WITH CONTROLLER CARD **$499**

DISK II DRIVE ADD ON **$439**

Apple Cards and Hardware

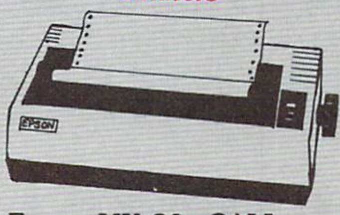
16K Ram Card by CCI	130
Language System w/Pascal	379
Silentype Printer w/Interface Card	349
Hayes Micromodem II	299
Novation Apple-Cat II	339
Videx Videoterm 80 Column w/Graphics	269
Z-80 Softcard by Microsoft	299
16K RamCard by Microsoft	159
ABT Numeric Keypad (old or new kybrd)	110
ALF 3 Voice Music Card	239
ALF 9 Voice Music Card	169
Lazer Lower Case Plus +	55
Micro-Sci Disk Drives (A-40 & A-70)	CALL
SSM AIO Serial/Parallel Card A&T	189
Sup-R-Terminal 80 Column Card	329
SVA ZVX4 Megabyter 8" Disk Controller	589
SVA 2+2 Single Den. 8" Disk Controller	345
ThunderClock Plus	119
Symtec Hi-Res Light Pen	210
Integer or Applesoft Firmware Card	145
Graphics Tablet	619
Parallel Printer Interface Card	135
Hi-Speed Serial Interface Card	135
Smarterm 80 Column Card	299
Joystick by Keyboard Co.	45
Music System (16 Voices)	479
A/D + D/A Interface	289
Expansion Chassis	599
Introl/X-10 Controller Card	169
Clock/Calendar Card	225
CPS Multi-function Card	189
Supertalker SD-200	239
Romplus + Card	135
Romwriter Card	149
Clock/Calendar Module	99
GPIB IEEE-488 Card	249
Asynchronous Serial Interface Card	129
Centronics Parallel Interface Card	99
Arithmetic Processor Card	299

We carry all CCS cards, please call for best prices.

Software for the Apple

Visicalc 3.3	169
CCA Data Management	85
DB Master	169
WordStar (Apple 80 col. version)	299
Applewriter	65
Easywriter	225
Peachtree Business Software	CALL
Visiterm	129
Visitrend/Visiplot	219
Real Estate Analyzer	89
Tax Preparer	99
DOS Toolkit	65
Tax Planner	99
Dow Jones Portfolio Evaluator	45
Dow Jones News & Quotes Reporter	85
Apple Fortran	165
Controller Gen. Bus. System	499

Printers



Epson MX-80 CALL

Anadex 9500/9501 w/2K Buffer	1299
C. Itoh Starwriter 25 CPS	1499
C. Itho Starwriter 45 CPS	1899
Epson MX-70	CALL
Epson MX-80 F/T	CALL
Paper Tiger IDS-445	699
Paper Tiger IDS-460	949
Paper Tiger IDS-560	1249
Silentype w/Apple II interface card	349
Qume Sprint 5/45	2495

TOLL FREE ORDER LINE
800-854-6654
California and outside Continental US
(714) 698-8088
TELEX 695000 BETA CCMO

Video Monitors

Amdek/Leedex Video 100 12" B&W	139
Amdek (Hitachi) 13" Color	359
NEC 12" Green Phospher Screen	CALL
NEC 12" RGB Hi-Res Color	CALL
Panasonic 13" Color	449
Sanyo 9" B&W	185
Sanyo 9" Green Phospher Screen	CALL
Sanyo 12" B&W	269
Sanyo 12" Green Phospher Screen	285
Sanyo 13" Color w/excellent resolution	449

Atari Personal Computer

ATARI 800 16K $749

Atari 400 16K	349
410 Program Recorder	65
810 Disk Drive	449
815 Dual Disk Drive (Dbl. Den.)	1295
822 Thermal Printer	349
850 Interface Module	159
16K Ram Memory Module	149
16K Ram Memory Module (by ConComp)	89

NEC Microcomputer

PC-8001A 32K Ram Computer	CALL
PC-8012A I/O Unit w/32K Ram	CALL
PC-8031A Dual Mini-Disk Drive Unit	CALL

Please write for more information about the NEC computer.

Ordering information: Phone orders using VISA, MASTERCARD, AMERICAN EXPRESS, DINER'S CLUB, CARTE BLANCHE, bank wire transfer, cashier's or certified check, money order, or personal check (allow ten days to clear). Unless prepaid with cash, please add 5% for shipping, handling and insurance. (minimum 5.00). California residents add 6% sales tax. We accept CODs. OEM's, Institutions and corporations please send for a written quotation. All equipment is subject to price change and availability without notice. All equipment is new and complete with manufacturer's warranty (usually 90 days). Showroom prices may differ from mail order prices.

Send Orders to:

consumer computers Mail Order

8314 Parkway Drive
La Mesa, California 92041

Advertiser's Index

The 6502 Resource Magazine
COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**__________
☐ **Don't yet have one...**

☐ 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_____/_____/

10 11 12

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:

10 11 12

The 6502 Resource Magazine
COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service,
Call Our **Toll-Free**
US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**__________
☐ **Don't yet have one...**

☐ 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_____/_____/

10 11 12

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Place
Stamp
Here

COMPUTE! Magazine
Post Office Box 5406
Greensboro, NC 27403

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

FREE NEECO CATALOG!

PLACE STAMP HERE

NEECO
NEW ENGLAND ELECTRONICS CO., INC.
679 Highland Avenue
Needham, MA 02194

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

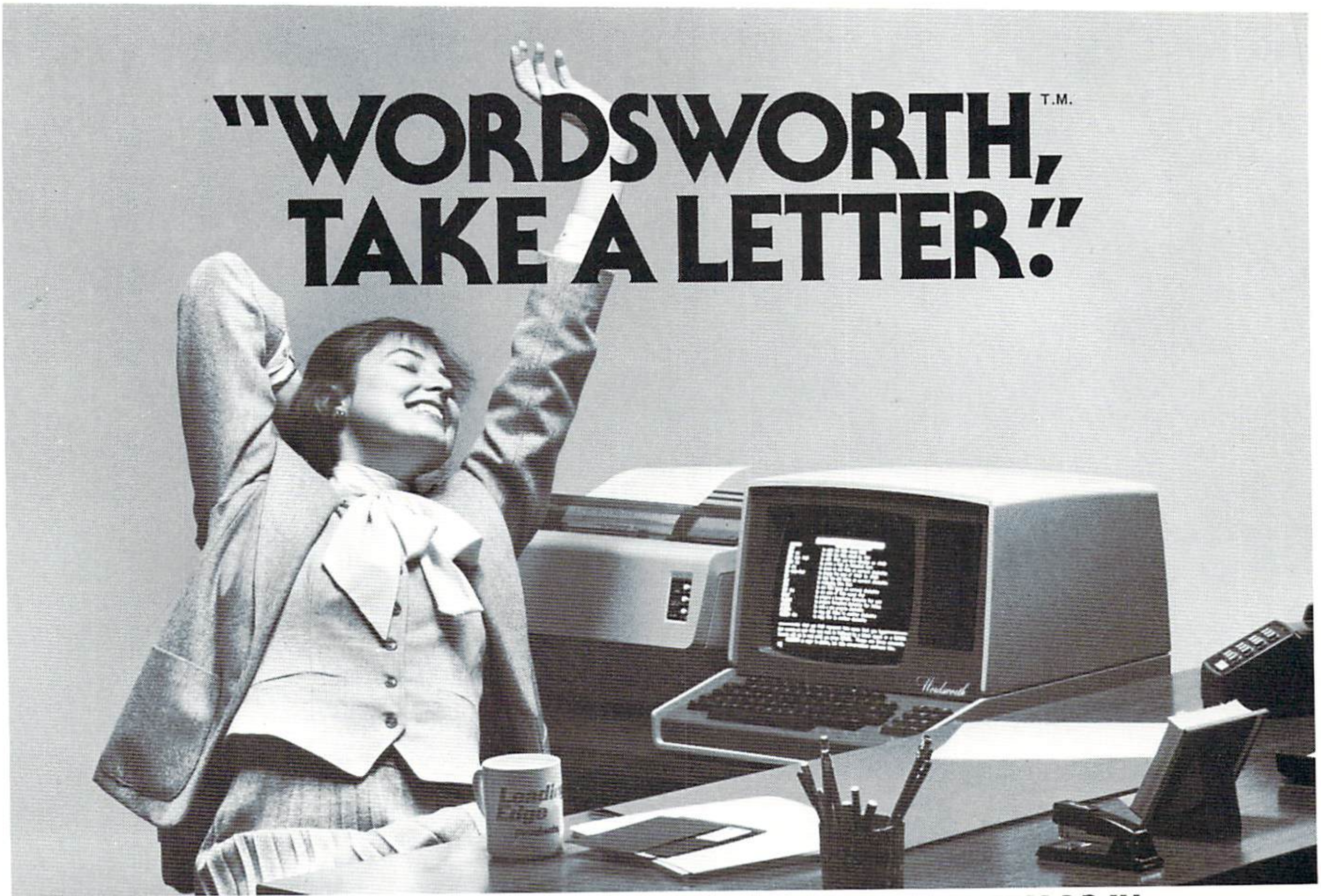
BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Author Feedback:

COMPUTE!



THE FIRST SUPER-SIMPLE LETTER QUALITY WORD PROCESSOR IN THE WORLD THAT CAN SIT NEXT TO YOUR SECRETARY— FOR LESS THAN $5,000.

"Wordsworth™" removes the fear and loathing many people have about things called "computers" and "word processing." Because Wordsworth just sits at a desk and does what he's told. With unique new, simplified software, it can tell its typist what to do—step by simple step—to perform even the most complicated tasks.

It can not only "take a letter," but it can revise it, customize it in millions of ways (quite literally), personalize it, print it on your letterhead or business form so you can't tell it from hand-typed . . . in short, everything but put it in an envelope.

Moreover, it can perform all sorts of mundane business functions, like general ledgers and accounting reports, using easily available industry-compatible CP/M™ software.

Installation? Plug it in. (Cable supplied.)

Size? Bigger than a breadbox—but not by much. It's about half as big, or less, than systems that can't do half as much. Definitely desktop—a sub-compact 20″ by 40″.

Service? Available everywhere.

Price? About $4,995. Complete and ready to go.

For the name of your nearest dealer—and a free hands-on demonstration—just pick up the nearest telephone.

TOLL-FREE 1-800-343-6833.

In Massachusetts, call collect (617) 828-8150. Telex 951-624.

LEADING EDGE.™

Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021
DEALERS: For immediate delivery from the Leading Edge Inventory Bank,™ on "Wordsworth" and other of the industry's most popular products, just give us a call.