A 6502 Version Of The Winter Consumer Electronics Show

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

Clearing Apple II's Low-Resolution Graphics Screen

COMPUTE!

$2.50
March, 1981
Issue 10
Vol. 3, No. 3
63379

The Journal For Progressive Computing™

Designing Your Own Atari Character Set

Machine Language Taking The Plunge

Keyprint (For The Pet) Revisited

Six-Gun Shootout Game For The OSI C1P

0 74470 63379 03

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.

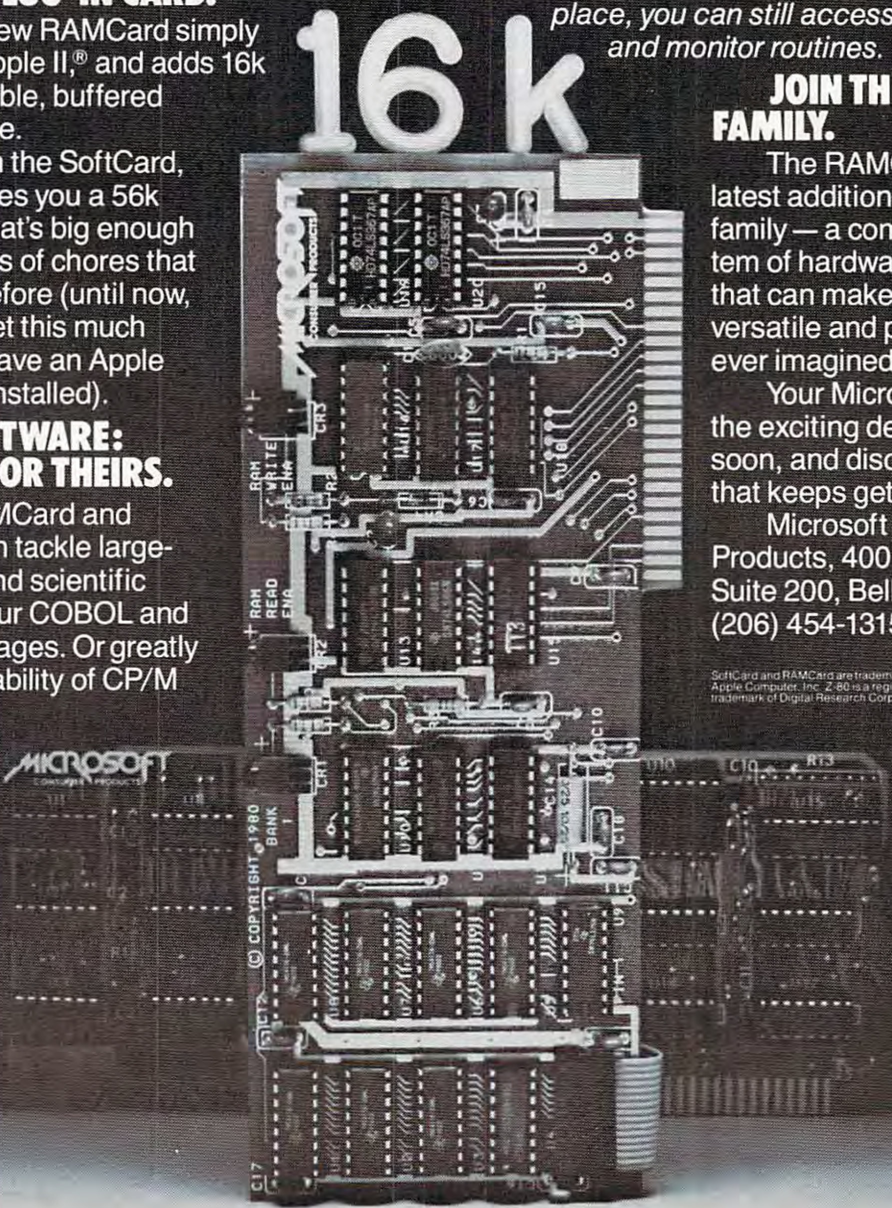
Microsoft's new RAMCard simply plugs into your Apple II,® and adds 16k bytes of dependable, buffered read/write storage.

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.

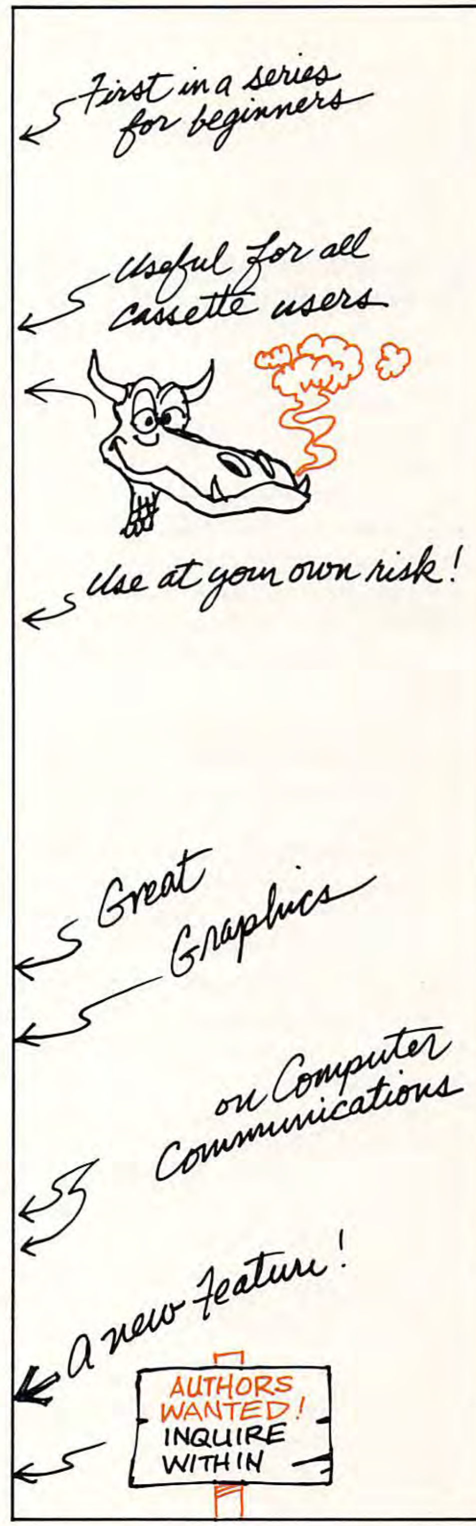
Microsoft Consumer Products, 400 108th Ave. N.E., Suite 200, Bellevue, WA 98004. (206) 454-1315.

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

MICROSOFT

Table of Contents

March, 1981, Vol. 3. No. 3



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

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

 ISSN 0194-357X.

A Small System Services, Inc. Publication

COMPUTE! The Journal For Progressive Computing

The Editor's notes...

Robert Lock,
Editor/Publisher

Software Copying Revisited, Or Who's Paying The Bills?

We recently received an extensive letter from a Canadian subscriber commenting on the January editorial regarding software piracy. I'd like to respond to one specific point of the letter now, and request other readers to respond at length as well. Let's get the dialogue rolling, and we'll continue it from here. I really think it's a critical turning point in the industry's future development. I don't presume to suggest we can solve it here, but I'd like **COMPUTE!** readers to help spark and maintain the dialogue. Here's the prompting remark from the reader:

> *I agree to a great deal with your comment on copyright of magazines and books and software. I disliked your generalizations about schools. Schools are rather like big businesses. If a business has a number of machines and purchases say a "General Package", is it really breaking copyright to make enough copies to be used on all its machines? I don't think so. I reckon schools are in a similar situation...*

First of all, your comment presumes that big business has the right to copy (reproduce) proprietary software. They don't have the right to do so any more than Commodore (for example) would have the right to buy one copy of a program and give it away with all of their computers. Frequently, "big business" is able to buy a piece of software with limited license rights, they pay additional fees. This is not the case with the software we're talking about for our marketplace. I'd like to propose an analogy... suggested by an individual involved in software sales. He commented on a "textbook defense" of software theft, and I'll expand on it here:

Let's assume that you've taken your notes and teaching experiments from the last few years, and spent a recent summer developing them into a textbook, complete with student workbook full of exercises.

You find a publisher, obtain a contract wherein you receive a royalty on sales, go through numerous editings, and finally see your first copy in the mail.

Your book is selling for $19.95, and you're looking forward to some return from your royalties.

Time Passes

Sales are going along okay, but not up to your expectations. I mean it is an *excellent* book. You attend a regional educational conference, and run into Dr. So and So from a neighboring school district. He says, "We really enjoyed your new text, and it's super! We're using it in all of our classes next year."

Great (you think). Hundreds of sales. "Oh, by the way," he continues. "We thought $19.95 was pretty expensive, so we only bought one copy. We're running off class copies in our own print shop... See you later."

There goes your work, and your royalty. I assume you're concerned, if not angry. Is this any different in principle, from the defense of schools and software copying. Now we all know that realistically, the book would be rather expensive to duplicate. And I presume no one would doubt the illegality of the act. The essence of the argument would seem to become one of ease and expense of the copying. Software comes on an inexpensive, easily transportable media. Therefore, does copying it suddenly become okay? We need to give thought as well to the producers of the work. With commercial software, there's someone out there, after some amount of hard work, patiently waiting for their royalty.

I would suggest that users should not define a $19.95 software purchase as an unlimited right to copy, just as they wouldn't consider doing the same with a $19.95 textbook.

It would be more appropriate to approach producers about school licensing agreements, or multiple copies at school discount through their vendors, or whatever. This rational (negotiated) approach, if you will, would solve problems for several parties... Schools who currently buy multiple copies of software would save money, schools (individuals) who currently buy and reproduce would have full vendor support, and so on. The vendor support is an element we haven't touched on, but is part of the whole problem. I've been told, essentially, "Why should I support the educational market? They just steal my stuff."

A Post-Script

These comments should in no way infer that most schools aren't absolutely honest in their software purchases. We're just using this initial analogy to get the ball rolling. We've heard recently of a major industrial research center on the West Coast that's using approximately 80 copies of a $150.00 plus software package... copies cloned from their original single copy purchase. And how many times have you picked up that "back-up" copy of Cursor magazine to use just this once?

Defining The Right Of Back-up

Several other magazines in the industry have recently been running advertisements for a program that copies (duplicates) protected software for a particular

ATARI® PERSONAL COMPUTER SYSTEMS THAT GROW WITH YOU

Start with a better computer. Atari computers have built-in capabilities you can't even add onto many other personal computers. Three programming formats (ROM cartridge, disk and cassette). A 57 key upper/lower case ASCII keyboard with 29 keystroke graphics symbols. 128 colors and hues. Four separate sound channels and a built-in speaker. Four controller ports. A built-in RF modulator and FCC approval for connection to any TV. Plus, nationwide Atari Authorized Service Centers. And more.

Add memory. The ATARI 800™ is supplied with 16K of memory. You can expand up to a full 48K of RAM with 8K or 16K Memory Modules™ you install yourself. In less than a minute. The ATARI 400's™ 8K of RAM may be expanded to 16K at Authorized Service Centers. Both may be expanded to 26K of ROM with slip-in ROM cartridge programs.

Add peripherals. The ATARI 410™ audio-digital program recorder. Single or dual density* disk drives. The ATARI 800 individually addresses up to four drives. Add the ATARI 850™ RS232 Interface Module. Add high speed 40 or 80-column printers. Add an acoustic modem for remote data access. Add a light pen.* And there are more Atari peripherals on the way.

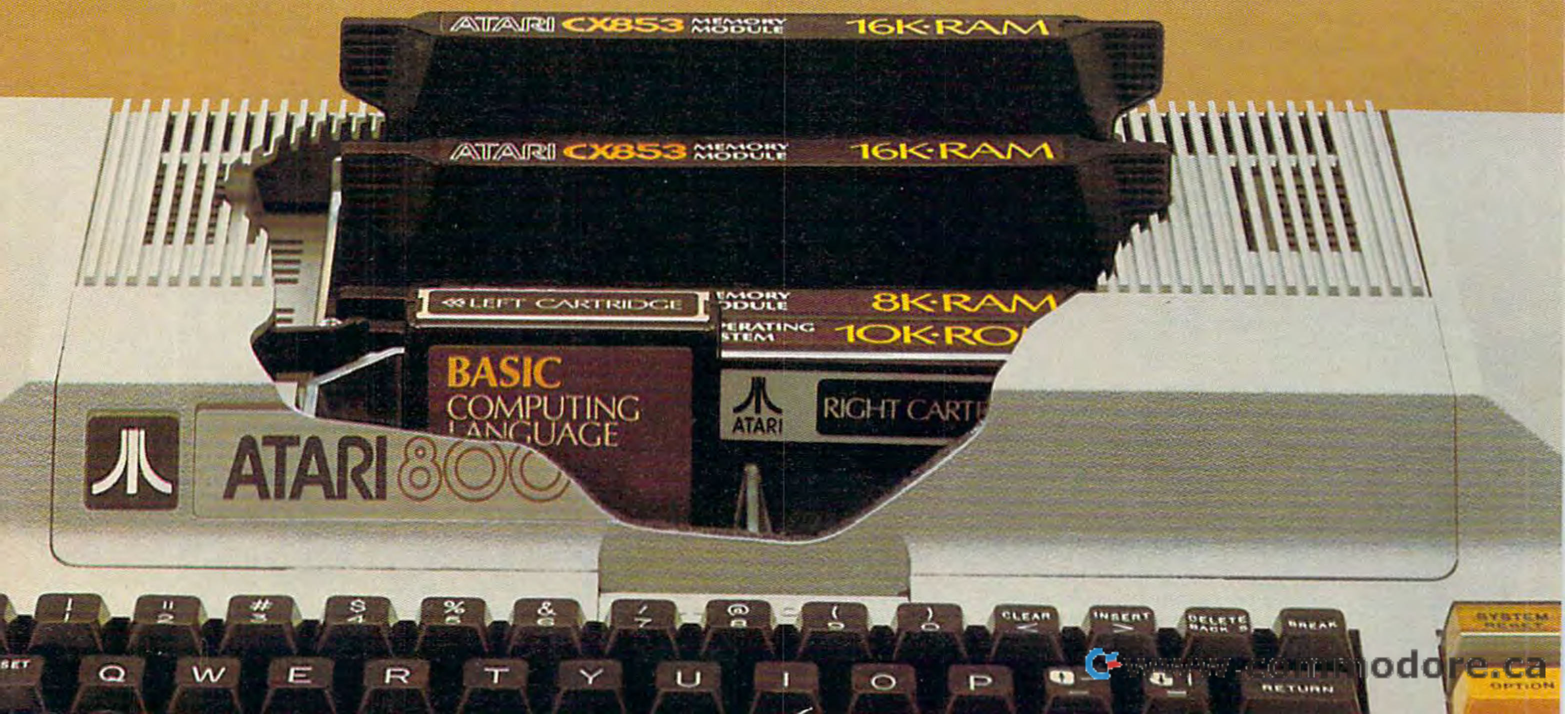
Add programs. Choose among dozens of programs in Atari's rapidly expanding software library. Programs categories include:

- **Personal Finance & Record Keeping**
- **Personal Interest & Development**
- **Professional Applications**
- **Education**
- **Information & Communication**
- **Entertainment**
- **Programming Languages**
- **Small Business Accounting**

Add it up. With Atari, you start with more. And you can build to more. Because Atari offers you personal computer systems that grow with you. Ask your Atari retailer to give you a full demonstration of Atari computers, peripherals and programs. Complete systems. Because when other people were thinking hardware and software, Atari was thinking systems.

*Available Fall, 1980

ATARI®
PERSONAL COMPUTERS
1265 Borregas Avenue, Sunnyvale, CA 94086
Call toll-free (800) 538-8547 (Except Alaska and Hawaii)
(In California: (800) 672-1404) for the name of your nearest Atari retailer.



Subscription Information

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

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

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

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

Advertising Sales

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

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

Northern California,
Pacific Northwest, Rocky Mountain States,
Texas, Oklahoma
Jules E. Thompson, Inc.
1290 Howard Avenue, #303
Burlingame, CA 94010
415 348-8222
Phoebe Thompson

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

New England,
New York State
The Gittelman Company
Statler Office Building
Suite 482
20 Providence Street
Boston, MA 02110
617 451-0822
Joan Donahue

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

The Gittelman Company
Summit Office Centre
7266 Summit Avenue
Fort Washington, PA 19034
215 646-5700
Doug Johnson

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

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

Telephone: (919) 275-9809

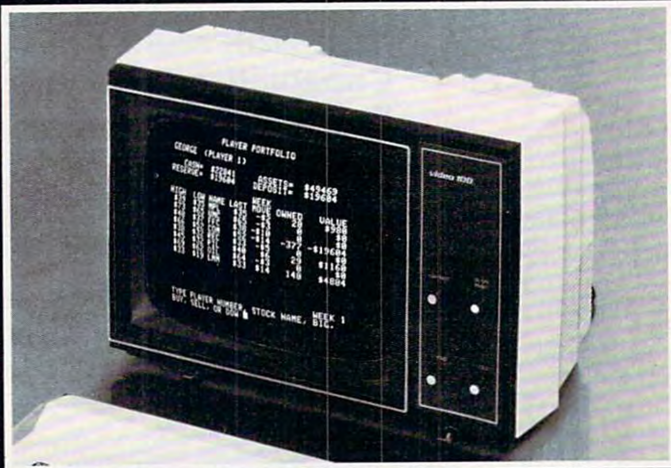
AMDEK

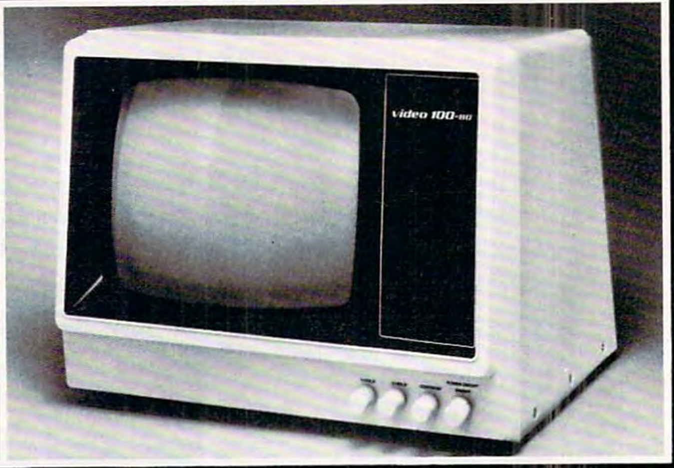
Professional Color & B/W Monitors

NEW 13″ COLOR

The Amdek 13″ Color Video Monitor is ideal for all personal and business computing applications. Bright jitter-free text allows viewing over extended periods without causing eye fatigue, especially in word processing applications. The low resolution display provides 40 characters wide by 24 characters deep with 260 horizontal lines and 300 vertical lines.

This compact, lightweight monitor (only 25 lbs.) has a molded-in carrying handle, making it easy to move. Its industrial grade construction and shatter resistant case provide reliable service in both office and industrial environments. All front mounted controls make it easily adjustable at a glance without the loss of a single keystroke.





VIDEO 100 12″ B/W MONITOR

This highly reliable 12″ black and white monitor features a 12MHz band width and 80 character by 24 line display. Plug-in compatibility with Apple, Atari, Radio Shack, O.S.I., Micro-Term and Exidy make this the perfect text display for almost any system.

VIDEO 100-80 12″ B/W MONITOR

The model 80 features an industrial grade metal cabinet with built-in disk mounting capability and space for an 11″ x 14″ PC board for custom designed electronics. Front mounted controls include power, contrast, horizontal hold, vertical hold, and brightness.

Amdek Corp. • 2420 E. Oakton Street, Suite "E" • Arlington Heights, IL 60005 • (312) 364-1180 TLX: 25-4786

Amdek Corp. the new name for **Leedex**... Expanding for the future, growing with success.

machine. One of the magazines (Micro) checked with us to see our feelings on such advertising. We indicated that we wouldn't run such ads, given that the software in question could be used to produce duplicate copies of "protected" and proprietary software. We understand that Micro has since decided to do the same.

We applaud this move toward protecting the rights of the software industry, and encourage additional comment.

The Rights Of The Buyer

This isn't intended to be an inclusive comment on the rights of software buyers, but an attempt to open discussion. I already know of one **COMPUTE!** Associate Editor who has definite feelings on the subject. We'll try to get him involved in the discussion on these pages.

The buyer of a copy-protected diskette has potential problems, and deserves to be protected as well. Vendors of software would remove the legitimate need for copying software if they would adopt a customer oriented, fully responsive plan for allowing licensed owners of software to quickly, conveniently and most of all, economically, obtain a back-up in the case of failure of a diskette.

The principal of repetitive back-ups is rightly embedded in the history of data processing technique, and can't be ignored in an industry rush to protect proprietary software from duplication. Personal Software appears to be trying to deal with this with Visicalc, and I'm sure other vendors are as well. The vendor has the right to protect proprietary software. The vendor also has the obligation to protect the customer's interest. The extent of this protection of the mutual interests of both will ultimately help define the existence of the protection violation market, and the strength of the software market as a whole. We'll eagerly await your comments.

The Boston Commodore Show; VIC Meets The Consumer

Judging from the interest in VIC at Commodore's Boston Show (February 6, 7), the $299.95 color computer entry from Commodore will be well received. One of the biggest unresolved problems of the moment is: "When will production be up and running?" They're currently saying March-April, and I think they'll make it. BASIC, by the way, is the well-known upgrade ROM set, and color animation capabilities are rather nice. We hope that Commodore will see their way to a nice introductory package price. Watch our April issue for a full review of VIC, and mid-April for the availability of our brand new quarterly magazine, exclusively for the VIC. It's called Home and Educational COMPUTING!, and is announced elsewhere in this issue.

Atari Update

They're still selling as fast as they can make them. What else can we say? ©

A Beginners Guide To COMPUTE!

If you're just getting started with your computer or with **COMPUTE!**, here are several notes to help you use **COMPUTE!**:

Presentation

In every issue we try to present a balanced group of articles ranging from material for beginners to material for old hands. Frequently, a beginner can get a great deal out of an advanced article, even though much of it may be over his or her head.

Program listings are presented as legibly as possible. Pet programs are generally reproduced and reformatted here where we've developed software to "translate" the special Pet graphics characters into characters printable by our equipment. These are explained below:

Program Listings for COMPUTE

Cursor control characters will appear in source listings as shown below:

```
h=HOME         , ĥ=CLEAR SCREEN
↓=DOWN CURSOR  , ↑=UP CURSOR
→=RIGHT CURSOR, ←=LEFT CURSOR
r=REVERSE      , r̂=REVERSE OFF
```

Graphics (i.e. shifted) characters will appear as the unshifted alphanumeric character with an underline. This does not apply to the cursor control characters. The Spinwriter thimble doesn't have a backarrow symbol, so a "~" is used instead.

The "¬" is used to indicate the beginning of a continuation line. It is also used to indicate the end of a line which ends with a space. This prevents any spaces from being hidden.

If, for example, you're an Apple owner using a Pet program that's reproduced in this fashion, you'll need to be familiar with these special characters so you can program around them. As more computers implement versions of MicroSoft BASIC, the programs should become more and more transportable. ©

Reader's Feedback *will return next issue in a revised and expanded format. Keep those letters and editor's feedback cards coming. By the way, see our New Products section, new this issue. RCL*

A 6502 Version Of The Winter Consumer Electronics Show: January, 1981

David D. Thornburg
Innovision
P.O. Box 1317
Los Altos, CA 94022

At a time when most normal folks are taking down the holiday decorations, and preparing for the new year, those of us who haunt the trade shows were anxiously preparing for our January fix — the Winter CES (Consumer Electronics Show) held in Las Vegas. Unlike specialized trade shows, like Comdex, the CES has exhibits covering almost all consumer products that are likely to contain silicon. Because of the continuing recession, only 55,000 people attended this show which was held in the Las Vegas Convention Center and in two nearby hotels. Rather than describe some of the more novel products, such as the talking microwave oven (with, would you believe it, a Japanese-English accent), or the solar rechargeable flashlight (look, I couldn't make this stuff up if I tried, so believe me!), I decided to mention some of the products of greater relevance to **COMPUTE!** readers: the 6502-based microcomputers which were displayed.

As far as hardware is concerned, the big hit of the show was the **Commodore VIC-20** (your fearless scribe is preparing a review of this machine to appear in a forthcoming issue of **COMPUTE!**). At a suggested retail price of $299, it is apparent to me that Commodore has the technical ability to give the Radio Shack Color Computer a solid run for its money. In fact, I expect VIC sales to place Commodore firmly in the number two spot for total machine placements, and perhaps to even edge up on our Texas friends. The styling is beautiful and the price is right, but even more importantly, Commodore is going all out to support the cottage industry that has kept the PET well supplied with software. Watch for the FCC approval, followed by the VIC showing up in your corner computer store sometime in March or April.

Software had its day at CES also. **Atari** showed both their **new PILOT cartridge** (see this month's Computers and Society column), and also showed their **word processor package.** Both pieces of software are very well done, and should do much to help Atari on its accelerating growth curve. While I didn't see any new Atari hardware on the floor, there were rumors of some nice new things hidden away in their hospitality suite. It is apparent that Atari is in this game for the long haul.

Those of you who are waiting for the keyboard portion of the **Mattel Intellivision** (complete with a 6502-based computer with 16 K of RAM and a Microsoft BASIC) will have to wait a little longer. Once again they say that deliveries will begin in March — only the year has changed.

The **absence of Apple and OSI from this show** was noted. Apple has apparently decided to focus its efforts in the small business market, and leave the home computer market to fend for itself for awhile. I saw lots of Apple folks at the show, however, so they can't have totally lost interest in the consumer market. OSI, on the other hand, has had nice exhibits at this show for quite some time, and I have to assume (without checking it out) that their absence was due more to their recent acquisition than to any plans they have to depart from the low-end market.

For those of us who have invested in 6502-based systems, it is heartening to see that this processor continues to be among the most popular.

The **6502-based hand held computer from Matsushita** (which will be marketed both through Panasonic and Quasar) was shown running a communications interface hooked up to the Source. It appears that the software for this computer is almost finished, and that we can expect to see it hit the market in a few months. Considering that this computer, with modem and coupler, will have a retail price in the $1,000 range, I find its small keyboard and one-line liquid crystal display to be annoying. On the other hand, if you want the ultimate in portability, this might be just the computer for you.

It was interesting to note that all the other computers at the CES (excepting the TI 99/4) used either the 8080 or Z-80 microprocessor. There were no new 16-bit computers introduced. For those of us who have invested in 6502-based systems, it is heartening to see that this processor continues to be among the most popular. It is clear that our investment will retain its value for quite some time.

The Beginner's Page

Robert Lock, Editor/Publisher

COMPUTE! is a specialized resource magazine that provides editorial coverage of a family of microcomputers that share a common "central processing" chip — the 6502. With the proliferation of small computers, we find that you can obtain more useful and relevant information in a magazine that doesn't (for example) cover the TRS-80 computer line. It, by the way, uses a different microprocessor chip: the Z-80. A whole different family of chips, and a different family of machines.

Mapping COMPUTE!

We're organized sequentially, with the front section of the magazine devoted to material of general interest and utility. The following sections are devoted to specific machines. These are called Gazettes, and contain information pertinent to your special machine. Frequently, however, you'll find useful material in other sections (Gazettes) of the magazine.

Where The Sets Merge, Or Common Interests

All of your machines have available a programming language called BASIC. **BASIC** stands for **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. It's what is called a higher-level language. BASIC differs somewhat on the different machines, so you'll find some things in common with other BASIC's and some differences. In practice, BASIC simply makes it easier for you to talk to your 6502 microprocessor. The 6502 sits at the heart of your machine. It and supporting "firmware" and "hardware" make your machine capable of doing what it does. In a nutshell, you feed your 6502 instructions that it can act on. It only acts on very picky little sets of numbers, and that's why you have BASIC. BASIC serves as an English-language like translator for you, taking your instructions in a pseudo-English format and translating them for the 6502.

Other portions of firmware contribute to your machine's features. Someone has already written a 6502 program that resides in your machine, interacting with BASIC, and passing out instructions to the 6502. This program (or set of programs) is called the **Operating System**. It is permanently inscribed onto a chip or set of chips inside your machine. Its permanence implies its generic name: firmware. Hardware characteristics also help define your machine and its capabilities. Memory is one important characteristic. Memory is your working space. It's much like the work area you have on top of your table. Let's assume you have a table of x size, and you're ready to start a project. You have a set of notes and instructions previously developed to help you with your task. Let's also assume that you may not write on your previous notes, but you may refer to and use them freely. These notes and instructions are much like one kind of memory in your computer: **ROM. R**ead **O**nly **M**emory is memory that has a program already saved on it. The program doesn't go away when you turn your machine on and off. This, then, is your firmware: the ROM, or set of ROMs, where "permanent" programs, or sets of instructions, reside. The number of instructions your Operating System can have, e.g. in part its complexity, is then directly related to the amount of ROM in your machine. Let's stack up these beginning "notes" on one side of your desktop. We'll assume that they take up one-quarter of the table space. We can use them always, but we can't stack anything else there.

Now let's add a clean notepad to our workspace. We'll call this our work area. Everytime we come back to our desk to work, we'll assume we have x amount of clean space to put our notepad on, and do whatever work we want. This "flexible" work area equates to **RAM. R**andom **A**ccess **M**emory is another type of memory inside your computer. Unlike ROM, which has a set of instructions "built-in", RAM is empty. It's your working area; the space you use for putting in your own information. After you put something in it, you can use it, reading from it later, and so on, just like your note pad. And like your notepad, you can continue to return to it, reading from it and writing to it. Until, of course, you turn your computer off. Your RAM is wiped clean when the power goes away.

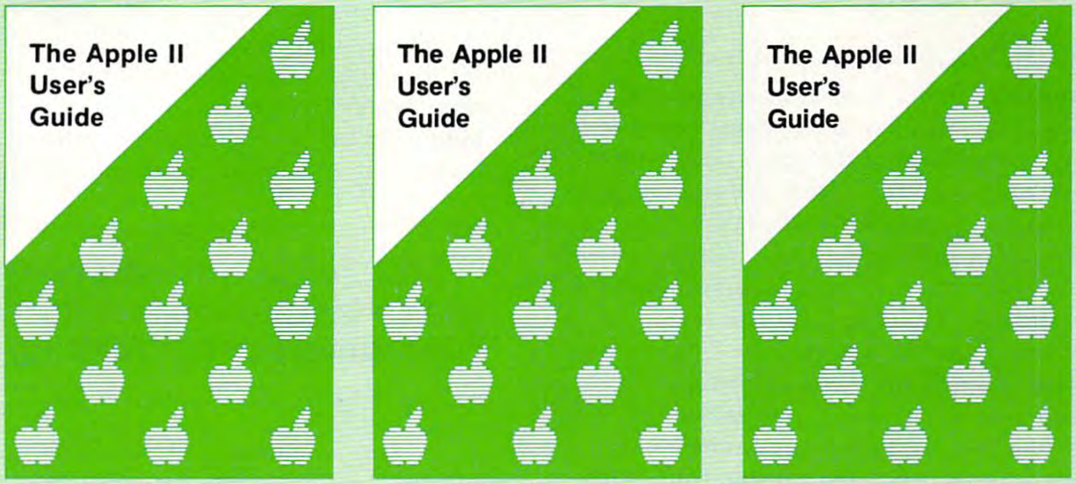
Storage Media

This is why your computer has a storage device. Whether it's a tape cassette or a disk drive, it's there to save the contents of your RAM. Let's assume you've entered a set of instructions. We know where they are: they're in RAM. We also know that once the machine is turned off, these instructions will be erased from RAM. The solution is simple. Save them somewhere. When you want them back, your Operating System will take care of reading them back into RAM.

Where Are We?

We've just covered the areas that make your small computer a truly remarkable and powerful device. It has a set of predefined features, controlled by permanent instructions, and aided by design features that always make it "act" the same way. Your calculator is much like this; and your microwave oven; and the host of other consumer products that are utilizing some form of microprocessor chip. The firmware in these devices is simply a set of permanent instructions to a computer chip. But your computer has a whole lot more. It has RAM, allowing you work space to develop your own sets of instructions to your computer chip. Aha! Tremendous flexibility. And it has a storage device, allowing you to save the effort of your labors, retrieving them at will for use and further development.

Osborne Polishes THE APPLE



and Publishes THE GUIDE

The Apple II User's Guide by Lon Poole, Martin McNiff, and Steven Cook #46-2, $15. ☐

This Guide is the key to unlocking the full power of your Apple II or Apple II plus computer. The Apple II User's Guide brings together in one place a wealth of information for Apple computer users. It will tell you more about your Apple than any other single source.

Topics include: **Applesoft and Integer BASIC programming** - especially how to make the best use of Apple's sound, color and graphics capabilities. The book presents a thorough description of every BASIC statement, command and function. **Machine level programming** - although not a machine language programming guide, this book covers the Machine Language Monitor in detail. **Hardware features** - the disk drive and printer are covered in separate chapters. **Advanced programming** - special sections describe high resolution graphics techniques and other advanced applications.

This book will save you both time and effort. No longer will you have to search endlessly for useful information. It's all here, in the Apple II User's Guide, thoughtfully organized and easy to use.

*Apple is a trademark of the Apple Computer Corporation.

More Osborne Books for Apple Users

6502 Assembly Language Programming
by Lance Leventhal
Here's the book that shows you how to program the Apple in assembly language. "With its numerous reference tables, explanations and descriptions, this book is a major accomplishment. It's by far the most complete 6502 book to appear . . . Make space on your bookshelf. You'll want to keep this one handy."
COMPUTE
Book #27-6 $16.99, 606 pp. ☐

Some Common BASIC Programs
by L. Poole and M. Borchers
Written in a subset of standard BASIC that is compatible with many microcomputers, this book contains 76 useful and easy-to-use programs. The book explains how each program works, shows sample runs, and has program listings with remarks. "If you want to do mathematics or statistical programming on your computer, this book is a must." *ON COMPUTING* Magazine
Book #06-3, $14.99, 200 pp. ☐

Practical Basic Programs
edited by Lon Poole
Here's a collection of 40 programs you can easily key in and use on most microcomputers. *Practical Basic Programs* is especially useful in small business applications. It solves problems in finance, management, and statistics. The book contains sample runs, practical problems, BASIC source listings, and an easy-to-follow narrative to help you realize the potential uses of each program.
Book #38-1, $15.99, 200 pp. ☐

An Introduction to Microcomputers Volume I - Basic Concepts Second Edition
by Adam Osborne
The world's best selling textbook on microcomputers develops a detailed picture of what a microcomputer can do, how it does what it does, and how it can be used in any practical environment. This is the most comprehensive and up-to-date introduction to microprocessor systems available anywhere.
Book #34-9, $12.99, 320 pp. ☐

Make check payable to: OSBORNE/McGraw-Hill
630 Bancroft Way, Berkeley, CA 94710 **Dept. G4**

Phone Orders: (415) 548-2805 or Toll Free 800-227-2895

Name ______________________________

Address ______________________________

City/State/Zip ______________________________

Plus ☐ .75/item 4th class ☐ $1.25/item UPS ☐ $2.50/item Air Mail ☐ $4.00/item Overseas (California residents add applicable tax.)

Total Amount Enclosed $ __________ or charge my ☐ VISA ☐ Mastercharge Card # __________ Exp. Date _____

Signature ______________________________ ☐ Please send me your free catalog.

Computers and Society

David D. Thornburg
Innovision
P.O. Box 1317, Los Altos, CA 94022

As R. Buckminster Fuller is fond of pointing out, synergy is the behavior of whole systems which is not predicted by the behavior of the parts taken separately. There have been two recent developments in the personal computer world which, taken together, have the promise of true synergism. These events are the publication of Seymour Papert's long-awaited book: **Mindstorms - Children, Computers, and Powerful Ideas** (Basic Books), and the introduction of the language PILOT for the Atari computers.

Since this year's theme is communications, it is only appropriate that we spend some time looking at the communication between the user and the computer. While the mechanical means through which this communication takes place are worthy of extensive discussion, I want to concentrate this month on the nature of the language through which we interact with computers, since this also is an area of intense importance.

One might argue that there is little reason for concern with computer languages at this point, since we all have access to fairly powerful versions of BASIC on our computers. We might all agree that BASIC is not terribly hard to learn, and that there are lots of very exciting BASIC programs, and even that BASIC has become the *de facto* standard computer language for personal computers.

But even with the tremendous penetration of BASIC in the marketplace, I have yet to find any serious computer user (regardless of age) who really likes it. At the primitive level of programming at which we all start, BASIC works pretty well. But as we get more sophisticated, most of us find ourselves writing code that we can't understand two weeks after we write it.

Of course, there are detractors of BASIC who feel that languages like PASCAL are the solution. I must confess that I find PASCAL lacking in that it doesn't encourage the user to sit down and start writing some small part of a program, to play with the bits and pieces, and to then bring everything together later on. This is one area in which BASIC excels. For those who feel that people should be organized when they write a program, PASCAL (and C and other "serious" languages) may very well be the best choice. But what about the new computer user who wants to build a highly interactive program, or the child who wants to explore concepts in geometry through the experience of programming rather than through playing a pre-defined "game" or sitting at a "canned" CAI lesson? These people need languages which are interactive, highly flexible, extraordinarily powerful, and are easy to get started with.

LOGO And PILOT Are Two Such Languages.

While LOGO (as this is being written) does not yet exist on commercial personal computers, it has been the subject of an extensive research program at MIT for more than a decade. Much of the research has been devoted less to the development of computer languages *per se*, than to the development of a computer assisted learning environment for children. The goals, aspirations and results of this work are the subject of Papert's **Mindstorms.** It is hard to imagine any person who is intensely concerned with the use of computers by children who would fail to be moved by the sweeping vision implicit in Papert's work. Writing from the perspective of a mathemetician who spent much time with Jean Paiget, Papert presents a variation on the Paigetian model of the "child as builder" in that he sees the need for children to have an abundance of materials with which to build things.

...for some educators, Computer Aided Instruction has come to mean "computers programming children".

That the computer can be one such building tool is the cornerstone of the many computer literacy activities we see springing up around the world. But, for some educators, Computer Aided Instruction (CAI) has come to mean "computers programming children". There is much to be gained from reversing this process — and that is where the need arises for an exceptionally powerful (and easy to learn) computer language.

LOGO is a highly interactive language which contains a graphics environment containing something called a "Turtle". The Turtle is a non-Euclidian point (having both position and direction, rather than a position alone). The programmer can send messages to the Turtle which cause pictures to be drawn on the display screen. Those of you who are familiar with the Milton Bradley **Big Trak** are already familiar with this idea. To draw a square on the screen, for example, a child working in the Turtle microworld might type:

```
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
```

As each instruction is executed, the Turtle first moves forward by 100 units, and then turns right by 90 degrees, drawing its path on the screen as it moves. The desired square thus takes shape on the screen. A programmer wishing to use squares quite frequently might wish to extend the repetoire of commands the Turtle uses by defining a new procedure which the Turtle then "understands":

...children are asked to find the bug by "playing Turtle".

```
TO SQUARE
REPEAT 4
  FORWARD 100
  RIGHT 90
END
```

If squares of arbitrary size are required, one might write:

```
TO SQUARE :SIZE
REPEAT 4
  FORWARD :SIZE
  RIGHT 90
END
```

and then, anytime a square is desired, one would type, for example,

```
SQUARE 47
```

to draw a square with each side 47 units long.

The value of using the Turtle environment in an interactive way is expressed by Papert this way:

> *Working in Turtle microworlds is a model for what it is to get to know an idea the way you know a person. Students who work in these environments certainly do discover facts, make propositional generalizations, and learn skills. But the primary learning experience is not one of memorizing facts or of practicing skills. Rather, it is getting to know the Turtle, exploring what a Turtle can and cannot do. It is similar to the child's every day activities, such as making mudpies and testing the limits of parental authority — all of which have a component of "getting to know".*

One of the more valuable experiences for children involved with computers is learning how to "debug" a program with errors in it. Traditionally, we are taught that errors are bad. Papert shows that, by accepting the inevitability of errors in programs, children can learn to analyze the results of the error and then learn to avoid the error in the future, and to "patch" it in the meantime.

In order to make the debugging process as meaningful as possible, children are asked to find the bug by "playing Turtle". The child then walks around the floor, executing each instruction in turn until the "faulty" instruction is found. But doesn't this method for finding errors lead the child to "thinking like the computer"? Papert sees this experience in a larger context. He says:

> *In my experience, the fact that I ask myself to "think like a computer" does not close off other epistemologies. It simply opens new ways for approaching thinking. The cultural assimilation of the computer presence will give rise to computer literacy. This phrase is often taken as meaning knowing how to program, or knowing about the varied uses made of computers. But true computer literacy is not just knowing how to make use of computers and computational ideas. It is knowing when it is appropriate to do so.*

While Papert is quite heartened by the growth of the personal computer industry, since this growth will

He likens BASIC to the QWERTY layout on the keyboard—an artifact from a time when better things didn't exist.

result in children having ever easier access to computers, he is frustrated by the limitations of these machines and by the extremely strong penetration of BASIC into the marketplace. He likens BASIC to the QWERTY layout on the keyboard — an artifact from a time when better ways didn't exist. But what of LOGO itself? This language will not be forever locked in the University laboratory. Versions for the TI 99/4 and the Apple computer will probably come into general availability soon.

Even if LOGO, with all its power, doesn't make its appearance in the marketplace soon, I feel that many of Papert's ideas can be implemented today on the small computers whose capabilities he dislikes, through the medium of the language PILOT.

As normally written, PILOT interpreters allow the user to create spectacular text manipulation programs (*c.f.*, the article by Thornburg and Thornburg in the first issue of **COMPUTE!**). Recent embellishments have made PILOT a splendid language to use on computers with high quality graphics environments, such as the Atari 400 and 800. Those of us who use Atari computers can, with Atari PILOT, do many of the things Papert does with LOGO.

Those of you who are familiar with PILOT probably think of it as a language best suited for creating text-based learning materials. My view of the language is far more open than that, because it is so easy to teach to youngsters. It has long been my dream to see the superb text manipulative power of PILOT extended to give the user similar power to create pictures. The Atari PILOT is the answer to this dream since it contains a graphics package that is, in some ways, very similar to the Turtle graphics of LOGO.

For example, an Atari computer user running PILOT might draw a square this way:

```
GR: CLEAR
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
GR: DRAW 25
GR: TURN 90
```

As each instruction is carried out, the square begins to take shape on the screen. If the user wants to draw lots of squares PILOT allows one to create a "module" as a deferred program. By typing AUTO at the command level, and then typing:

```
*SQUARE
  GR: 4(DRAW 25; TURN 90)
E:
```

A module (SQUARE) is created. On leaving the AUTO mode (the AUTO mode automatically places line numbers in front of each statement, thus keeping them from being executed immediately), the user can draw a square by typing:

```
U: *SQUARE
```

in which **U:** is the USE operator found in all versions of PILOT.

My reasons for giving this particular example are two-fold. First, it shows the similarity between the Turtle graphics of Atari PILOT and that of LOGO. Secondly, it shows that PILOT can be used in an interactive mode which combines deferred program segments (modules) with immediate execution of commands.

Can (or should) PILOT replace BASIC? I can only answer by saying that, on the Atari computers, it has for me. I find the language much easier to learn, much easier to use, and capable of doing anything I have ever wanted to do. One of its major features (especially important when working with children) is that a PILOT program can be read by someone other than its author. This is rarely the case for large BASIC programs.

Finally, while most users will want to use PILOT to write self contained programs, I am very happy with the fact that the Atari implementation of PILOT allows the user to interact with the language without having to write "finished" programs. As Papert has shown, the value of "playing around" with an interactive language can be great for all users, and especially for children. ©

Editor's Note: Atari PILOT is not expected to be available until late spring. Check with your dealer for more information. RCL

Editor's Note: Although Richard refers to the PET in this overview, I recommend this article to all who've expressed interest in machine language. RCL

Taking the Plunge— Machine Language Programming for Beginners

Richard Mansfield
Philipsburg, PA

If you have been using BASIC for a while now, you can probably go in and out of STR$ and VAL and there is no mystery to ON GOTO anymore. In fact, the only strange BASIC statements at this point are USR, SYS, and PEEK and POKE. They are gateways into Machine Language and that is still an unknown area. Take heart. It is said that people who first learned programming using Machine Language (M.L.) can find BASIC confusing.

In this article I will discuss aspects of M.L. programming which were unclear or difficult for me when I went on to learn M.L. after a fairly complete grasp of BASIC. I had seen "assembler listings" in magazines with their usual warning that the numbers must be entered *exactly* or the program could not work. And the numbers themselves were in HEX—7 and 10 was 0A! It seemed difficult. It really isn't that hard (but try to explain to a non-computerist that BASIC isn't that hard).

The first thing to do is to get a good book on 6502 (our computers' CPU chip) programming. There are five or so, but among the best are *"Programming the 6502"* by R. Zaks (Sybex) and *"6502 Assembly Language Programming"* by Lance Leventhal (Osborne). You can ignore such information as signed binary, floating point, octal, hardware and input-output chapters. What you want to learn is the meaning of hexadecimal and binary — two new ways to express numbers.

"Machine Language" means that you are entering statements in exactly the way that your 6502 processor will see them. By contrast, a BASIC statement such as LOAD represents hundreds of M.L. statements which have already been programmed by somebody at Microsoft and frozen into ROM chips inside the computer. When the computer (always scanning and waiting for carriage returns) finds that you have typed LOAD, it has a list of addresses and chooses the one associated with LOAD and jumps (JMP it's called) to the address in ROM where the proper sequence of M.L. operations is set down. This sequence is like a subroutine. And BASIC itself is nothing more than a huge web of thousands of M.L. subroutines. In the PET, for example, if you want to jump to the subroutine that sends the number in the 6502's "accumulator" (defined below) to your screen, you type SYS 65490 and the computer is thrown into its M.L. mode and told to start doing the task which begins at the 65490th memory cell in it's brain. That is near the top. There are maps of the computer's memory cells.

A Simple Map of the PET's Brain

0 to 1023—RAM (you can change it's contents), but used by BASIC to store addresses (called **pointers**), temporary data (such as what you type in from the keyboard, called **input buffer**), temporary data of its own in a **stack**, and all manner of reminders to itself about whether or not the tape recorder is on, etc., (called **flags**). So, if you tamper with these memory cells, you might confuse the computer enough to send it into an **endless loop** within itself and you cannot communicate with it again until you turn off the power and force it to **reset** (get itself together).

1024 to 32767—your RAM to use for BASIC programs, or M.L. programs which you put together. Unlike ROM, these cells can each contain any number from 0 to 255. ROM is frozen with its various numbers carved in forever. All PETs start their RAM here, but if you have 8k then you can only use RAM up 8000 cells from 1024.

32768 to 33791—the cells of your screen (40 column screens).

36864 to 45055—space for you to add new ROM chips such as Toolkit.

45056 to 65535—BASIC itself, along with the computer's instructions about interrupting itself (if you should press STOP, for example), how to run the T.V. (**CRT** or **monitor**), how to talk to the peripherals (**I/O**), and other housekeeping chores (called the **operating system**).

Far more detailed maps are available to tell you exactly where things happen inside. See back issues of COMPUTE! for Jim Butterfield's exhaustive maps for PET (issues #1, #6), APPLE (issue #2), and others (issue #2).

QUALITY DISK SOFTWARE

BACKED BY ON-GOING APPLICATIONS SUPPORT

from SPECTRUM SOFTWARE

APPLE II Ⓐ *TRS•80* Ⓣ

HOME FINANCE PAK I: Entire Series $49.95 Ⓐ Ⓣ

CHECK REGISTER AND BUDGET: This comprehensive CHECKING ACCOUNT MANAGEMENT SYSTEM not only keeps complete records, it also gives you the analysis and control tools you need to actively manage your account. The system provides routines for BUDGETING INCOME AND EXPENSE, AUTOMATIC CHECK SEARCH, and BANK STATEMENT RECONCILING, CRT or printer reports are produced for ACTUAL EXPENSE vs BUDGET, CHECK SEARCH DISPLAY RECONCILIATION REPORT and CHECK REGISTER DISPLAY by month. Check entry is prompted by user-defined menus of standard purposes and recipient codes, speeding data entry and reducing disk storage and retrieval time. Six fields of data are stored for each check: amount, check no., date, purpose, recipient and TAX DEDUCTIBLE REMINDER. CHECK SEARCH routines allow searching on any of these data fields. Routines are also provided for CHECK SORT by date and check no., DATA EDITING and Report Formats. Up to 100 checks/mo. storage $39.95

SAVINGS: Account management system for up to 20 separate Savings accounts. Organizes, files and displays deposits, withdrawals and interest earned for each account. Complete records shown via CRT or printer $14.95

CREDIT CARD: Get Control of your credit cards with this program. Organizes, stores and displays purchases, payments and service charges for up to 20 separate cards. Use for credit cards or bank loans. CRT or printer reports $14.95

UNIVERSAL COMPUTING MACHINE: $49.95 Ⓐ

A user programmable computing system structured around a 50 row x 50 column table. User defines row and column names and equations forming a unique computing machine. Table elements can be multiplied, divided, subtracted or added to any other element. User can define repeated functions common to row or column greatly simplifying table setup. Hundreds of unique computing machines can be defined, used and stored, and recalled, with or without old data, for later use. Excellent for sales forecasts, engineering design analysis, budgets, inventory lists, income statements, production planning, project cost estimates-in short for any planning, analysis or reporting problem that can by solved with a table. Unique curser commands allow you to move to any element, change its value and immediately see the effect on other table values. Entire table can be printed by machine pages (user-defined 3-5 columns) on a 40 column printer.

COLOR CALENDAR: $29.95 Ⓐ

Got a busy calendar? Organize it with Color Calendar. Whether it's birthdays, appointments, business meetings or a regular office schedule, this program is the perfect way to schedule your activities.
The calendar display is a beautiful HI-RES color graphics calendar of the selected month with each scheduled day highlighted in color. Using the daily schedule, you can review any day of the month and schedule an event or activity in any one of 20 time slots from 8:00 A.M. to 5:30 P.M. Your description can be up to 20 characters long. The system will also print out hard copies on your minimum 40-column printer.

BUSINESS SOFTWARE: Entire Series $159.95 Ⓐ Ⓣ

MICROACCOUNTANT: The ideal accounting system for small businesses. Based on classic T-accounts and double-entry booking, this efficient program provides a ledger journal for recording posting and reviewing up to 1,000 transactions per month to any one of 300 accounts. The program produces CRT and printer reports covering:

Transaction Journal	Balance Sheet
Accounts Ledgers	Income and Expense Statement

Includes a short primer on Financial Accounting. Requires 48K Ram $49.95

UNIVERSAL BUSINESS MACHINE: This program is designed to SIMPLIFY and SAVE TIME for the serious businessman who must periodically Analyze, Plan and Estimate. The program was created using our Universal Computing Machine and it is programmed to provide the following planning and forecasting tools.

CASH FLOW ANALYSIS	SALES FORECASTER
PROFORMA PROFIT & LOSS	SOURCE AND USE OF FUNDS
PROFORMA BALANCE SHEET	JOB COST ESTIMATOR
REAL ESTATE INVESTMENT	INVENTORY ANALYSIS

Price, including a copy of the Universal Computing Machine $89.95

BUSINESS CHECK REGISTER AND BUDGET: Our Check Register and Budget programs expanded to include up to 50 budgetable items and up to 400 checks per month. Includes bank statement reconciling and automatic check search (48K) $49.95

ELECTRONICS SERIES: Entire Series $259.95

LOGIC SIMULATOR: SAVE TIME AND MONEY. Simulate your digital logic circuits before you build them. CMOS, TTL, or whatever, if it's digital logic, this program can handle it. The program is an interactive, menu driven, full-fledged logic simulator capable of simulating the bit-time response of a logic network to user-specified input patterns. It will handle up to 1000 gates, including NANDS, NORS, INVERTERS, FLIP-FLOPS, SHIFT REGISTERS, COUNTERS and user-defined MACROS. UP to 40 user-defined random, or binary input patterns. Simulation results displayed on CRT or printer. Accepts network descriptions from keyboard or from LOGIC DESIGNER for simulation .... $159.95 Ⓐ Ⓣ

LOGIC DESIGNER: Interactive HI-RES Graphics program for designing digital logic system. A menu driven series of keyboard commands allow you to draw directly on the screen up to 15 different gate types, including 10 gate shape patterns supplied with the program and 5 reserved for user specification. Standard patterns supplied are NAND, NOR, INVERTER, EX-OR, T-FLOP, JK-FLOP, D-FLOP, RS-FLOP, 4 BIT COUNTER and N-BIT SHIFT REGISTER. User interconnects gates just as you would normally draw using line graphics commands. Network descriptions for LOGIC SIMULATOR generated simultaneously with the CRT diagram being drawn. $159.95 Ⓐ

MANUAL AND DEMO DISK: Instruction Manual and demo disk illustrating capabilities of both programs $29.95

MATHEMATICS SERIES: Entire Series $49.95 Ⓐ

STATISTICAL ANALYSIS I: This menu driven program performs LINEAR REGRESSION analysis, determines the mean, standard deviation and plots the frequency distribution of user-supplied data sets. Printer, Disk, I/O routines $19.95

NUMERICAL ANALYSIS: HI-RES 2-Dimensional plot of any function. Automatic scaling. At your option, the program will plot the function, plot the INTEGRAL, plot the DERIVATIVE, determine the ROOTS, MAXIMA, MINIMA, INTEGRAL VALUE $19.95

MATRIX: A general purpose, menu driven program for determining the INVERSE and DETERMINANT of any matrix, as well as the SOLUTION to any set of SIMULTANEOUS LINEAR EQUATIONS. $19.95

3-D SURFACE PLOTTER: Explore the ELEGANCE and BEAUTY of MATHEMATICS by creating HI-RES PLOTS of 3-dimensional surfaces from any 3-variable equation. Disk save and recall routines for plots. Menu driven to vary surface parameters. Hidden line or transparent plotting ... $19.95

ACTION ADVENTURE GAMES: Entire Series $29.95 Ⓐ

RED BARON: Can you outfly the RED BARON? This fast action game simulates a machine-gun DOGFIGHT between your WORLD WAR I BI-PLANE and the baron's. You can LOOP, DIVE, BANK or CLIMB-and so can the BARON. In HI-RES graphics. $14.95

BATTLE OF MIDWAY: You are in command of the U.S.S. HORNETS' DIVE-BOMBER squadron. Your targets are the Aircraft carriers, Akagi, Soryu and Kaga. You must fly your way through ZEROS and AA FIRE to make your DIVE-BOMB run. In HI-RES graphics $14.95

FREE CATALOG-All programs are supplied on disk and run on Apple II w/Disk & Applesoft ROM Card & TRS-80 Level II and require 32K RAM unless otherwise noted. Detailed instructions included. Orders shipped within 5 days. Card users include card number. Add $1.50 postage and handling with each order. California residents add 6½% sales tax. Foreign orders add $5.00 postage and handling.

SPECTRUM SOFTWARE
142 Carlow, P.O. Box 2084
Sunnyvale, CA 94087

FOR PHONE ORDERS: (408) 738-4387
DEALER INQUIRIES INVITED

VISA master charge

The Monitor and the Three Kinds of Numbers

It is important in learning M.L. programming to grasp a distinction between the three ways that the computer could see any number. Depending on the *context*, it will think that a given number is either a datum, an address, or instruction (a task it should perform, such as fetching something). To illustrate this distinction, we can construct a simple, but very common, M.L. routine using the PET M.L. Monitor. (If you have another computer, the addresses where you locate this experiment and the address of your screen RAM might differ, but all our 6502-based machines use the same M.L. instruction set). To enter the monitor, we must SYS to any address in the PET which contains a zero. There is always a zero at address 1024 so we can type SYS 1024 and the PET will display the "registers" and the cursor will land beside a dot, indicating that the monitor is available for our commands.

Let's ignore the register, and simply notice that the fourth number listed is under an "ac" which means that, at this time, the "accumulator" in the 6502 chip contains this number. For a long time, I wondered which addresses in the PET contained the accumulator, the x register and the y register. They are actually unique and not part of the RAM or ROM memory as such. These registers are stopping places for data as it streams from one place to another, from one actual memory cell to another. But on to the experiment.

We will put the letter "A" on the screen. Following the dot, type:

.m 0360 370 (this asks for the numbers between these hex addresses)

Then, when the numbers appear, type in these new numbers right over the ones on the screen:

.0360 A9 41 8D 00 80 00 (we have written a complete action for PET with this, so hit the return key to let the monitor enter these new hex numbers in place of the old ones). If your monitor types a "?" then you have made an error where the "?" appears on the line. Try again.

What have we got here? When the PET is told to start with the instruction A9 it will load the next number in our sequence into the accumulator. That will be the 41. Then it looks at the 8D which tells it that the following two numbers (00 80) are the address to store what is in the accumulator so it puts the 41 at address 8000 (which is hex for the first cell on the screen — and an "A" will appear there. How did 00 80 get changed into 8000? You just have to get used to it. An address is read into the computer LSB (least significant byte) first, followed by the MSB (most significant byte).

The last number we entered was the 00. This is hex for 0, and it is called a "break" which was the way we got into the monitor with our SYS 1024. In this case, when finshed printing our "A", it will come upon our break instruction. Now type: .g 0360 (which means go to 3060 and do what it says there). The "A" will appear and the monitor will come back on showing its registers. Notice what is in the "ac". You can print any other letter by increasing the value where the 41 is. To return to the BASIC mode, type an "x".

This example, so simple, is just how the complex tasks are performed by the computer — one thing at a time (but fast). Organize enough of these segments and you have BASIC, or FORTRAN, or any other "higher" language. Look at the two 00's we used. They represent two different types of numbers which are context-defined in the computer. Since the first 00 followed an instruction (8D) which said put the "A" here, the computer knew that this 00 was the less important part of an address and the next number it found would be the MSB of that same address. Having finished that job, it asks, what next? The next number can *never be an address or a datum.* It must be an instruction to the computer, so the 00 in this position is the instruction "break." A number can only be either a datum, an address, or an instruction. Of these three possibilities, the computer knows how to interpret a number by the "syntax" (the relative position to other numbers in the sequence). This is exactly how we know what someone means when they say "TOO TOO" on the phone, as in "My little girl is two today."

So, our "41" can translate three ways: datum — the actual number (or what that number means in a code, "A" in the ASCII standard translation system); 2. address — the 41st (65th in decimal) address cell in the computer; 3. instruction — please exclusive — or the number located in the cell pointed to by the address in the first 256 bytes as offset by the x register. (Before you are alarmed, there is very little chance that you will ever use this particular instruction with this addressing mode in your entire life.)

M.L. or Assembly Language

What we have just done is the most elemental level of coding next to flipping switches for each bit in each byte. We have entered our code a byte at a time using hex humbers. But this is slow. And, since numbers are so abstract, they are hard to remember. The term "mnemonic" means "memory helper" and this is the next step up. Simple toggle switch or hex programming is usually called "machine level" or "machine language" programming. If you use a three-letter mnemonic instead of A9 to help you remember that this loads the accumulator, things will be easier. LDA means load the "ac", BRK stands for break, and STA means store "ac", and so forth. There are 55 mnemonics, one for each task that our 6502 can perform. However, some of them are so rarely used that you can easily copy down the main ones and learn the strange ones later if you want. Most everything can be done with about 20 of them.

Of course, the computer will not understand

LDA. It only reads numbers, so you will use a program which lets you enter the LDA, but pokes A9 (actually it hands the PET a decimal number and BASIC takes it down to the binary level for you.) The program which gives BASIC a translation of your LDA is called an **assembler,** hence assembly language or assembly programming. The terms M.L. and assembly language are used interchangeably, though, and both refer to an entry of code in the same way that the computer will later follow it, byte by byte.

Using the M.L. Routines from BASIC

In many cases, you can use a routine in the BASIC code by finding its starting address with a map and then examining it with a disassembler (a program which looks at the raw numbers and translates them back into mnemonics). Then you can just JMP (jump) or JSR (jump to subroutine) from your M.L. program directly into BASIC's M.L. code.

If you are programming in BASIC itself, life is simple, but execution of your program can be too slow. To use our example, if you wanted to print an "A" from BASIC you would type! PRINT "HA" and the computer would put a 41 into the accumulator and jump to 65490 where an all-purpose routine for outputting a byte is located in the BASIC ROM. You could also do this with an M.L. routine by typing: 0360 A9 41 20 D2 FF 00 (The 20 is JSR and the FFD2 is hex for 65490).

But this, too, is slow. After landing in the BASIC ROMs, the first thing that PET does is a jump to another address where it determines that you mean to send the "A" to the screen and not to the tape or a printer. Then it flys down to a "vector" at address 00B0 which is rather like a corner shot in pool — when it gets here it just picks up another address and goes back up into ROM memory pretty close to where it jumped from. And so on. Since BASIC must do all kinds of jobs, it is more general than any routine you program in M.L. yourself. It has to check many parameters before acting to send your "A" to the screen. So, often, you will want to save time and code in M.L. yourself. Using routines from the BASIC ROM also requires that you know what these routines expect as preconditions. That output routine will print whatever is in the accumulator, so you must have loaded it already with the character wanted.

To give another example, you can print a large decimal number to the screen (as in scorekeeping during a game) by a JSR to CF83 (in BASIC 4.0), but you must already have placed the LSB of that number in the X register and the MSB in the accumulator. If you want to experiment with this, go into the monitor and when the registers show on the screen, type over the number in the "ac" and the number in the "xr" with the MSB & LSB of the number you want to have printed. Typing return will change these registers. Then: .0360 20 83 CF 00 .g 0360

What you are doing here is entering BASIC where it prints line numbers on the screen during a LIST, but you are going in and out of that particular area without using any other aspect of that code. Trying to set up this sort of printout would be unnecessary and time-consuming if you tried to do it yourself. So, in this case, we are happy to "borrow" some already programmed M.L. routines from the BASIC ROM.

M.L. or BASIC—Which is best?

Often, BASIC is best. It is easier to program and easier to debug (fix errors). Whole tasks can be left to the computer which you would have to carefully program in M.L. code. And BASIC uses a language which is crypto-English. At least for the beginner, PEEK is easier to relate to than LDA.

M.L. code, when you RUN a program, will often enter never-never-land — an endless loop which you cannot get out of without turning off the computer and destroying the program. There are "warm" resets which you can add to the PET which will exit such a loop and leave your program intact. One helpful technique is to fill the memory area that you are coding with zeros before you start. Then, if something goes awry, you might land on a zero which, as a BRK instruction, will safely send the

NEECO announces the MIPLOT Plotter from WATANABE

MIPLOT WX4671

	Code(ASCII)	Name	Function
Vector commands	D	DRAW	Draw a straight line to the point specified by absolute coordinates.
	I	RELATIVE DRAW	Draw a straight line to the point specified by relative coordinates.
	M	MOVE	Move with pen up to the point specified by absolute coordinates.
	R	RELATIVE MOVE	Move with pen up to the point specified by relative coordinates.
	L	LINE TYPE	Specify solid or broken line.
	B	LINE SCALE	Specify the pitch of a broken line (0.1 – 12.7mm).
	X	AXIS	Draw X or Y coordinate axis.
	H	HOME	Return to the origin with the pen up.
Character commands	S	ALPHA SCALE	Specify character size (1 to 16 times basic 0.7mm x 0.4mm)
	Q	ALPHA ROTATE	Specify character orientation. (Four directions)
	P	PRINT	Draw ASCII code characters.
	N	MARK	Draw mark centered on the pen position. (Six kinds)

$1250.00*

*Includes interface to CBM, Atari®, Apple, or TRS•80**. Please specify.

- 26 cm X 36 cm Plotting area
- .1 mm Addressing Resolution
- Full ASCII in any of 15 different character sizes
- Uses standard felt tip pens

Please CALL or WRITE for specifications and information.

®Atari is a registered trademark
**TRS•80 is a registered trademark of Tandy Corp.

ORIGINAL 8K PET 2001* OWNERS TAKE NOTE!

The following peripherals and accessories are IN STOCK AT NEECO:

1. AXIOM PRINTER

- Complete PET graphics
- Plug compatible
- Electrostatic paper
- 40 or 80 columns

$299.00

2. 16 or 24K EXPANDAMEM

INTERNAL MEMORY EXPANSION UNIT

- Plug compatible
- Dynamic low heat memory
- Proven reliability
- No adaptor needed

16K - $299.00
24K - $379.00

3. FULL SIZE KEYBOARD

- Complete PET graphics
- Separate keypad
- Plug compatible
- With cover

$99.95

*8K-2001 with original keyboard and built-in cassette

NEECO
679 HIGHLAND AVE.
NEEDHAM, MA 02194

NEW ENGLAND ELECTRONICS CO., INC.
"NEW ENGLAND's Largest Computer Showroom"

(617) 449-1760
MASTERCHARGE OR VISA ACCEPTED
TELEX NUMBER 951021 NEECO
MON-FRI. 9:00-5:30. E.S.T.

Computer Communications Experiments

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO

I. Introduction

This article describes a RS-232C interface circuit for serial input/output that can be used with any computer peripheral that uses such an interface. In this instance, the peripheral is a modem (NOVATION CAT) that can be used to transmit and receive data over telephone lines. Many modems require a RS-232C interface, hence the need for the circuit which in turn uses a 6551 ACIA (Asynchronous Communication Interface Adapter). The purpose of publishing this work is to find one or more persons who would be willing to experiment with computer communications over telephone lines. The article also describes some very simple software that can be used with a modem to transmit and receive messages over telephone lines. Later, more sophisticated load and dump routines can be written to transfer large amounts of data and/or programs from one user to another.

A true confession is that I am a beginner in the area of computer communications, and I would like to try some simple experiments before I fork-up a big subscription fee to one of the networks, only to find that my equipment or my understanding is inadequate. If you can obtain the necessary equipment and if you are in roughly the same position, write me a letter when you have said equipment operating and we will try to arrange a time to try our hardware and software on a telephone link. I might add that the software and hardware described here have *not* been tested, except in the "TEST" mode on my modem, in which case everything worked properly. I am aware that my routines are simple and slow, and I would welcome suggestions for improvement.

II. Circuits And Things

I sometimes wonder if there are any hardware enthusiasts like myself out there. You might let your editor/publisher know of your interests. My hardware fan club seems to be the null set, judging from the amount of mail I get. But here is another circuit even if no one ever builds it. You can always buy an RS-232C interface anyway. The circuit is shown in Figure 1. It consists of three integrated circuits, a 6551 ACIA, a MC1489 RS-232 line receiver, and a MC1483 RS-232 line driver. The latter two circuits change the RS-232 signal levels to TTL levels, and TTL level signals to RS-232 signal levels, respectively. The 6551 ACIA operates at TTL levels (5 volts is logic one, zero volts is logic zero), while the modem operates at RS-232C signal levels (see Michael E. Day's RS232 Communications in COMPUTE!, September/October 1980, page 26). The power connections for the MC1488 and 1489 devices are given in Table 1.

Table 1. Power Connections for the RS-232 line driver and line receiver.

MC1488	**Pin 1 = −12V**	**Pin 14 = +12V**	**Pin 7 = GND**
MC1489	**Pin 14 = +5V**	**Pin 7 = GND**	

The connections to the left of the 6551 ACIA are made to the user's microcomputer system. Most of the signals are standard 6502 system bus signals, and require no explanation. Thus, address lines AØ and A1 are used to address the four registers of the 6551, and are connected to the register-select pins RSØ and RS1. (You will probably want to obtain a specification sheet from either Rockwell or Synertek when you get your 6551; in fact, I advise you not to build the circuit without a spec sheet.) The data bus connections are shown in Figure 1, as well as several of the 6502 control lines (R/W, $\emptyset_2$, $\overline{\text{RES}}$, $\overline{\text{IRQ}}$). A 1.8432 MHz crystal is also required. Still referring to the connections on the left-hand side of the 6551 in Figure 1, we are left with pins CSØ and CS1, the chip selects.

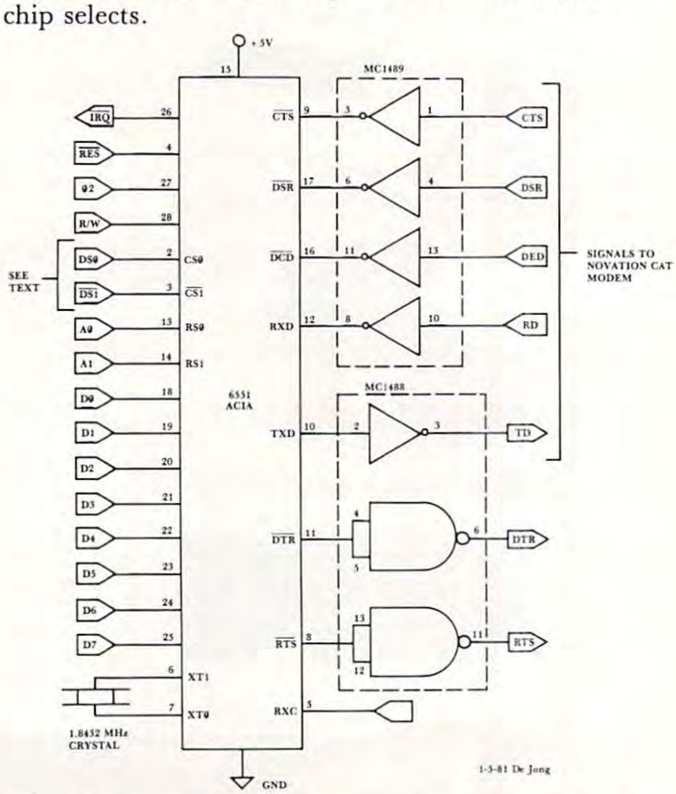


Figure 1. Circuit for the 6502-to-RS-232C-to-Modem interface.

DYNACOMP

Quality software for⁺:

ATARI	TRS-80 (Level II)*
PET	NORTH STAR
APPLE II Plus	CP/M 8" Disk

GAMES, SIMULATIONS, EDUCATION and MISCELLANEOUS

BRIDGE 2.0 (Available for all computers) — **Price: $17.95 Cassette/$21.95 Diskette**
An all-inclusive version of this most popular of card games. This program both BIDS and PLAYS either contract or duplicate bridge. Depending on the contract, your computer opponents will either play the offense OR defense. If you bid too high, the computer will double your contract! BRIDGE 2.0 provides challenging entertainment for advanced players and is an excellent learning tool for the bridge novice.

HEARTS 1.5 (Available for all computers) — **Price: $14.95 Cassette/$18.95 Diskette**
An exciting and entertaining computer version of this popular card game. Hearts is a trick-oriented game in which the purpose is not to take any hearts or the queen of spades. Play against two computer opponents who are armed with hard-to-beat playing strategies.

STUD POKER (ATARI only) — **Price: $11.95 Cassette/$15.95 Diskette**
This is the classic gambler's card game. The computer deals the cards one at a time and you (and the computer) bet on what you see. The computer does not cheat and *usually* bets the odds. However, it sometimes bluffs! Also included is a five card draw poker betting practice program. This package will run on a 16K ATARI. Color, graphics, sound.

POKER PARTY (Available for all computers) — **Price: $17.95 Cassette/$21.95 Diskette**
POKER PARTY is a draw poker simulation based on the book, POKER, by Oswald Jacoby. This is the most comprehensive version available for microcomputers. The party consists of yourself and six other (computer) players. Each of these players (you will get to know them) has a different personality in the form of a varying propensity to bluff or fold under pressure. Practice with POKER PARTY before going to that expensive game tonight! Apple Cassette and diskette versions require a 32 K (or larger) Apple II.

VALDEZ (Available for all computers) — **Price: $14.95 Cassette/$18.95 Diskette**
A simulation of supertanker navigation in the Prince William Sound and Valdez Narrows. The program uses an extensive 256X256 element radar map and employs physical models of ship response and tidal patterns. Chart your own course through ship and iceberg traffic. Any standard terminal may be used for display.

FLIGHT SIMULATOR (Available for all computers) — **Price: $17.95 Cassette/$21.95 Diskette**
A realistic and extensive mathematical simulation of take-off, flight and landing. The program utilizes aerodynamic equations and the characteristics of a real airfoil. You can practice instrument approaches and navigation using radials and compass headings. The more advanced flyer can also perform loops, half-rolls and similar aerobatic maneuvers.

CRIBBAGE 2.0 (TRS-80 only) — **Price: $14.95 Cassette/$18.95 Diskette**
This is a well-designed and nicely executed two-handed version of the classic card game, cribbage. It is an excellent program for the cribbage player in search of a worthy opponent as well as the beginner wishing to learn the game, in particular the scoring and jargon. The standard cribbage score board is continually shown at the top of the display (utilizing the TRS-80's graphics capabilities), with the cards shown underneath. The computer automatically scores and also announces the points using the traditional phrases.

CHESS MASTER (North Star and TRS-80 only) — **Price: $19.95 Cassette/$23.95 Diskette**
This complete and very powerful program provides five levels of play. It includes castling, en passant captures and the promotion of pawns. Additionally, the board may be preset before the start of play, permitting the examination of "book" plays. To maximize execution speed, the program is written in assembly language (by SOFTWARE SPECIALISTS of California). Full graphics are employed in the TRS-80 version, and two widths of alphanumeric display are provided to accommodate North Star users.

STARTREK 3.2 (Available for all computers) — **Price: $ 9.95 Cassette/$13.95 Diskette**
This is the classic Startrek simulation, but with several new features. For example, the Klingons now shoot at the Enterprise without warning while also attacking starbases in other quadrants. The Klingons also attack with both light and heavy cruisers and move when shot at! The situation is hectic when the Enterprise is besieged by three heavy cruisers and a starbase S.O.S. is received! The Klingons get even!

SPACE TILT (Apple only) — **Price: $10.95 Cassette/$14.95 Diskette**
Use the game paddles to tilt the plane of the TV screen to "roll" a ball into a hole in the screen. Sound simple? Not when the hole gets smaller and smaller! A built-in timer allows you to measure your skill against others in this habit-forming action game.

GAMES PACK I (Available for all computers) — **Price: $9.95 Cassette/$13.95 Diskette**
GAMES PACK I contains the classic computer games of BLACKJACK, LUNAR LANDER, CRAPS, HORSERACE, SWITCH and more. These games have been combined into one large program for ease in loading. They are individually accessed by a convenient menu.

GAMES PACK II (Available for all computers) — **Price: $9.95 Cassette/$13.95 Diskette**
GAMES PACK II includes the games CRAZY EIGHTS, JOTTO, ACEY-DUCEY, LIFE, WUMPUS and others. As with GAMES PACK I, all the games are loaded as one program and are called from a menu.

Why pay $7.95 or more per program when you can buy a DYNACOMP collection for just $9.95?

NOMINOES JIGSAW (TRS-80 only) — **Price: $16.95 Cassette/$20.95 Diskette**
NOMINOES JIGSAW is an intriguing and sophisticated graphical puzzle. The jigsaw consists of a 9 by 9 board partially filled with randomly chosen shapes (nominoes), of which there are 60 types. By knowing that the shapes must be legally connected, and by guessing the shape at each location, all the nominoes may be eventually deduced. Scoring is based on the number of guesses required and the difficulty of the board set-up.

MOVING MAZE (Apple only) — **Price: $10.95 Cassette/$14.95 Diskette**
MOVING MAZE employs the games paddles to direct a puck from one side of a maze to the other. However, the maze is dynamically (and randomly) built and is continually being modified. The objective is to cross the maze without touching (or being hit by) a wall. Scoring is by an elapsed time indicator, and three levels of play are provided.

BLACK HOLE (Apple only) — **Price: $14.95 Cassette/$18.95 Diskette**
This is an exciting graphical simulation of the problems involved in closely observing a black hole with a space probe. The object is to enter and maintain, for a prescribed time, an orbit close to a small black hole. This is to be achieved without coming so near the anomaly that the tidal stress destroys the probe. Control of the craft is realistically simulated using side jets for rotation and main thrusters for acceleration. This program employs Hi-Res graphics and is educational as well as challenging.

TEACHER'S PET I (Available for all computers) — **Price: $ 9.95 Cassette/$13.95 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.

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.

CRANSTON MANOR ADVENTURE (North Star only) — **Price: $19.95**
At last! A comprehensive Adventure game for the North Star. CRANSTON MANOR ADVENTURE takes you into mysterious CRANSTON MANOR where you attempt to gather fabulous treasures. Lurking in the manor are wild animals and robots who will not give up the treasures without a fight. The number of rooms is greater and the associated descriptions are much more elaborate than the current popular series of Adventure programs, making this game the top in its class. Play can be stopped at any time and the status stored on diskette. Requires 32K.

NORTH STAR SOFTWARE EXCHANGE (NSSE) LIBRARY — **Price $9.95 Diskette**
DYNACOMP now distributes the 20+ volume NSSE library. Most of these diskettes offer an outstanding value for the purchase price. Write for details regarding the contents of this library and quantity (four or more) purchases.

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.

BUSINESS and UTILITIES

MAIL LIST II (Apple and North Star diskettes only) — **Price: $24.95**
This many-featured program now includes full alphabetic and zip code sorting as well as file merging. Entries can be retrieved by user-defined code, client name or Zip Code. The printout format allows the use of standard size address labels. Each diskette can store more than 1100 entries (single density North Star or Apple DOS 3.2; over 2200 with double density North Star or Apple DOS 3.3)!

FORM LETTER SYSTEM (FLS) (Apple and North Star diskette only) — **Price: $17.95**
FLS may be employed to generate individually addressed form letters. The user creates the address file and separately composes the letter. FLS will then print form letters using each address. FLS is completely compatible with MAIL LIST II, which may be used to manage your address files.

FLS and MAIL LIST II are available as a combined package for $37.95.

TEXT EDITOR I (Letter Writer) — **Price: $14.95 Cassette/$18.95 Diskette**
An easy to use, line-oriented text editor which provides variable line widths and simple paragraph indexing. This text editor is ideally suited for composing letters and is quite capable of handling much larger jobs. Available for all computers.

PERSONAL FINANCE SYSTEM (ATARI only) — **Price: $34.95 Diskette**
PFS is a single disk menu oriented system composed of 10 programs designed to organize and simplify your personal finances. Features include a 300 transaction capacity; fast access; 26 optional user codes; data retrieval by month, code or payee; optional printing of reports; checkbook balancing; bar graph plotting and more. Also provides on the diskette is ATARI DOS 2.

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.

DFILE (North Star only) — **Price: $19.95**
This handy program allows North Star 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.

COMPARE (North Star only) — **Price: $12.95**
COMPARE is a single disk utility software package which compares two BASIC programs and displays the file sizes of the programs in bytes, the lengths in terms of the number of statement lines, and the line numbers at which various listed differences occur. COMPARE permits the user to examine versions of his software to verify which are the more current, and to clearly identify the changes made during development.

COMPRESS (North Star only) — **Price: $12.95**
COMPRESS is a single-disk utility program which removes all unnecessary spaces and (optionally) REMark statements from North Star BASIC programs. The source file is processed one line at a time, thus permitting very large programs to be compressed using only a small amount of computer memory. File compressions of 20-50% are commonly achieved.

GRAFIX (TRS-80 only) — **Price: $12.95 Cassette/$16.95 Diskette**
This unique program allows you to easily create graphics directly from the keyboard. You "draw" your figure using the program's extensive cursor controls. Once the figure is made, it is automatically appended to your BASIC program as a string variable. Draw a "happy face", call it H$ and then print it from your program using PRINT H$! This is a very easy way to create and save graphics.

TIDY (TRS-80 only) — **Price: $10.95 Cassette/$14.95 Diskette**
TIDY is an assembly language program which allows you to renumber the lines in your BASIC programs. TIDY also removes unnecessary spaces and REMark statements. The result is a compacted BASIC program which uses much less memory space and executes significantly faster. Once loaded, TIDY remains in memory; you may load any number of BASIC programs without having to reload TIDY!

STATISTICS and ENGINEERING

DATA SMOOTHER (Not available for ATARI) — **Price: $14.95 Cassette/$18.95 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: $14.95 Cassette/$18.95 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 $44.95 (three cassettes) and $56.95 (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: $19.95 Cassette / $23.95 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 $49.95 (three cassettes) or $61.95 (three diskettes).

BASIC SCIENTIFIC SUBROUTINES, Volume I (Not available for ATARI)
DYNACOMP is the exclusive distributor for the software keyed to the 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.

ROOTS (Available for all computers) — **Price $9.95 Cassette/$13.95 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.

Ordering Information

All orders are processed and shipped postpaid within 48 hours. Please enclose payment with order along with computer information. If paying by VISA or Master Card, include all numbers on card. For orders outside North America add 10% for shipping and handling.

Add $2.50 to diskette price for 8" floppy disk (IBM format soft sectored, CP/M, Microsoft BASIC)

*TRS-80 diskettes are not supplied with DOS or BASIC.

Deduct 10% when ordering 3 or more programs.

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

DYNACOMP, Inc.
6 Rippingale Road Dept. E
Pittsford, New York 14534
24 hour mail order phone: (716)586-7579
Office phone (9AM-5PM EST): (716)442-8960

VISA

master charge

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

⁺ ATARI, PET, APPLE II, TRS-80, NORTH STAR, CP/M and IBM are registered trade names and/or trademarks.

The chip select pins must be controlled by the address decoding circuitry in your microcomputer system, or else you must add your own address decoding circuitry to produce the chip select pulses. Since I have an AIM 65 system, I used one of the device selects available on the expansion connector, namely $\overline{DS9}$. This signal is active at logic zero for any address in the range $9000 to $9FFF. It was tied to $\overline{CS1}$, the active-low chip select. If CSØ is tied to +5V, then the registers on the 6551 are selected by any address of the form $9XX0 to $9XX3 where XX is a "don't care" number. Thus, in the programs you will find the four 6551 registers selected by addresses $9400 to $9403. If you have a SYM-1 you can make use of device select $\overline{DS18}$. If you have an Apple you must provide your own decoding circuitry. The reason lies in the fact that the device select pulses generated by the Apple for the peripheral cards have been logically ANDED with $\emptyset_0$, and consequently they do not become active early enough in either the READ or WRITE cycles to work with 6500 family devices. For a discussion of address decoding see De Jong's (1) book. The circuits are generally quite simple. In the case of the Apple, I suggest trying an inverter and a 74133 to generate a device pulse for say $C08X, and perhaps a 74LS245 as a data bus buffer. Try an inverted $\emptyset_1$ to replace $\emptyset_2$. My familiarity with the PET does not justify suggesting any circuits, but I am sure the 6551 can be interfaced to a PET.

We turn next to the signals on the right-hand side of the 6551 as it is shown in Figure 1. The RXC input to the 6551 is the easiest to explain because it is not used in this application. The remaining pins have labels that are almost identical to the RS-232C designations. In fact, the only one that is different is the $\overline{DCD}$ which is simply CD (Carrier Detect) in RS-232C lingo. Again, refer to references (2) and (3) for a more complete explanation of the RS-232C interface.

Although the signal designations on the 6551 ACIA are almost identical to the RS-232C labels, the signal levels are not, and some arrangement must be made to transform the TTL levels of the 6551 to the RS-232C signal levels. We chose to use integrated circuits designed expressly for that task, namely the 1488 and the 1489 line driver and line receiver. Note that the 1488 requires a positive and negative supply voltage as well as ground. Also, the RS-232C ground (pin 7 on the DB-225 connector) should have the same ground as the 1488 and the 1489. The connections in Figure 1 that are found on the right-hand side of the figure made up a rather complete RS-232C serial interface that may be used to interface to a variety of peripheral devices. Furthermore, the fact that the data format and Baud rate of the 6551 are under the programmer's control makes this an extremely flexible RS-232C interface.

Since computer communication by telephone is the subject of this article, a modem is required. There are a variety of modems with RS-232C interfaces on the market and we do not wish to make any recommendations about them. I purchased a Novation Cat because that appears to be one of the more popular devices. Skyles Electric Works and other advertisers in COMPUTE! offer modems for sale. In any case, my Novation Cat requires the signals designated in Figure 1 in addition to the signal ground. Other modems may require the DTR and RTS signals so we have shown the correct TTL-to-RS-232C interface in the event you may need these signals.

This completes our description of the circuit and we turn next to a simple program that is supposed to allow communication to take place using the 6551 ACIA.

III. A Simple Communications Program

A program that was designed to allow two people to communicate over telephone lines with their computers is given in Listing 1 and a flowchart is shown in Figure 2. This program is very simple and very slow, and it is offered here merely as a way to test the circuit, the program, and the modem. Eventually, one would want to construct more elaborate routines to transfer information quickly. Our interest here is in experimenting for the sake of learning. Hence there is no necessity for encryption devices, bells, whistles or even parity checks.

Here is how it is supposed to work. The caller loads the program and places his modem in the **originate mode** with **full-duplex operation** selected. He loads the indirect jump location with the vector $0F13 so that after the program is begun, his program will go to the transmit loop.

He makes the telephone call to an anxiously awaiting friend who also has this interface and this program operating. The friend has loaded the indirect jump location at $0000 and $0001 with the vector $0F26 (remember, $26 goes in $0000 and $0F goes in $0001). The friend has also placed his modem in the **answer mode** with **full-duplex operation** selected. After an informal chat, both friends put their modems into action by placing the handset into the muffs (assuming acoustic modems). The originator begins to type a message.

He ends his part of the message with an 'EOT' character (Control D on your keyboard). While he is transmitting, the friend's program echoes the message back to the originator where it is read and printed by the computer. It's nice to see what you have said, and to know that it got where it was going with no mistakes. When an 'EOT' character is sent, it automatically transfers the originator's program to the receive loop and the receiver's program to the send loop, giving him a chance to retaliate. Having made no visible symbol to indicate when this changeover takes place, may I suggest sending a

Listing 1. An Experimental Communications Routine.

$0F00 A9 0B	START	LDA #$0B	Initialize the 6551 by loading the command
0F02 8D 02 94		STA CMNDRG	register (see 6551 spec sheet for details).
0F05 A9 13		LDA #$13	Load the control register for 8-bit word
0F07 8D 03 94		STA CNTRG	length and Baud rate of 110.
0F0A 78		SEI	Prevent interrupts.
0F0B D8		CLD	Clear decimal mode.
0F0C EA		NOP	A mistake of mine.
0F0D AD 01 94		LDA STATUS	Clear any interrupts on the 6551.
0F10 6C 00 00		JMP (THERE)	Jump to transmit loop to transmit, receive
0F13 20 3C E9	TXLOOP	JSR KYBD	loop to receive. Get a character from the
0F16 20 F0 0F		JSR TXMIT	keyboard read routine. Send it to the 6551
0F19 C9 04		CMP #'EOT'	transmit subroutine. If an "End of Transmission"
0F1B F0 09		BEQ RXLOOP	(Control D) is sent, branch to receive loop.
0F1D 20 E0 0F		JSR RCVDAT	Get the echo from the receive subroutine.
0F20 20 7A E9		JSR OUTPUT	Output it to your own printer to see what you
0F23 18		CLC	sent. Force a jump back to TXLOOP
0F24 90 ED		BCC TXLOOP	and get another character to send.
0F26 20 E0 0F	RXLOOP	JSR RCVDAT	Wait for a character to be sent.
0F29 C9 04		CMP #'EOT'	Is he finished with his transmission?
0F2B F0 E6		BEQ TXLOOP	Yes, then go to transmit loop.
0F2D 20 F0 0F		JSR TXMIT	Echo the character that was sent.
0F30 20 7A E9		JSR OUTPUT	Output the character to your printer.
0F33 18		CLC	Force a jump back to RXLOOP
0F34 90 F0		BCC RXLOOP	and get another character when it is sent.

Subroutines

0FE0 AD 01 94	RCVDAT	LDA STATUS	Read the status register to see if a word
0FE3 29 08		AND #$08	has been received, otherwise wait for one.
0FE5 F0 F9		BEQ RCVDAT	
0FE7 AD 00 94		LDA RCVRG	Get the word from the receiver register.
0FEA 60		RTS	Return to the calling program.
0FF0 48	TXMIT	PHA	Save the accumulator temporarily
0FF1 AD 01 94	WAIT	LDA STATUS	Is the transmitter register empty?
0FF4 29 10		AND #$10	No. Wait until it is.
0FF6 F0 F9		BEQ WAIT	
0FF8 68		PLA	Get the character from the stack.
0FF9 8D 00 94		STA TMTRG	Store it in the 6551 transmit register.
0FFC 60		RTS	Return to the calling program.

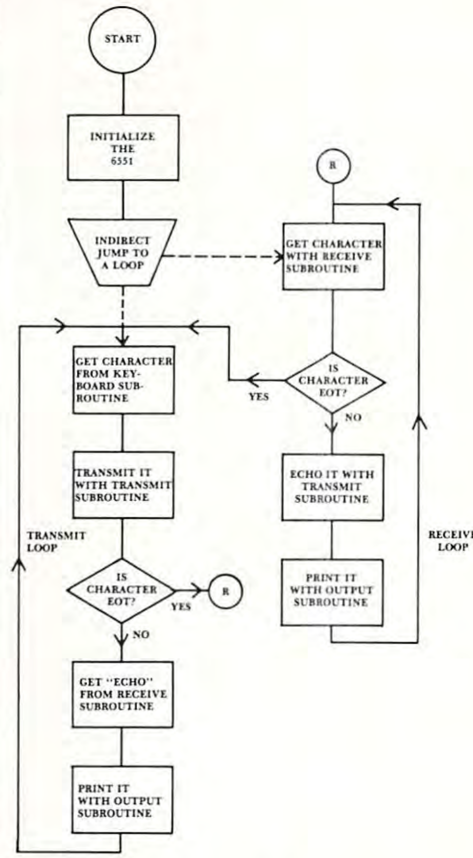


Figure 2. Flowchart of the Transmit/Receive Program. See text for details.

question mark, or perhaps there is some CB lingo that suggests it is the other person's turn to talk. If all else fails, pick up the handset and holler something. *Do not* change your modem from its original answer or originate mode.

So back and forth the conversation goes. Once you have the transmit option in your hands nothing can stop you from talking until you send an 'EOT' and give your friend a chance to say something. Clearly, the program lacks a certain elegance (it may not even work, in which case it lacks a whole lot of elegance), but maybe it will get some fun started. By the way, the originator of the phone call usually gets the phone bill.

Study the flowchart and the program listing. The program begins by intitializing the 6551. An eight-bit word (TTY compatible) is used, with the parity check disabled, and one stop bit is sent. The Baud rate chosen here is 110, but it should be possible to go to 300 Baud. Both participants must have the same rate. Next, the program jumps to either the receive loop or the transmit loop depending on the vector loaded into $0000 and $0001. This was a crude way to start, but it should work.

In the transmit loop the program first waits for an input from a keyboard read routine. The address in the program belongs to an AIM 65 monitor subroutine that returns the ASCII representation of the depressed key in the accumulator. This character is sent by calling the transmit subroutine which loads the 6551 transmit register with the character. The 6551 takes over and sends the character. The program then waits for the character to be echoed from the other telephone and computer. The echoed character is printed to make sure that what was sent was actually received. Then control returns to the keyboard subroutine to wait for the next character to be sent.

In the receive loop the program jumps to the receive subroutine that watches the 6551 until a character is in the receive data register. If this character is an 'EOT' then control goes back to the transmit loop and you may begin transmitting. Otherwise the received character is immediately echoed back to the sender and also printed with your OUTPUT routine. Again, the address of the OUTPUT routine in Listing 1 belongs to an AIM 65 subroutine. Both the KYBD and OUTPUT subroutines must be supplied by the user's monitor or the user himself, otherwise the program in Listing 1 is complete.

While in the transmit loop, the selection of the 'EOT' character by the sender will automatically transfer control out of the transmit loop into the receive loop. Note again that no bells or whistles have been programmed to occur when you send an 'EOT' character, so if you are transmitting you better let your friend know you are passing control of the system to him.

So hopefully all this will work. If it doesn't you have only me to blame, and I will not assume the cost of your labor or your equipment to conduct this experiment. Perhaps it would be best if you waited until someone else tried it; think it over before you take the plunge.

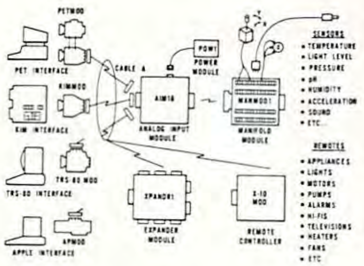
Besides, my next project is to launch a 6502 Communications Satellite using dynamite in my back yard and you may want to save your money to buy shares in that enterprise.

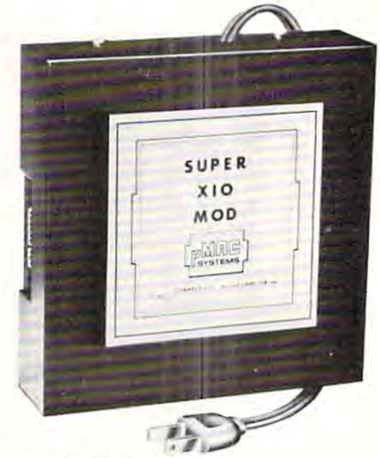
References

1. De Jong, Marvin L., *Programming & Interfacing the 6502, With Experiments,* Howard W. Sams & Co., Inc. Indianapolis, 1980.
2. Day, Michael E., "RS232 Communications,", *COMPUTE!,* Sept/Oct 1980, 26.
3. Ciarcia, Steve, "I/O Expansion for the TRS-80," *BYTE,* June 1980, 42.

©

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





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

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

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

Connectors

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

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

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

Analog Input Module

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

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

Power requirements are 12 volts DC at 60 ma.

POW1

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

TEMPSENS

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

Remote Controller-Clock and Calendar

AN INEXPENSIVE CONTROL SOLUTION FOR

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

SUPER X-10 MOD SPECS

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

XPANDR1

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

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

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

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

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

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

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

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

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

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

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

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

PETSET1a (Commodore PET - 110 VAC) **295.00**
PETSET1e (Commodore PET - 230 VAC) **305.00**

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

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

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

Basics of Light Pen Operation

Robert A. Peck
Manager, Technical Support to
Advanced Manufacturing
Memorex Corporation

Manufacturers of personal computers are attempting to make the computer more easily accessible to the public. In doing so, various means have been tried, such as games, simple home budget programs, and the like. The entry format for each of these had, for the most part, been by means of the keyboard, or a game paddle of some kind.

Just recently, the trend toward light pen "menu" selection for ease of data entry has been tried. Let's look at the actual techniques which could be employed to implement this type of input device on a personal computer. The hardware and software requirements will be discussed here. The reader, after studying these techniques, may be able to construct a working form of a light pen with as little as $5.00 worth of materials.

First a note about the definition of a light pen. It does not emit light... rather it is intended to sense the light of an illuminated area on the TV monitor screen. As a photo-sensitive device, some form of output of the light pen will occur as a result of the electron beam energizing a portion of the phosphor of the screen, thus causing it to glow.

To clarify this a bit further, the picture on a TV screen is not all produced at exactly the same time by a single "photo" flashed on the screen. Instead it is made up of a single electron beam being swept from left to right (and down and up) across the screen, with its intensity varied as it sweeps across the screen, to form the picture we see, one line at a time. In this manner, the sweeping beam produces 30 or 60 complete pictures per second on the TV screen. Our own visual system enables us to perceive the screen as though the entire surface of the screen was continuously lit, thus forming a complete picture. The persistence of the screen, the time it remains bright after the beam has passed a particular location, is minimal for most monitor screens.

Let's act, for the moment, in the same manner that the light pen will act. Imagine, if you will, taking a small cylindrical tube and placing it against the surface of a fully illuminated TV screen. If we place our eyes at the opposite end of this tube, and restrict our vision only to what is at the end of the tube and not to the rest of the screen, we are in a position to make a judgement about what is going on in our narrowly restricted view of the world.

Now we must imagine either that we are able to speed up our ability to perceive rapidly the changing intensity of the light on the area of the screen in front of us or that the beam slows down to our level of perception. Either position is ok for our purposes.

As we are looking into the end of the tube, we will notice that there is no light there most of the time. Specifically the phosphor will only be lit up exactly at the time the beam is striking it, and for a short time thereafter, based on the persistence of the screen. But of course, for the most part, we will be kept in the dark. This will be true at any position on the surface of the screen.

Since we know that we have light for a short time and dark for the rest of the time, it is a yes-no situation and something ideally suited to being handled by the computer. So let's give our eyes a rest and place a lens and a phototransistor within the tube in place of our eye. We know that the phototransistor will produce an output when it sees the light and no output when the light is absent.

...a very simple design will serve most purposes admirably.

How complex must the circuit for the phototransistor be to allow us to make this a useful, reliable device? Well, it depends on the type of selection which we wish to make in the usage of the light pen. We'll soon see that for the largest percentage of potential uses, at least at the hobbyist level, a very simple design will serve most purposes admirably.

In order to grasp the significance of the output/no output capability of the phototransistor, we'll next look at the way the computer or its terminal device is producing the output display which we are seeing on the TV.

Let's say we have a terminal which can display 80 columns by 24 lines of usable character positions on the monitor screen. In the process of output, the scan controller must select, each in turn, line 1 of the character memory, then columns 1, 2, 3, ..., all the way out to column 80. Then it must repeat the line scan for as many TV scan lines a character line is supposed to take up. Then it will go on to the next line of 80 characters, the next, and so forth, going back to the beginning again once all 25 lines have been displayed.

From this, you can see that the scan controller will be continuously fetching characters from the character memory. Thus the different addresses of the different characters will each be available on the address bus of the scan controller at the time that character is being fetched for output.

To put it another way, if our character memory was set up such that line 1 position 1 represented address 0 of the character memory and line 1 position 80 represented character address 79, line 2 position 1 as address 80 and line 25 position 80 as address 1999, we would then have a specific point of reference to use. We now take our light pen and place it on the screen directly over one specific character position which is, let us say, occupied by a single solid block character such as a nonblinking cursor.

Every time the phototransistor sees a light output on the screen, at the exact time it occurs, the scan controller address bus has, on it, the exact address within the scan memory occupied by the character which is producing the light output on the screen.

Just as an example of what this address would mean to us, consider the following example. Suppose that at screen location 400 (first position in line 6) we place a cursor character followed by the description .. "CHECKBOOK BALANCER" and at location 800 in the scan memory we placed another cursor character labeled "TREK", we can place the light pen over the cursor character representing the specific program which we wish to have called in next and will expect the light pen scan program to provide us with the data required to do it. In this case when the light pen senses an output, the address of either position 400 or 800 will be on the scan position address bus, ready to be picked up for use by our program. We know that if our program finds 400 on the bus, it must next call in the Checkbook Program. Conversely, if it finds 800 on the bus, it must retrieve the Startrek Program.

Now if we wanted to do so, we would add some additional hardware to our terminal which would act, in association with the phototransistor output, to capture the address present on the bus at the time a light output is sensed. As an exercise, let's examine some of the hardware this would require.

First we need something to capture the scan address from the controller bus and a way to transfer it to the data bus of the computer. A set of three 74175's could be used here. Each is a 4-bit tri-state latch, where the input (capture side) would be connected to the scan memory data bus and the output (storage side) would be connected to the computer data bus for later retrieval. The control lines for the latches would have to be connected in some manner to the light pen through a flip-flop of some kind to assure only a single sampling of the address from the scan counter per application of the light pen to the screen. The tri-state control lines would be connected to the address decoders of the computer so that it could retrieve any one of the three 4-bit stored parts of the scan address after it was triggered.

The best features of this technique are the simplicity of the software required and the non-critical nature of the components of the light pen...

It might, at first glance, seem a pretty straightforward approach to follow, but let's look at a few of the drawbacks. The first would be the critical control of the level of light intensity sensed by the pen. Specifically, it could possibly be accidentally triggered either by an outside source of light, or by the phosphor persistance (as little as there is) when we first place the pen against the surface of the screen. In either case, the address we sense on the scan control bus does not really represent the actual address of the sample point we are trying to isolate. This might entail some special circuitry to be added to sense only the rising edge of the beam light intensity, where that rising edge has a specific rise time, and therefore not trigger on an outside incandescent light source operating on a 60 Hz sine wave.

To complicate matters further, even though we succeeded in developing this type of edge sensitive equipment, we still run into some problems with fluorescent light sources in the area, in that these have a very fast rise time and have a phosphor afterglow as well. Both items make the light from the fluorescent vary in a manner similar to that of the TV screen. Our software could, of course, compensate for this, but combined with the hardware requirements, we have selected a really complex task. One more area of difficulty, just to mention it here, is the inability to accurately sense the difference in address locations between two adjacent, or very nearly adjacent, squares on the screen unless special circuitry is added for the rise-edge, as described above.

Fortunately, there is a way to absolutely minimize the amount of circuitry needed to establish a workable light pen, along with a way to minimize the complexity of the software which has to go along with it. In addition, the pen needs only to accept a source of power and ground from the computer, and will need only a single bit input port to operate fully. Some manufacturers suggest the use of the same paddle input for the light pen. Below is described the technique which can accomplish this form of operation.

The best features of this technique are the simplicity of the software required and the non-critical nature of the components of the light pen, as

well as the non-critical nature of the level adjustments required. We also have an easy way of compensating for any external light source which may have an effect on the pen, and actually ignore it. Let's examine this technique now.

First, let us attack the problem of light sensitivity adjustment. It is proposed that, for this method we will work with, the pen need only be able to distinguish between the presence or absence of illumination within a selected square on the screen. If we are working within a range of light or no light, you can see that we will have a wide degree of adjustment available and still allow the pen to operate perfectly well.

With the original example, let's say we had placed a menu selection box at both scan memory locations 400 and 800 and assume also we are using the simple-form light pen which plugs into the game paddle input. Instead of using a hardware-based scan technique, we will use a software based scan as follows.

Assume for example that we have placed the pen over the square at scan location 800 and we begin our scan. Both the square at 400 and at 800 appear illuminated at this time. Therefore if the light pen is pointing to either one of them, during the period of time of the sweep of the beam across the screen, the light pen will put out a series of pulses coincident with the presence of the beam in the area occupied by the pen.

Now we can begin our scan by replacing the selector box at scan memory location 400 with a blank space (no output on the screen at this point). We will then go to the light pen input and stay in a loop for about 1/60 or 1/30 second and find out if, during that loop, there were any light pulses output. If there were still output light pulses, it means that we had not turned off the square over which the light pen is resting now, so we must continue the scan. Then we relight the square at location 400 and proceed to replace the square at 800 with a blank space. We will again loop through the test program area to determine if there have been any light pulse outputs during the time that location 800 was turned off. If no outputs were sensed during this time, we know we have found the correct location where the light pen is sitting.

We can then take the address we have found this way and use it to control which action is to be done next, just as in the previous hardware controlled case, but here with a good deal less complexity. You can also see that we may have many many menu boxes on the screen and by this means accurately determine exactly which one is being addressed by the light pen. After all, we are the one who is controlling whether the light pen can see a light output from a specific square. So if we turn off a square, and then see that the light pen no longer has an output, we know which square we just controlled and therfore we know what the required operation will be.

We have then substituted a software scan technique for both the complex hardware and complex software the other approach would have required. The primary limit in the number of menu boxes we can use is the amount of time which would be required, at 1/60 or 1/30 sec per box, for the light-pulse-present scan per box on screen. If we have no concern for the time this takes, then there is little reason to limit the number of boxes on the screen except to keep them far enough apart so that the light pen will see the light output from only one at a time, maintaining the wide range of light sensitivity we discussed earlier.

...thus far we have substituted a software scan for the set of complex hardware.

Speaking of light sensitivity, let's discuss the way we'd handle an outside light source and ignore its influence in our selection of the item to be performed. First, a reminder that the single spot on the TV screen we are monitoring is dark for most of the time and is lit by sweeps of the beam only as it passes the area we are monitoring. Now if we consider the outside source of light, it will rather seen to be a continuous sequence of pulses (fluorescent) or a continuous single light level. In the event that there is some continuous pulse interference, we must adjust our software to test that there are no more than X (let's say 50) pulses which occur during a single sweep through our software scan subroutine. This would allow us to ignore such interference as is caused by either a fluorescent or an incandescent source. Certain types of light, such as the sun, cannot be distinguished by the pen as a wave, so are translated as a continuous level, thereby resulting in nearly zero (perhaps one) transitions during the time of a single scan. Thus, we decide that unless greater than one and less than 50 pen state transitions have occurred in one scan, we could probably assume that the visible part of the screen scan probably had been triggered by an outside source, and we can enter into some type of wait state, scanning the pen itself for a time when the correct number of transitions is sensed and, within the wait state loop, also scan our keyboard and any other alternate input device which may be connected to the system and intended for use with the particular program as an input.

A final note about outside light sources; when we have the pen up against the screen, the major influence on the pen will be the light from the screen alone. In this position, the pen will not be affected very much by the outside light.

So thus far we have substituted a software scan for the set of complex hardware. We have used a technique which requires very little translation of address sensed into work to be done. We could probably go into the type of construction required for the light pen itself.

But wait, there seems to be some griping from the back of the room. Yes ... Oh ... OK! The gentleman in the last row says "That's ok for you guys who have the Visible Memory (direct access) display screens, but how about the rest of us who only have the scrolling type of screen?"

A fair question, I agree. All right. A scrolling type of screen is one where everything moves up one notch to make room for the bottom line once the screen is filled. Well, a number of these types of screens have the ability to move the cursor in a relative manner or an absolute manner. If it does have this capability, then the technique still works exactly the same way... we just have to work a little harder. Lets look at a quick example.

We'll print a cursor box followed by a descriptive line on the screen, followed by a blank line, and repeat this for 5 selections. Now to do the cursor scan, we will begin from the lower left (home) cursor position and move-relative-cursor until we get to the position occupied by one of the selector boxes. Then we replace it with a blank space instead. Scan. Are there light pulses present? If not, we've found the light pen position. If so, cursor backspace, put back the selector box, space relative cursor to the next selector box and repeat the process. As you can see, there is no basic change in the procedure, just a slightly different approach.

Sir, ... you do have relative cursor?! Ok then, at least we've got one satisfied customer. By the way, you'd probably be interested; the terminal I use on my machine is a scrolling type and thats why I was prepared for that question!

Now for the construction of the light pen itself. We'll need some kind of small cylinder to house it. The cylinder will have to have enough room for the phototransistor itself. And, it should have some room for a small variable resistor and a voltage comparator IC if we want to have it fully self-contained and ready to plug into the game paddle input of our computer.

I have provided a sketch of the proposed construction of the pen, along with the schematic of the one I use. These parts I had were primarily junk-box components, and as a result, my total cost was about $1.00 (plus the software development time). You could probably obtain most of the components for $5.00 or less.

Well, best of luck with your construction and testing. If you develop some interesting applications for your light pen, I would appreciate the chance to hear about them:

Robert A. Peck
Manager, Technical Support to Advanced Manufacturing
MEMOREX CORP
San Tomas at Central Expy MS 10-10
Santa Clara, CA 95052

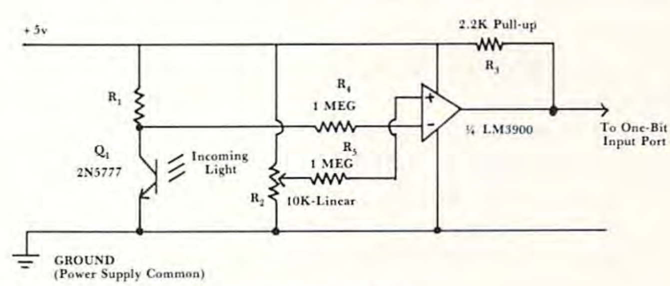


Fig. 1. Basic Inexpensive Light Sensor

Q_1 Mounted in Tip of Pen

R_2 Sensitivity adjustment, adjusted so that plus pulses are present while pen is on screen opposite a part of screen which is lit up.

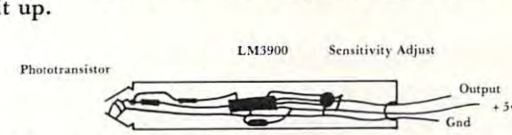


Fig. 2. Typical Construction

ABOUT THE AUTHOR: Bob has a BSEE from Marquette University and an MBA in Finance and Economics from Northwestern. He has been involved with computers since 1965, and has taught microcomputer courses for Cogswell College in Sunnyvale, CA. He has authored three booklets on hardware and software for the 6502-based SYM-1 Single-Board computer. His assignment at Memorex involves arranging a smooth transition for new products from development engineering to Manufacturing. ©

Getting The Most From Your Pet Cassette Deck

Editor's Note: There's much of value here for any cassette user, regardless of machine type. A couple of asides — the new recorders from Commodore (the VIC version) have tape counters. The second point is simply a comment on Mr. Sander's remark regarding mail-order computer store tapes... most are quite reliable in business practice and quality control. RCL.

Louis F. Sander
Pittsburgh, PA

PET owners not fortunate enough to own a disk spend many minutes, and ultimately many hours, waiting for the cassette deck to finish its work. SEARCHING seems to take forever, and we never know whether the search will finish with a READY, or with the dreaded ?LOAD ERROR. This article gives some practical advice on making that waiting time shorter, more productive, and less filled with anxiety. It is oriented toward the novice, and it contains much that has been explained before, although never to our knowledge all in one place. But even the most experienced PLAY presser should find something of value in it. We begin with a treatise on tape buying, proceed to information on recorder care and useful accessories, and end with a compendium of helpful hints for the recordist, librarian and programmer.

What Should You Feed A PET?

Standard cassettes can be had at prices from under 50¢ each to over $5.00, and it seems impossible to know which ones to buy. Since the typical PET owner will end up with dozens of tapes in his library, knowing a bit about cassettes can be quite important — we want to be sure that ours will perform reliably, without contributing to the loss or ruin of valuable programs, but we don't want to pay extra for quality we can't really use. (After all, most of us are saving up for that disk system.) A careful study will show that there are three main areas of difference among cassettes, each of which we'll discuss here: playing time, mechanical construction, and type of magnetic tape.

First, playing time. Every cassette is marked with a number such as C-30, C-60, C-90, etc. The digits after the 'C' tell how many total minutes of playing time there are on *both sides* of the tape. A C-30, for example, has two 15-minute sides, for a total of 30. Even though the longer tapes cost very little more than the short ones, for most PET owners the C-30 is the longest one to buy. One side of a C-30 will hold at least six long (8K) programs, and can be fully rewound in about 60 seconds. A C-60 will hold twice as many programs, but it gets tedious to search through all that tape to find the one you're looking for; the rewind time is longer, too. The C-90's and above tend to be made with very thin tape that likes to break, or to let data leak through from one side of the tape to the other, either of which can ruin your program and your day. Probably the *best* size is the C-10, which is not widely available, but which holds one or two programs on a side, and which nicely minimizes search time.

Cassette construction is less obvious than the other two factors, but it does bear some discussion. Cassette housings range from sloppily molded boxes to finely assembled mechanisms with bearings and other anti-friction devices. Most housings are glued together, but some are assembled with screws. Many experienced PET users prefer the screw type, which can be taken apart for emergency untangling of tapes. (That can be a big factor when the fouled tape has your latest masterpiece on it.) Sloppy construction is most often found in off-brand discount store cassettes, and it should be avoided, since a sloppy housing tends to let the tape escape and be mangled by your recorder. In general, the more expensive cassettes have better housings, and are easier to rewind or fast forward, but you should have little trouble with any but the very poorest housings.

The finest and most expensive magnetic tape has a chromium dioxide (CrO_2) coating, and should *not* be used in the PET. It requires special circuitry that the PET doesn't have, and its greater abrasiveness can cause rapid wear to tape-handling parts. The next step down is extra-quality tape with a ferric oxide coating, usually selling for $2.50 -$5.00 or more per cassette; this tape is designed to give a very wide frequency response in stereo recording and playback. It will work fine in the PET's monophonic recorder, but its premium quality doesn't add much to performance, and for many people the extra quality is not worth the extra price. The same can be said of the "certified" computer cassettes in this price range. Your PET doesn't need "computer quality" tape, or leaderless tape, so why pay extra for it?

Further down the line is garden-variety ferric oxide tape with a well-known audio or electronics brand, usually sold for under $2, or much less in multi-packs. For most PET owners, this is the best combination of price and performance. The tape is designed for monophonic recorders like the PET's, and it has the uniform quality usually found in well-known brands of any product. The widely available Radio Shack Concertape, starting at 3/$1.99, is a good example of this kind of tape. Also in this price range are the cassettes sold by mail-order computer stores that cater to PET owners. There are some real advantages to these cassettes — the price is right, they are available in the convenient C-10 size, and they are usually screw-assembled. But there can be risks, too. Some mail-order computer stores are

shaky operations with flaky quality control and fluky business practices. A good policy with these tapes is to try them if it suits you, but keep a close eye on what you get.

At the bottom of the list are the tapes you should avoid — the ones sold in discount stores, with brands you never heard of in audio or electronics. These are not much cheaper than Concertapes, and the tape inside is sometimes uneven and dirty. For most of us, the risk of getting junk is not worth the savings, so we should stick with something better.

Looking Out For Tape #1

Every tape head needs periodic maintenance, and the two on your PET are no exception. Experts recommend cleaning and demagnetizing tape heads after every ten hours of use, and you do yourself a big favor by following their advice. If you neglect these important tasks, sooner or later you'll begin to notice frequent LOAD ERROR messages, and you may permanently damage every recording you pass by the head. Tape head tolerances are measured in microinches, and it's very common for an invisible buildup of oxide residues to cause major signal losses, often leading to LOAD ERRORs. It's also common for recording heads to become magnetized after a period of use, especially if the recorder power is cut while doing a SAVE. A magnetized head partially erases every tape that is run past it. A dirty head can scratch tapes. Remember, a good head session takes only about 1010_2 minutes, and it clears your head for another $ØA_6$ hours of use, so it's well worth the effort.

To start your maintenance program, get a bottle or spray can of tape head cleaner and a package of swabs. It's helpful, too, to get a small angled mirror, so you can inspect the heads while you work on them. Also get a head demagnetizer, of the type that plugs into the wall. (The cleaners and demagnetizers that look like cassettes are not as effective as the other types, and some poor ones can actually damage your head, so we suggest that you avoid them.) All these items can be purchased, often in kit form for under $20.00, at any good audio or electronics store. Sometimes you can borrow them from a friend who's into stereo or home computers.

When maintenance time comes, follow the instructions that come with the cleaner, and thoroughly swab the heads, tape guides, capstan and pinch roller, all of which you can get to by unplugging the PET and depressing the PLAY control. If you can't identify which parts to clean, any knowledgeable stereo salesman can show you the corresponding parts on his equipment, and that should be enough to get you started. Next, demagnetize the heads, meticulously following the instructions that come with your demagnetizer. Particularly avoid cutting power to the demagnetizer when it is anywhere close to a head, or you may magnetize it yourself. Keep your tapes at least 5 - 6 feet away from the demagnetizer at all times, or you may accidentally erase them. Remember that magnetic fields pass easily through everything but steel, and that a wooden desk drawer can hide tapes from you, but not from your demagger.

Useful Tape Accessories

The most useful tape accessory is a second recorder, but *not* the kind that plugs into the Second Cassette Port. You will gain many enjoyable minutes by using an extra recorder of any kind to search or rewind one tape while LOADing another. When searching, just play the tape until you hear the high-pitched leader tone, and start it right there on your PET. The buzzsaw sound after the leader tone is the actual program material. If your extra recorder has a tape counter, you can use it to keep track of program locations on the tape, and further lessen your SEARCHING time. If it has the Cue/Review feature, you can listen to the recorded material while rewinding or fast forwarding, which is also very helpful in finding things. If your recorder has a built-in microphone, make or buy a short-circuited plug to fit the MIC jack and cut out the microphone; that will let you erase selected areas on your tapes, which is useful if you're recording over other material and getting a lot of VERIFY ERRORs. Without the built-in mike, you don't need the shorting plug.

The extra recorder, used in audio mode, can help you type in programs from COMPUTE! and other sources, too. Just read the program aloud into the microphone, carefully enunciating every comma and semicolon, then play it back to yourself and type in the program as you hear it. This is a super method for proofreading programs that don't work.

Another useful accessory is a bulk eraser, for quickly erasing tapes when you want to re-use them. Mine is a Nortronics Sound-Off, a permanent magnet unit that works by just sliding the cassette through a slot. Most of them plug into the wall, and work like head demagnetizers, but on a grander scale. Be careful with bulk erasers — they create a strong magnetic field that can erase your good tapes if they are anywhere close by.

The stores have many other items you might find worthwhile. Radio Shack has a slick manual rewinder. The Sams book "Tape Recording for the Hobbyist" and the Nortronics "Recorder Care Manual" are good sources of useful information. Advanced tape hobbyists may also like to have a tape splicer and a head alignment tape, but these are beyond the needs of most of us.

Tape Handling Tips

1. Running new tapes back and forth a few times before using them will minimize binding and breakage. Erasing tapes before re-use will minimize read errors due to "junk" on the tape. Breaking out

the write protect tabs on a cassette will keep you from writing over programs by mistake. Covering the write protect hole with tape will override the protection.

2. Keep your tapes clean: Rewind cassettes before putting them aside, and never touch the magnetic tape itself. Always use plastic cassette boxes; the soft ones are cheaper and tougher, but the hard ones are prettier. (I use hard boxes for master tapes and soft ones for working copies.) Keep your cassette boxes in metal containers; stray magnetic fields are everywhere, especially around motors and transformers, and they can damage unprotected tapes.

3. As soon as you SAVE a program, label the cassette with the program name. Half-inch masking tape makes an easily removable label for cassettes, and also fits perfectly on the edge of hard or soft cassette boxes. Half-inch Scotch Magic Tape makes a neatly erasable label for the same places.

4. A 1K program takes about 35 seconds to SAVE, VERIFY, or LOAD. A 4K program takes about 90 seconds, and an 8K program about 150 seconds, or 2½ minutes. The practice of SAVEing each program twice on the same tape will keep you happy in the face of minor malfunctions; the practice of keeping master copies on a separate tape in a separate room will keep you happy in the face of disaster.

5. There is a small but real danger of write-through when programs are recorded on both sides of one piece of tape. You can avoid it by using only one side of your cassette, or by using both sides and recording no further than mid-tape. I usually SAVE one program twice on each side of a C-10. That way I have minimal search and rewind time, conveniently located second copies of each program, and no overlapping.

6. During program development, SAVE your work frequently, so you'll have something to work with after an unanticipated NEW or system crash. To keep track of the different versions, make the date and time of the SAVE an integral part of your program's working name: "02141015SPACEWAR" fits into the 16-character limit, and indicates that this version of SPACEWAR was SAVEd on 2/14 at 10:15 AM. If there's ever a question, it will be obvious that "02141300SPACEWAR" is a later version, and that "01312200SPACEWAR" is an earlier one.

That's the end of one user's notes on saving time and grief with your PET's tape deck. There must be many other good ideas on the subject. If you have some, let us know about them.

Part 3 of several

The Mysterious And Unpredictable RND

Bob Albrecht and
George Firedrake

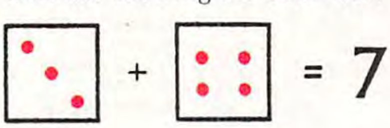
Editor's Note: You may reach Bob & George by mail at: P.O. Box 310 Menlo Park, CA 94025

Dice Roller

OK PET, let's roll one die a bunch of times. We will simulate rolling an ordinary six-sided die. For each roll, the possible outcomes are 1 or 2 or 3 or 4 or 5 or 6.

```
100 REM***DICE ROLLER #1

200 REM***FIND OUT HOW MANY ROLLS
210 PRINT "[CLR]" ;
220 INPUT "HOW MANY DICE ROLLS" ; N

400 REM***ROLL ONE DIE N TIMES
410 FOR K = 1 TO N
420   DIE = INT(6*RND(1)) + 1
430   PRINT DIE,
440 NEXT K
450 PRINT

999 END
```

For many dice games or other uses of dice, we roll *two* dice. The outcome of a roll is the total of the "spots" or number showing on both dice.



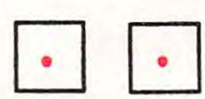
Your turn. Tell PET how to simulate rolling two dice.

Exercise 11. Write a program to simulate rolling two dice, N times.

```
HOW MANY DICE ROLLS? 20

8        7        3        10
9        9        8        4
6        7        10       11
5        6        5        6

READY
■
```

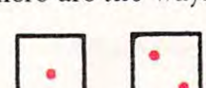
When we roll two dice, the possible outcomes are numbers from 2 to 12. However, they are *not* equally likely.

- There is only *one* way to get 2.



1 + 1 = 2

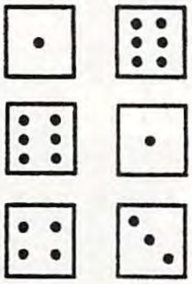
- There are *two* ways to get 3.



1 + 2 = 3

2 + 1 = 3

• There are several ways to get 7.



1 + 6 = 7

6 + 1 = 7

4 + 3 = 7

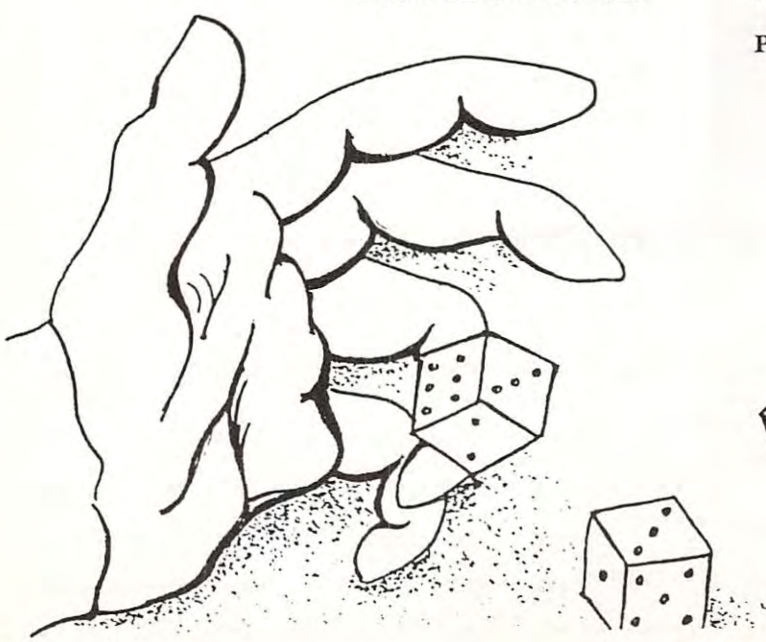
And several more!

Exercise 12. Complete the following table showing the number of different ways to get each *possible* outcome (2 through 12) in rolling two 6-sided dice.

OUTCOME	NUMBER OF WAYS
2	1
3	2
4	5
5	—
6	—
7	—
8	—
9	—
10	—
11	—
12	—

Next, we would like to compute proportions, as defined below (X is any outcome, 2 through 12).

$$\text{PROPORTION FOR OUTCOME} = \frac{\text{NUMBER OF WAYS TO GET OUTCOME X}}{\text{TOTAL NUMBER OF WAYS FOR ALL OUTCOMES}}$$



Or, since the total for all outcomes is 36,

$$\text{PROPORTION FOR OUTCOME X} = \frac{\text{NUMBER OF WAYS TO GET OUTCOME X}}{36}$$

Exercise 13. Complete the following table (use a calculator!)

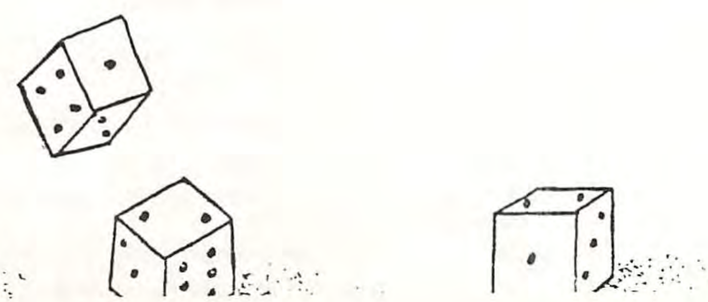
OUTCOME	NUMBER OF WAYS	PROPORTION
2	1	1/36 = .0278
3	2	2/36 = .0556
4	3	3/36 = .0833
5	—	______
6	—	______
7	—	______
8	—	______
9	—	______
10	—	______
11	—	______
12	—	______

If we flip a coin, the *probability* of getting HEADS is ½ = .5. What is the *probability of getting TAILS?*

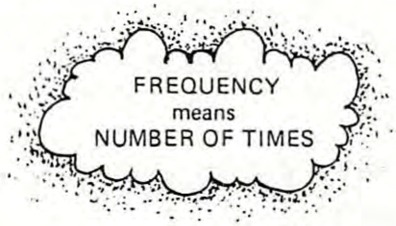
Yes, we are leading up to a heavy exercise. But, you can probably do it!

Exercise 14. Write a program to simulate N rolls of two dice. Don't print the results. Instead, count the number of occurrences of each possible outcome (2 through 12), then print this information *and* also print the proportion of each outcome. Huh? For N rolls, the proportion for outcome X is:

$$\text{PROPORTION FOR OUTCOME X} = \frac{\text{NUMBER OF TIMES X OCCURRED}}{\text{N}}$$



We did it, we wrote the program and ran it. Here is what happened.

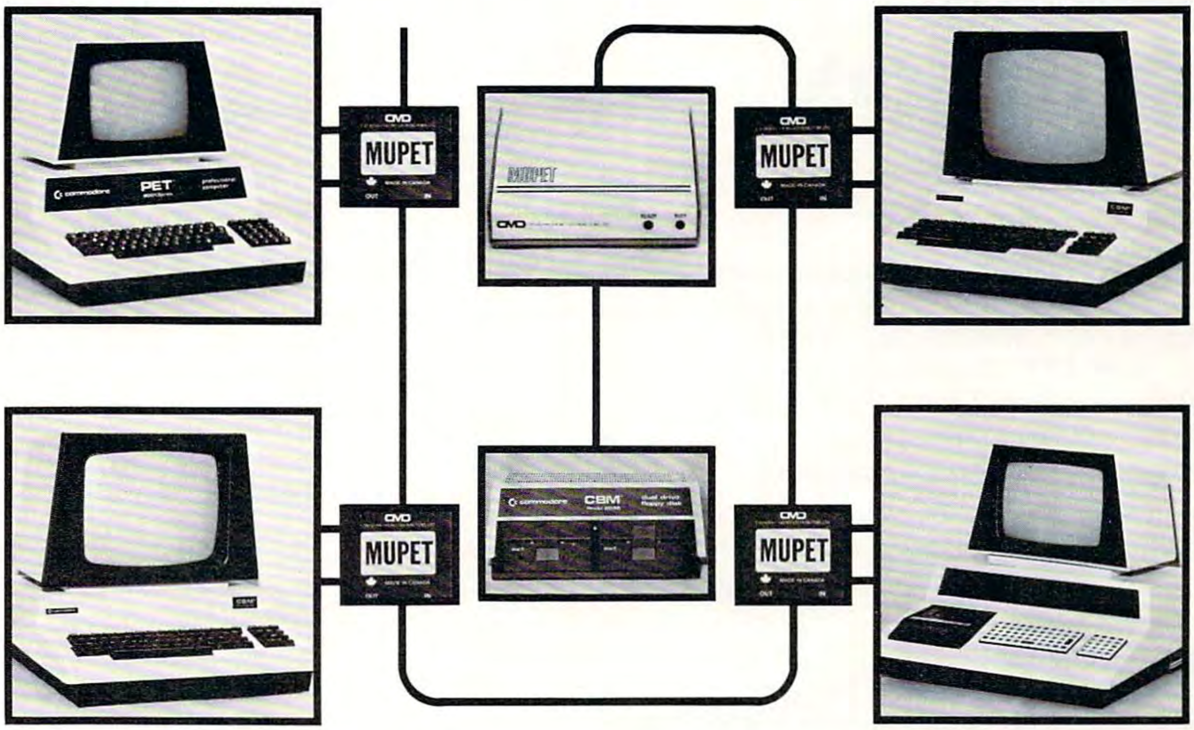


OUTCOME	FREQUENCY	PROPORTION
2	23	.024
3	62	.062
4	81	.081
5	109	.109
6	140	.14
7	142	.142
8	137	.137
99	126	.126
10	80	.08
11	72	.072
12	27	.027

If you have the time, try 10000 rolls, or 20000 rolls, or even 100000 rolls. Compare the proportions with the proportions you wrote down for Exercise 13. Or, compare with our answers for Exercise 13. ©

Now The PET's Know How To Share

MUPET MULTIUSER SYSTEM



EDUCATION

* allows direct access to any programmes or data files
* reduces media cost - one set of disks needed for up to 8 users
* disk based languages such as Assembler, Pascal, Word Processing, available to all users

BUSINESS

* true multi-station wordprocessing with access to common data files
* cost per station drastically reduced
* printer spooling is supported

MANUFACTURING

* using optional DOUBLE MUPET, both local and common IEEE devices are supported
* test equipment can be controlled locally, while reports and programmes are managed universally

MUPET

USES

NO
NONE
NOT ANY
NOT ONE BYTE

OF
SOFTWARE

****

YOU CAN RUN
ANY COMBINATION
OF PROGRAMMES
OR LANGUAGES
AT THE SAME TIME

- word processing
- Pascal
- Assembler
- Future???

* MUPET supports any combination of CBM hardware — present or future — which supports the IEEE bus
* MUPET plugs in externally. No need to open the PET/CBM cabinet
* MUPET supports all existing and future CBM disk based software
* MUPET supports printers as well as disks
* MUPET supports **all** versions of Commodore Basic

PET/CBM Trademark of
Commodore Business Machines

CMD

CANADIAN MICRO DISTRIBUTORS LTD.
365 Main Street, Milton, Ontario L9T 1P7 / 416 878-7277

A CAI program called LINEAR EQUATION

Peter Oakes
Muskegon, MI

This article is about Computer-Assisted-Instruction or commonly called CAI programming. My example program is called LINEAR EQUATION. It is written for an 8K PET computer. Since it uses a minimum of graphics I believe it could be modified for many other small computers.

The Program Has These Features:

choice of using the computer monitor or a printer.
choice of 6 different randomly generated problems.
problem solutions complete with step by step procedures for solving.

Program Description

Lines 100-106 simply announces the program.
Line 108 makes the RND (random number generator) truly random for the "older" original ROM PETs.
Lines 110-122 asks if the user wants to use a printer. If this option is executed, then the problem question and solution (and procedure) will be written to the printer. Everything else is still done on the monitor. Figure 1 shows a sample output for a printer. Of course, a similar output would appear on the monitor if the printer option is not executed.
Lines 124-144 ask for the problem type the user wants generated. Line 144 forces the user to answer only with a 1,2,3,4,5,or 6. A similar control occurs in line 118 making sure the user answers with Y,N, or T.
Lines 146-168 gets the random data to generate the problems. Line 152 generates a random number V(I) in the range [-11.0 to +11.0] excluding [-0.9 to +0.9]
Lines 158-168 calculate specific problem data. Example: X1 = <u>INT</u>(-V(2)/V(1)*<u>100 + .5)/100</u> calculates the x-intercept of a line rounded off to hundredths by the underlined portion of the statement.
Line 172 opens the PET to a device (ie: opens to write to the monitor or printer depending on the value of U8 from lines 114-116). Line 172 also clears the monitor if the printer is not used. Line 174 prints a "divider" between problems if a printer is used. Line 176 will GOTO the printing of the selected problem as does line 250 print the appropriate solution.
Lines 234-250 checks (on the monitor) to see if a solution is wanted.

Note that in the printing of signs care has been taken to print the appropriate - or + sign. An example is found in the subroutine at lines 402-406 (as used from line 264). If T had a value of -7.2, then the subroutine would make T$ be -7.2 whereas if T had a value of 7.2, then T$ would be made +7.2 which assures the correct printing of T$.

The rest of the program lines are unique to what each line does and would take too much space to explain every detail. I'll let the reader read those lines over on his own. I hope this program will be of value to the reader as CAI programs can be very helpful in mathematics. Figure 2 shows a complete listing of the program with graphics noted.

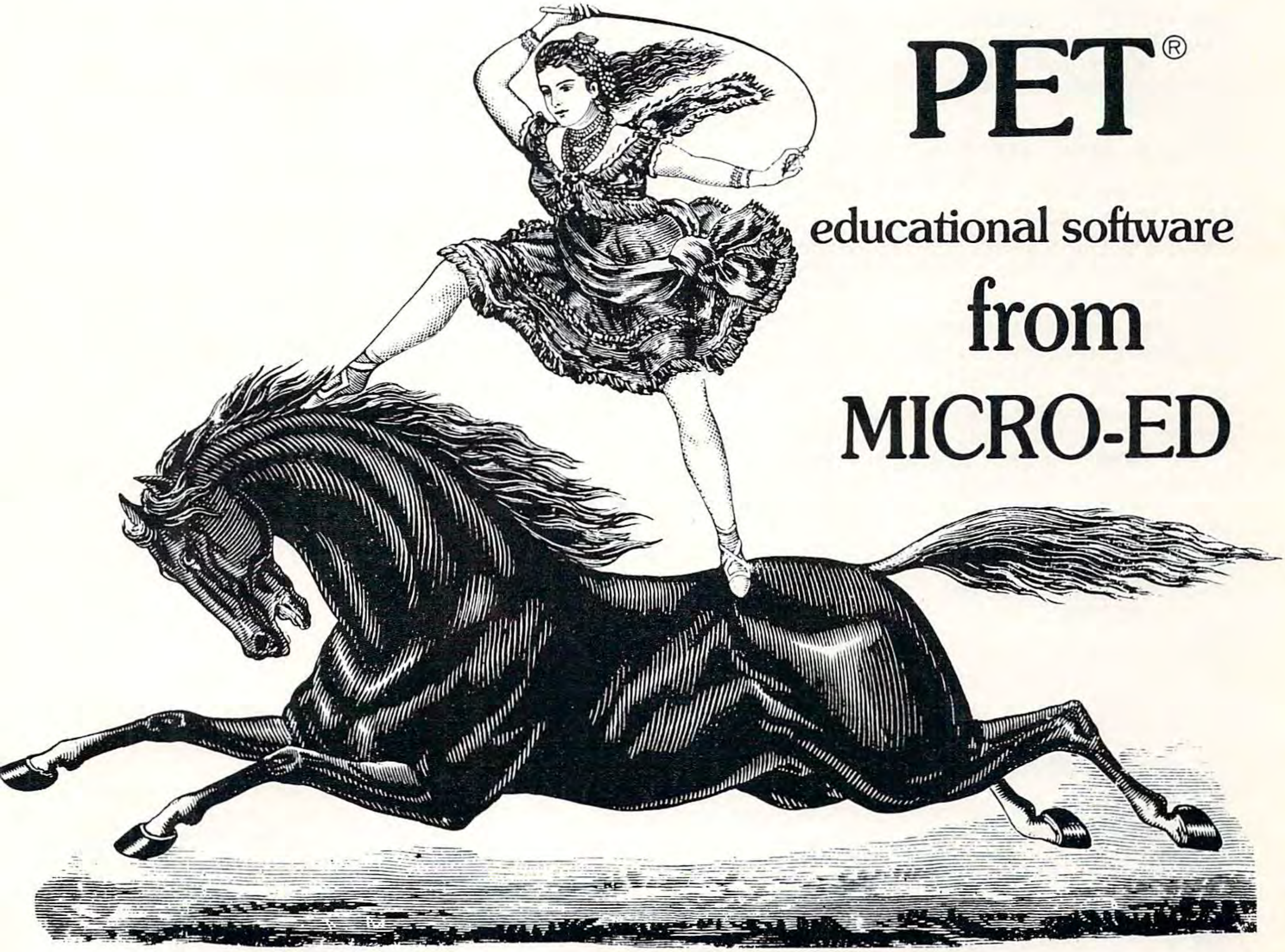
Figure 1

```
====================================

GIVEN: SLOPE =  9.2
       Y-INTERCEPT ( 0,-5.6 )

FIND:  AX+BY+C=0 WITH B=-1
ALSO:  X-INTERCEPT.




USING:  Y - Y1 = M(X - X1)
WHERE   M = SLOPE OF THE LINE
        (X1,Y1) = A POINT ON THE LINE

THEN  Y + 5.6 =  9.2 ( X - 0 )
      Y + 5.6 =  9.2 X
        0 =  9.2 X - Y - 5.6  (EQUATION)



IF Y=0:   0 =  9.2 X - 0 - 5.6
          0 =  9.2 X - 5.6
       -9.2 X = -5.6
          X =  .61

THUS      ( .61 , 0 ) = X-INTERCEPT
```

Figure 2

```
100 PRINT"ĥLINEAR EQUATION":PRINT"PETER ¬
     ¬OAKES,10-1-80,7K
102 PRINT:PRINT"PROGRAM GENERATES ¬
     ¬LINEAR EQUATION
104 PRINT"PROBLEMS AND PROVIDES A ¬
     ¬SOLUTION
106 PRINT"PROCEDURE.
107 :
108 U9=RND(-TI):REM RANDOMIZE RND
109 :
110 :REM USE PRINTER ?
112 PRINT:INPUT"USE PRINTER (Y,N,T)";Q$:
     ¬IFQ$="T"GOTO388
114 IFQ$="Y"THENU8=4:GOTO120
116 IFQ$="N"THENU8=3:GOTO126
118 GOSUB392:GOTO112
120 PRINT:PRINT"WHEN PRINTER IS READY - ¬
     ¬PRESS ɼSPACEr̂ KEY
122 GETQ$:IFQ$=""GOTO122
123 :
124 :REM PROB CHOICE
126 PRINT"ĥWANT TO SOLVE A LINEAR ¬
     ¬EQUATION GIVEN
128 PRINT" 1. SLOPE & Y INTERCEPT
130 PRINT" 2. SLOPE & A POINT
```

Perfectly Balanced



PET®

educational software from MICRO-ED

More than 200 instructional programs

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

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

Send for free catalogue:

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

or telephone us at (612) 926-2292

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

```
132 PRINT" 3. TWO POINTS
134 PRINT" 4. X & Y INTERCEPTS
136 PRINT" 5. PARALLEL LINE & A POINT
138 PRINT" 6. PERPENDICULAR LINE & A ¬
    ¬POINT
14Ø PRINT" T. TERMINATE THE PROGRAM
142 PRINT:INPUT"WANT TYPE: 1,2,3,4,5,6,
    ¬OR T";Q$:IFQ$="T"GOTO388
144 N=VAL(Q$):IFN<1ORN>6ORINT(N)<>NGOTO1
    ¬42
145 :
146 :REM DATA -1ØØ<V(I)<1ØØ; V(1)=M & ¬
    ¬V(2)=B IN Y=MX+B; V(3) & V(4)=X-CO
    ¬ORD'S
148 FOR I=1 TO 4
152 V(I)=INT(RND(1)*1ØØ+.5)/1Ø+1:
    ¬IF RND(1)>.5 THEN V(I)=-V(I)
154 NEXT I
155 :
156 :REM ASSIGN VARIABLES
158 X1=INT(-V(2)/V(1)*1ØØ+.5)/1ØØ:
    ¬REM X-INTERCEPT
16Ø Y1=INT((V(1)*V(3)+V(2))*1ØØ+.5)/1ØØ:
    ¬REM Y-COORD AT A POINT
162 Y2=INT((V(1)*V(4)+V(2))*1ØØ+.5)/1ØØ:
    ¬REM Y-COORD AT ANOTHER POINT
164 B1=INT((V(4)-V(1)*V(3))*1E2+.5)/1E2:
    ¬REM Y-INTERCEPT OF PARALLEL SYSTEM
166 M2=INT((-1/V(1))*1ØØ+.5)/1ØØ:
    ¬REM SLOPE FOR PERPENDICULAR SYSTEM
168 B2=INT((V(4)-M2*V(3))*1ØØ+.5)/1ØØ:
    ¬REM Y-INTERCEPT OF PERPENDICULAR ¬
    ¬SYSTEM
169 :
17Ø :REM WRITE PROB
172 OPEN1,U8:CMD1:IFU8=3THENPRINT"ĥ":
    ¬GOTO176
174 PRINT"==============================
    ¬=========
176 ONNGOTO18Ø,194,2Ø6,216,226,232
177 :
18Ø :REM #1:M=V(1),B=V(2),X1=X-INTERCEPT
182 PRINT"GIVEN: SLOPE = "V(1)
184 PRINT"       Y-INTERCEPT ( Ø,"V(2)")
186 PRINT:PRINT"FIND:  AX+BY+C=Ø WITH ¬
    ¬B=-1
188 PRINT"ALSO:  X-INTERCEPT.":GOTO236
189 :
192 :REM #2:M=V(1),B=V(2),X1=X-INTERCEPT
    ¬ & POINT (V(3),Y1)
194 PRINT"GIVEN: SLOPE = "V(1)
196 PRINT"       F("V(3)") = "Y1
198 PRINT:PRINT"FIND:  Y=MX+B
2ØØ PRINT"ALSO:  X & Y INTERCEPTS.":
    ¬GOTO236
2Ø1 :
2Ø4 :REM #3:M=V(1),B=V(2) & POINTS:
    ¬ (V(3),Y1) & (V(4),Y2)
2Ø6 PRINT"GIVEN THE POINTS:  ("V(3)",
    ¬"Y1")
2Ø8 PRINT"                   ("V(4)",
    ¬"Y2")
21Ø PRINT:PRINT"FIND:  Y=MX+B":GOTO236
212 :
214 :REM #4:M=V(1),B=V(2),X1=X-INTERCEPT
216 PRINT"GIVEN: Y-INTERCEPT ( Ø,"V(2)")
218 PRINT"       X-INTERCEPT ("X1",Ø )
22Ø PRINT:PRINT"FIND:  Y=MX+B":GOTO236
222 :
224 :REM #5:M=V(1),B=V(2) & POINT:
    ¬ (V(3),V(4))
226 M$="PARALLEL":GOSUB432:GOTO236
228 :
23Ø :REM #6:M=V(1),B=V(2) & POINT:
    ¬ (V(3),V(4))
232 M$="PERPENDICULAR":GOSUB432
233 :
234 :REM WANT SOLUTION ?
236 PRINT:PRINT:PRINT#1:CLOSE1
238 PRINT:INPUT"WANT SOLUTION (Y,N,
    ¬T)";Q$:IFQ$="T"GOTO388
24Ø IFQ$="Y"GOTO246
242 IFQ$="N"GOTO378
244 GOSUB392:GOTO238
245 :
246 :REM WRITE SOLUTION
248 OPEN1,U8:CMD1:IFU8=3THENPRINT"ĥ
25Ø ONNGOTO254,276,294,318,34Ø,356
251 :
252 :REM #1
254 GOSUB41Ø
256 PRINT:PRINT"THEN  Y ";:T=V(2):
    ¬GOSUB398:PRINT T$" = "V(1)"( X - ¬
    ¬Ø )
258 PRINT"       Y "T$" = "V(1)"X
26Ø PRINT"       Ø = "V(1)"X - Y ";:
    ¬T=V(2):GOSUB4Ø4:PRINT T$;
262 PRINT"  (EQUATION)":PRINT:PRINT
264 PRINT:PRINT"IF Y=Ø:  Ø = "V(1)"X - ¬
    ¬Ø ";:T=V(2):GOSUB4Ø4:PRINT T$
266 PRINT"          Ø = "V(1)"X "T$
268 PRINT"         ";-V(1)"X = "V(2)
27Ø PRINT"          X = "X1
272 PRINT:PRINT"THUS      ("X1", Ø ) = ¬
    ¬X-INTERCEPT":GOTO376
273 :
274 :REM #2
276 GOSUB41Ø:PRINT
278 T=Y1:M=V(1):GOSUB418
28Ø PRINT:PRINT:PRINT"IF X=Ø:  Y = ¬
    ¬"V(1)"(Ø) "T$" = "T$
282 PRINT"         ( Ø ,"T$") = ¬
    ¬Y-INTERCEPT
284 PRINT:PRINT:PRINT"IF Y=Ø:  Ø = ¬
    ¬"V(1)"X "T$
286 PRINT"         ";T" = "V(1)"X
288 PRINT"         "X1" =  X
29Ø PRINT"         ("X1", Ø ) = ¬
    ¬X-INTERCEPT":GOTO376
291 :
292 :REM #3
294 PRINT"SLOPE = M = (Y1-Y2)/(X1-X2)
296 PRINT"        = ("Y1;:T=Y2:GOSUB398:
    ¬PRINT T$")/("V(3);
298 T=V(4):GOSUB398:PRINT T$")
3ØØ PRINT"        = "V(1)
3Ø2 PRINT:PRINT"THUS IN THE EQUATION:
    ¬  Y =   MX  +  B
3Ø4 PRINT"                       Y = ¬
    ¬"V(1)"X + B
3Ø6 PRINT:PRINT:PRINT"THEN  "Y2" = ¬
    ¬"V(1)"("V(4)") + B
3Ø8 PRINT"      "Y2" = ";INT((V(1)*V(4))
    ¬*1ØØ+.5)/1ØØ;" + B
31Ø PRINT"      "V(2)" =  B
312 PRINT:PRINT:PRINT"THUS THE EQUATION ¬
    ¬IS
314 PRINT:PRINT"        Y = "V(1)"X ";:
    ¬T=V(2):GOSUB4Ø4:PRINT T$:GOTO376
315 :
316 :REM #4
318 PRINT"USING:  Y = MX + B
```

```
320 PRINT"AND       ( 0 ,"V(2)")
322 PRINT:PRINT"THEN     "V(2)" = M(0) + ¬
     ¬B
324 PRINT"           "V(2)" =  B
326 PRINT:PRINT:PRINT"NOW USING:
     ¬  ("X1", 0 )
328 PRINT"IN              Y = MX";:T=V(2):
     ¬GOSUB404:V2$=T$:PRINT V2$
330 PRINT:PRINT"                0 = M("X1") ¬
     ¬"V2$
332 PRINT"          ";:T=X1:GOSUB398:
     ¬PRINT T$"M = "V2$
334 PRINT"           M = "V(1)
336 PRINT:PRINT:PRINT"EQUATION:   Y = ¬
     ¬"V(1)"X "V2$:GOTO376
337 :
338 :REM #5
340 PRINT"REWRITE  "V(1)"X - Y "T$" = 0
342 PRINT"AS           Y = "V(1)"X "T$
344 PRINT:PRINT:PRINT"THEN SINCE ¬
     ¬PARALLEL LINES HAVE
346 PRINT"EQUAL SLOPES TOGETHER WITH ¬
     ¬THE GIVEN
348 PRINT"POINT:  ("V(3)","V(4)")  AND
350 PRINT:GOSUB410:PRINT
352 T=V(4):M=V(1):GOSUB418:GOTO376
353 :
354 :REM #6
356 PRINT"REWRITE  "V(1)"X - Y "T$" = 0
358 PRINT"AS          Y = "V(1)"X "T$
360 PRINT:PRINT:PRINT"THEN SINCE ¬
     ¬PERPENDICULAR LINES HAVE
362 PRINT"SLOPES THAT ARE NEGATIVE ¬
     ¬RECIPROCALS
364 PRINT"THEN M = -1/("V(1)") = "M2
366 PRINT:PRINT"THUS TOGETHER WITH THE ¬
     ¬GIVEN
368 PRINT"POINT:  ("V(3)","V(4)")  AND
370 PRINT:GOSUB410:PRINT
372 T=V(4):M=M2:GOSUB418
373 :
374 :REM ANOTHER PROB ?
376 PRINT:PRINT:PRINT#1:CLOSE1
378 PRINT:INPUT"ANOTHER PROBLEM (Y,
     ¬N)";Q$:IFQ$="N"GOTO388
380 IFQ$="Y"GOTO384
382 PRINT:GOSUB394:GOTO378
384 IFU8=3THENPRINT"ĥ
386 GOTO124
388 END
389 :
390 :REM SUBROUTINES
391 :REM USE (Y,N,T)
392 PRINT:PRINT"USE T TO TERMINATE THE ¬
     ¬PROGRAM OR"
394 PRINT"USE Y FOR YES OR N FOR NO!":
     ¬RETURN
395 :
396 :REM T-->"-T",T>0 OR T-->"+T",T<=0
398 T$="-"+STR$(T):IF T<0 THEN T$="+"+ST
     ¬R$(ABS(T))
400 RETURN
401 :
402 :REM T-->"+T",T>0 OR T-->"-T",T<=0
404 T$="+"+STR$(T):IF T<0 THEN T$="-"+ST
     ¬R$(ABS(T))
406 RETURN
407 :
408 :REM PROB HEADER
410 PRINT"USING:  Y - Y1 = M(X - X1)
412 PRINT"WHERE    M = SLOPE OF THE LINE
414 PRINT"          (X1,Y1) = A POINT ON ¬
     ¬THE LINE":RETURN
415 :
416 :REM WRITE SOLUTION
418 PRINT"         Y ";:T1=T:GOSUB398:
     ¬Y1$=T$:PRINT Y1$" = "M"( X ";
420 T=V(3):GOSUB398:PRINT T$" )
422 PRINT"         Y "Y1$" = "M"X ";:
     ¬T=INT(M*V(3)*100+.5)/100:GOSUB398
424 PRINT T$
426 PRINT:PRINT"          Y = "M"X ";:
     ¬T=INT((-T1+T)*100+.5)/100:GOSUB398
428 PRINT T$"  (EQUATION)":RETURN
429 :
430 :REM WRITE PROB
432 PRINT"FIND:  Y=MX+B THAT IS "M$" TO
434 PRINT:PRINT"              "V(1)"X - Y ";:
     ¬T=V(2):GOSUB404:PRINT T$" = 0
436 PRINT:PRINT"          CONTAINING  ¬
     ¬("V(3)","V(4)")":RETURN
```

Sample Execution

```
======================================
GIVEN: SLOPE = -1.6
       Y-INTERCEPT ( 0,-7.9 )

FIND:  AX+BY+C=0 WITH B=-1
ALSO:  X-INTERCEPT.



USING:  Y - Y1 = M(X - X1)
WHERE   M = SLOPE OF THE LINE
        (X1,Y1) = A POINT ON THE LINE

THEN  Y + 7.9 = -1.6 ( X - 0 )
      Y + 7.9 = -1.6 X
      0 = -1.6 X - Y - 7.9  (EQUATION)


IF Y=0:  0 = -1.6 X - 0 - 7.9
         0 = -1.6 X - 7.9
         1.6 X = -7.9
         X = -4.94

THUS     (-4.94 , 0 ) = X-INTERCEPT
```

```
======================================
GIVEN: SLOPE =  10.3
       F( 1.8 ) =  17.04

FIND:  Y=MX+B
ALSO:  X & Y INTERCEPTS.



USING:  Y - Y1 = M(X - X1)
WHERE   M = SLOPE OF THE LINE
        (X1,Y1) = A POINT ON THE LINE

        Y - 17.04 =  10.3 ( X - 1.8 )
        Y - 17.04 =  10.3 X - 18.54

        Y =  10.3 X - 1.5  (EQUATION)


IF X=0:  Y =  10.3 (0) - 1.5 = - 1.5
         ( 0 ,- 1.5) = Y-INTERCEPT


IF Y=0:  0 =  10.3 X - 1.5
          1.5  =  10.3 X
          .15  =  X
         ( .15 , 0 ) = X-INTERCEPT
```

```
======================================
GIVEN THE POINTS:  ( 1.2 , 12.62 )
                   (-2.9 ,-10.34 )

FIND:  Y=MX+B


SLOPE = M = (Y1-Y2)/(X1-X2)
      = ( 12.62 + 10.34)/( 1.2 + 2.9)
      =  5.6

THUS IN THE EQUATION:  Y =   MX  +  B
                       Y =  5.6 X + B


THEN  -10.34  =  5.6 (-2.9 ) + B
      -10.34  = -16.24  + B
       5.9  =  B


THUS THE EQUATION IS

       Y =  5.6 X + 5.9
```

```
======================================
GIVEN: Y-INTERCEPT ( 0,-10.6 )
       X-INTERCEPT ( 7.07 ,0 )

FIND:  Y=MX+B


USING:  Y = MX + B
AND     ( 0 ,-10.6 )

THEN     -10.6  = M(0) + B
         -10.6  =  B


NOW USING:  ( 7.07 , 0 )
IN          Y = MX- 10.6

            0 = M( 7.07 ) - 10.6
       - 7.07M = - 10.6
            M =  1.5


EQUATION:   Y =  1.5 X - 10.6
```

```
======================================
FIND:  Y=MX+B THAT IS PARALLEL TO

          1.9 X - Y - 8.9 = 0

       CONTAINING  (-2.6 ,-5 )



REWRITE   1.9 X - Y - 8.9 = 0
AS          Y =  1.9 X - 8.9

THEN SINCE PARALLEL LINES HAVE
EQUAL SLOPES TOGETHER WITH THE GIVEN
POINT:  (-2.6 ,-5 )  AND

USING:  Y - Y1 = M(X - X1)
WHERE   M = SLOPE OF THE LINE
        (X1,Y1) = A POINT ON THE LINE

        Y + 5 =  1.9 ( X + 2.6 )
        Y + 5 =  1.9 X + 4.94

        Y =  1.9 X - .06  (EQUATION)
```

```
======================================
FIND:  Y=MX+B THAT IS PERPENDICULAR TO

          -6.1 X - Y + 5.1 = 0

       CONTAINING  ( 2.1 , 6.8 )



REWRITE  -6.1 X - Y + 5.1 = 0
AS         Y = -6.1 X + 5.1


THEN SINCE PERPENDICULAR LINES HAVE
SLOPES THAT ARE NEGATIVE RECIPROCALS
THEN M = -1/(-6.1 ) =  .16

THUS TOGETHER WITH THE GIVEN
POINT:  ( 2.1 , 6.8 )  AND

USING:  Y - Y1 = M(X - X1)
WHERE   M = SLOPE OF THE LINE
        (X1,Y1) = A POINT ON THE LINE

        Y - 6.8 =  .16 ( X - 2.1 )
        Y - 6.8 =  .16 X - .34

        Y =  .16 X + 6.46  (EQUATION)
```

Hex Conversion

Using The 6502's Decimal Mode

Jack Clarke

Since the advent of 8 bit microprocessors, the hexidecimal numbering system has been around to help provide a shorthand notation for binary numbers... remember 4 binary bits can be expressed with just 1 hexadecimal character? ($F = 1111_2$)

While this shorthand notation has revolutionized Assembly Language coding, undoubtedly many a new computerist has cursed the notation as problematical, confusing and cumbersome.

To assist the programmer (old and new), elaborate tables have been generated to convert the *radix* of a number from one base to another... remember radix and base are synonymous? To further the cause of this translation, numerous programs have been written in higher level languages. Take a look at Texas Instrument's hand-held "Programmer" which has gained a commendable respect in the programming community. Have you ever tried to poke or peek with your Apple without one of the above?

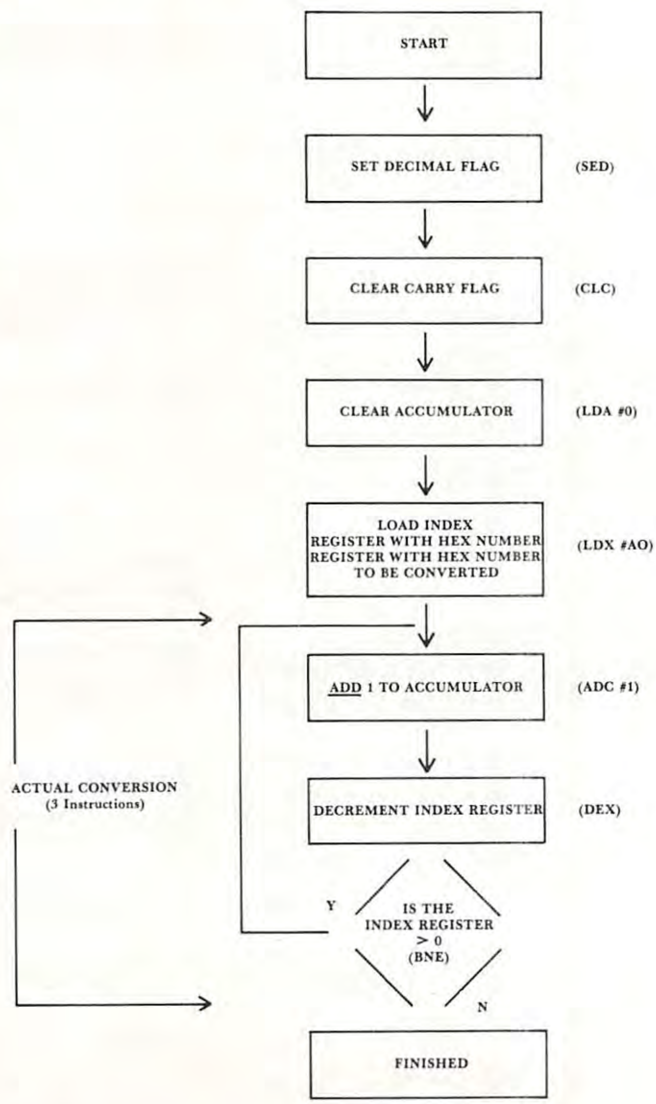
What is this decimal mode you ask? Simply defined it is a clever bit of binary manipulation that is performed inside the microprocessor to insure that when you add, a "1" to a "9" that the result is "Ø with carry" and not "A", (also known as BCD coding). In other words, 4 binary bits can express a decimal number 0 thru 9, (10 thru 15 is illegal). So an eight bit number provides numbering 0 thru 99.

Now, let's take a closer look at the 6502's instruction set and see how the *decimal* mode can help with this numbering conversion.

A "bit" of examination reveals that the decimal mode *only* works when performing an add (ADC) or subtract (SBC) instruction. All other instructions simply ignore the decimal mode. Take for example the increment/decrement instruction. It performs an addition or subtraction (by one) but always in the binary/hex mode. Now, what would happen if we combined a decrement/increment instruction with an add/subtract instruction. The increment instruction would count up one in hexadecimal while the add instruction would simultaneously count up in decimal...did I just see a hex to decimal conversion go by?

How about an example? Suppose you wish to convert the hex number "A0" to the equivalent decimal number. (Don't pull out your conversion tables yet). Follow the flow chart in Figure 1 and walk through the steps. First set the decimal mode (SED), clear the accumulator (LDA Ø IMM) and clear the carry flag (CLC). Next, load the x-register with the hex number to be converted (LDX A0 IMM). Now, the conversion starts. Decrement the x-register (DEX) and test for zero (BNE). If the x-register is >0 then add 1 to the accumulator (ADC 1 IMM). Repeat the sequence until the x-register has counted down to 0. When you examine the contents of the accumulator you will find the *decimal* equivalent of "A0" sitting there quietly. If you need a hex equivalent of a decimal number you would enter the decimal number in the accumulator and *subtract* one in the decimal mode...each time you subtract you would also increment the x-register. See any similarities?

For numbers greater than 99 you would perform the addition or subtraction using two or more memory locations and keep track of the carry flag, (double precision arithmetic). The X and Y registers could also be cascaded for extended range with 16 bits. Conversion of 0000 thru FFFF could be easily implemented.



Hex To Decimal Conversion

The best way to familiarize yourself with this type of approach is to try it on your own computer. After gaining a little confidence in the ease of the conversion, you will soon find the same techniques incredibly helpful in more complex operations such as multiplication and division. Take the example of a program that is sampling the rate of an asynchronous input... By knowing the "sample time" of your program (each time you read the port) and adding that *constant* instead of "1" you effectively convert and multiply in one operation resulting in a decimal formatted "total number of samples"

To summarize the concept of radix conversion using the 6502's decimal mode, start with zero in the accumulator and index register and add "1" to the accumulator (decimal mode) and increment the index register at the same time. You will observe the accumulator counting up in decimal and the index register counting up in binary/hex.

Say good-bye to those dog-eared tables and long involved conversion programs that you have been using. The 6502 takes another bow. ©

Are you using your computer in an interesting application?

Write about it for

COMPUTE!

THE apple® GAZETTE

Clearing The Apple II Low-Resolution Graphics Screen

Sherm Ostrowsky

Many applications require rapidly clearing the low-resolution graphics screen to black (COLOR = 0) or to some other color. In the latter case the process might be more accurately described as ''back-grounding''. Either way, this apparently simple operation can be done by several different methods. Each method will produce a distinctly different visual effect while in operation, although the end result will be the same. By doing the experiments to be described below, the experienced programmer can learn how to use the method best suited to his immediate purpose, and the novice programmer can learn some useful facts about the operation of the Apple low-resolution graphics. So go ahead and do the experiments on your Apple; you can't hurt it by pushing the keys (even the wrong keys), and you can learn a lot.

First of all, in order to see the effect of any kind of screen-clearing method it is best to begin with a screen that is loaded with colors and forms. You may do this in any way that pleases you; I have been using the following subroutine in Applesoft:

```
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 FOR PAUSE = 0 TO 2000: NEXT PAUSE
1070 RETURN
```

Notice that this subroutine colors-in the so-called ''mixed screen'' — the top 40 lines, but not the bottom part reserved for text. If you wish to use, and color-in, the whole screen (48 graphics lines), then the first two lines of the Applesoft subroutine can be amended to:

```
1000 POKE -16302,0 : POKE -16304,0
1010 FOR I = 0 TO 47
```

etc. The line of POKEs turns on the ''soft switches'' governing the full-screen lo-res graphics (see pages 12-13 in the new Apple II reference manual).

Now that the screen is colored, let's clear it. The first method which is likely to occur to the average programmer is to write a couple of lines in Applesoft. Suppose you want to clear the screen to a particular background color, say C (C = 0 to 15). A program to do this for a mixed screen might look like this:

```
10 GOSUB 1000 : REM PAINT THE SCREEN
20 COLOR = C
30 FOR I = 0 TO 39
40 VLIN 39,0 AT I
50 NEXT I
60 END
```

Try it. The screen clears rather ponderously, like a stage curtain rolling across from left to right. If you want the curtain to move from right to left, just change line 30 to

```
30 FOR I = 39 TO 0 STEP -1
```

If you want it to operate on whole-screen graphics, line 40 should be altered to

```
40 VLIN 47,0 AT I
```

This method works fine, if you don't mind the relatively slow speed of the clearing operation. In fact, for some special effects it might even be preferred. Notice how you can control the direction of motion of the apparently rolling curtain. As an ''exercise for the student'', consider how you might change lines 30 and 40 so as to cause the curtain to appear to be rising upwards. That can be a rather pretty effect, especially if you don't just leave a blank screen but instead ''paint-in'' a scene of some kind to coincide with the rising of the curtain (i.e., one horizontal line at a time, from bottom to top); it can look like a real stage curtain rising to reveal a scene already in place.

But what if you are not satisfied with the relatively slow speed with which an Applesoft program can clear the screen? If you don't mind being restricted to just a basic black clear, there are some dandy machine-language subroutines in the Apple's built-in ROM Monitor which are a lot faster. For mixed-screen graphics, try this little program:

```
10 GOSUB 1000 : REM PAINT SCREEN
20 CALL -1994
30 END
```

That's not only a heck of a lot faster, but pretty simple to use, too! If you're doing full-screen graphics, replace line 20 with

```
20 CALL -1998
```

ON-LINE *systems*

IS PROUD TO ANNOUNCE A NEW ADDITION TO OUR FAMILY!

SuperScribe

(Formerly SuperScript)

SuperScribe is the most powerful and easiest to use Word Processor available for your Apple II or II Plus computer. Besides leaping tall buildings in a single bound it:

- Gives true upper/lower case text on your screen with no additional hardware whatsoever.
- Works with documents larger than the amount of memory in your Apple - transparently to you!
- Edit not only letters but also any text or binary file, or even basic programs!
- Automatically generates up to 4 separate indices for your document!
- Save typing time through a unique ability to designate specified keys as commonly used words, phrases or even commands!
- Globally search for or replace character strings.
- SuperScribe has a built-in instruction capability such that if you forget how to use a command and the manual is not close by - you may simply ask SuperScribe!
- Supports multiple disk drives!
- Will support alternate character sets.
- Produces form letters using address files easily!
- Supports the shift key modification if made to your Apple.
- Lets you work with your text on a screen at a time basis - reducing typos and allowing you to see your document as you edit it.
- Works with any printer!
- Supports the language card or any 16K expansion Ram card to keep more of your document readily available in memory.

SuperScribe is 100% machine language and requires only a 48K Apple II or II Plus with a disk drive. It may be purchased through your local computer store or direct from us by sending $89.95 plus $2.50 to cover shipping to:

ON-LINE SYSTEMS - 36575 Mudge Ranch Road - Coarsegold, CA 93614 - 209-683-6858 C.O.D., Master Charge or Visa accepted.

Very neat. But this way you have no control over the direction of motion of the curtain, nor over the color to which the screen is cleared. Perhaps for your particular application neither of these restrictions makes any difference, in exchange for the very real advantages in speed and simplicity.

If you'd like to have your cake and eat it too, this can be arranged by POKEing a short machine-language subroutine into memory. Then you will be able to select your background color and still retain the speed advantage of the Monitor subroutine. You don't have to know anything about machine-language to do this, although for those who are curious I'll explain how it works in a few minutes. For the moment, just try the Applesoft program below:

```
10 GOSUB 1000 : REM PAINT SCREEN
20 FOR I = 768 TO 782 : REM POKE M/L SUB
30 READ J : POKE I,J
40 NEXT I
50 COLOR = C : REM YOUR CHOICE OF COLOR
60 CALL 768 : REM CALL THE SUBROUTINE
70 END
200 DATA 160, 39, 132, 45, 160, 39, 169, 0, 32, 40, 248,
  136, 16, 248, 96
```

For full-screen graphics, replace the second number in the DATA statment ("39") by the number "47".

If you RUN this program you'll see that it works just like the Monitor version, except that now the screen clears to the selected color, C, instead of only to black (C = 0). It should perhaps be pointed-out that once you have POKEd this subroutine into the computer by executing lines 20 through 40, you can CALL it any number of times in your program without having to POKE it in again. Lines 20 - 40 only have to appear and be executed once in each session at the computer.

Although quite fast, this screen-clearing operation is by no means instantaneous: you can still perceive a curtain-like movement across the screen. What if that's not good enough? I recently wrote a game program in which I wanted the screen to flash suddenly white, to indicate that an enemy torpedo had broken through my screens and wiped me out. Even the machine-language routines are too slow to make a believable explosion flash — an instantaneous white-out. Well, this can in fact be done with the help of a somewhat longer machine-language subroutine which I will now describe. And if you're not into writing game programs, you might still like to be able to clear your screen instantaneously to provide nice sharp transitions from one scene to the next.

The new program looks like this:

```
70 FOR PAUSE = 0 TO 2000 : NEXT PAUSE
80 GOSUB 1000 : REM REPAINT SCREEN
90 FOR I = 800 TO 844 : REM NEW SUB
100 READ J : POKE I,J
110 NEXT I
120 COLOR = C
130 CALL 800 : REM CALL NEW SUBROUTINE
140 END
300 DATA 165, 48, 160, 120, 32, 45, 3, 160, 80, 32, 61,
    3, 96, 136, 153, 0, 4, 153, 128, 4, 153, 0, 5, 153, 128, 5
310 DATA 208, 241, 96, 136, 153, 0, 6, 153, 128, 6, 153, 0,
    7, 153, 128, 7, 208, 241, 96
```

For full-screen graphics, replace the ninth number in DATA statement 300 ("80") by the number "120"

As before, once this new subroutine has been POKEd into memory it can be CALLed whenever you need it without having to rePOKE it (unless, of course, you happen to overwrite it in the meanwhile). This subroutine has been deliberately placed into different memory locations than the previous one, so they can coexist in your computer. Furthermore, the Applesoft routines associated with these two different methods were written in such a way that when both have been typed into your computer as indicated, they will run consecutively. When you type RUN, the screen first fills up with colors, pauses for a few seconds, and then is erased by the first machine-language subroutine. Then the screen fills up with a new random color pattern, pauses, and is suddenly cleared by the second subroutine. The speed difference between these two subroutines is readily apparent in operation.

Each of the several different screen-clearing methods which have been described above has its own special properties; they are all useful additions to your programming arsenal.

Now, for those who are interested, let me briefly discuss the functioning of the two machine-language subroutines. I will assume that the reader is at least somewhat familiar with 6502 Assembly Language and its standard notation.

The first subroutine, starting at location 768 decimal (equivalent to $0300 in hexidecimal) is just a very slightly altered version of the Monitor's routine which we used earlier by CALLing -1994. The Monitor version clears the screen by drawing vertical black lines one after another, exactly as we did it in our very first Applesoft program. The difference in speed between these routines simply reflects the well-known speed advantage of machine-language over Basic. Since the Monitor's version only paints in one color — black — it was changed to permit the color to be an input variable using the standard Applesoft COLOR = C instruction to define which one you want. In Assembler notation, this subroutine looks like this:

```
$0300: A0 27     BKGRND LDY #$27      ; Maximum Y for mixed-screen
                                        clear
 0302: 84 2D            STY V2        ; Store as line-bottom coordinate
 0304: A0 27            LDY #$27      ; Rightmost X-coord (column)
 0306: A9 00     CLRSCR LDA #$00      ; Will start clearing at top
 0308: 20 28 F8         JSR VLINE     ; Jump to line-drawing subroutine
 030B: 88               DEY           ; Next leftmost X-coord (column)
 030C: 10 F8            BPL CLRSCR    ; Loop until done
 030E: 60               RTS           ; Done. Return
```

For full-screen graphics, the number "27" in location $0301 is replaced by the (hexidecimal) number "2F".

The alert reader may have noticed that the color to be used did not appear anywhere in this

subroutine. In fact, the Applesoft statement COLOR = C automatically stores the appropriate color constant in location $30 (decimal 48), where the Monitor routine VLINE can get at it. VLINE draws a single vertical line of the specified color.

Now, the flash-clear subroutine beginning at location 800 decimal ($0320 hexidecimal) works by taking advantage of the "memory-mapped" nature of the Apple's low-resolution screen. Each of the 1600 screen positions on the mixed screen or the 1920 screen positions on the whole screen is defined by a specific half-byte (four bits, or one "nybble") in memory. Since these four bits can represent one of sixteen different hex numbers ($0 through F), each screen position will have one of sixteen different colors depending on how the defining nybble has been set. The two nybbles in each byte define the color for two screen positions in the same column but consecutive rows, that is, two vertically-stacked colored squares. To color a given square it is only necessary to find its corresponding nybble and set it to the appropriate value.

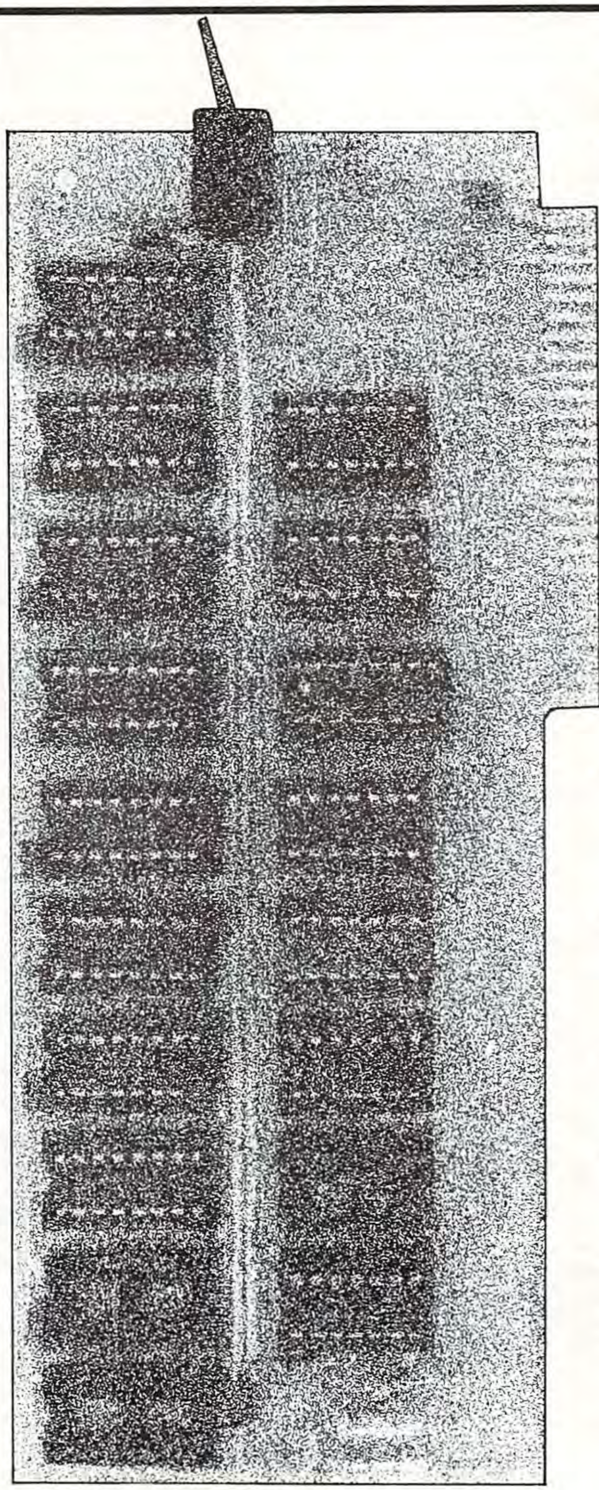
Unfortunately, for some reason the Apple designers didn't arrange the memory locations in any simple consecutive fashion to correspond to the screen rows in numerical order. It requires a special algorithm to find the byte which represents the first square of each row; all the rest of the squares in that row will be represented by consecutive bytes after that. To further complicate matters, the last eight bytes in every 128 bytes do not correspond to any screen positions at all, but rather are used as "scratchpad" memory for whatever devices might be in the motherboard slots.

This last little detail makes the required subroutine for clearing the screen much more complicated than it would otherwise have to be. It is necessary to take the byte in location $30, which represents the chosen color nybble repeated twice, and store it in each byte of screen memory, being careful not to disturb those special bytes which are possibly being used as scratchpad. The address of the first and last effective byte of each row in screen memory has to be known in advance in order to perform this operation in the fastest possible time, without taking time to compute these addresses during the operation. All this has been done in the algorithm represented by the assembly-language subroutine below:

```
$0320: A5 30      FLASH   LDA COLOR  ; Get selected color byte
 0322: A0 78              LDY #$78   ; Prepare to fill 120 bytes
 0324: 20 2D 03           JSR FILL1  ; Fill four sets of 120 bytes each
 0327: A0 50              LDY #$50   ; Prepare to fill 80 bytes
 0329: 20 3D 03           JSR FILL2  ; Fill four sets of 80 bytes each
 032C: 60                 RTS        ; Done. Return.
      ; Subroutine FILL1 puts the selected color byte into
      ; each of four sets of 120 consecutive screen-memory
      ; bytes, being careful to avoid the scratchpad bytes at
      ; the end of each set.

 032D: 88         FILL1   DEY
 032E: 99 00 04           STA $400, Y
 0331: 99 80 05           STA $480, Y
 0337: 99 80 05           STA $500, Y
 033A: D0 F1              STA $580, Y
 033C: 60                 BNE FILL1
                          RTS
      ; Subroutine FILL2 puts the selected color byte into each
      ; of four sets of 80 consecutive screen-memory bytes.
      ; These are the "short lines", leaving out at the end of
      ; each one of the four text lines at the bottom of the
      ; mixed screen.

 033D: 88         FILL2   DEY
 033E: 99 00 06           STA $600, Y
 0341: 99 80 06           STA $680, Y
 0344: 99 00 07           STA $700, Y
 0347: 99 80 07           STA $780, Y
 034A: D0 F1              BNE FILL2
 034C: 60                 RTS
```

For full-screen graphics, the "short lines" of the subroutine FILL2 become full-length lines as in FILL1, which is accomplished simply by changing the constant "$50" in location $0328 to a "$78".

And that's how we clear the screen in a flash. But before I quit, I'd like to leave you with one more little idea. If, instead of setting the color byte by an Applesoft line of the form COLOR = C, you simply POKE into location 48 (decimal) any integer less than 256 (decimal), you may get a surprise. Depending on what integer you POKE, the screen may "clear" to a pattern of horizontal stripes! I'll bet that some clever reader out there will find some interesting and unexpected application for it. ©



Fun With Apple and Pascal

Gene A. Mauney
Greensboro, NC

While using Kenneth Bowles' excellent textbook, **Problem Solving Using PASCAL,** to self teach Pascal, it occurred to me to write this game program and make learning Pascal even more exciting. Since completing this writing I have discovered that Bowles' 1980 book, **Beginner's Guide for the USDA Pascal System,** would have helped and I am sure will be helpful with my next Pascal ventures.

I tried to use as many of the Apple-Pascal graphics functions as feasible in order to gain experience with these and of course depended on the **Apple Pascal Reference Manual** for this. From TURTLEGRAPHICS used are: MOVE, MOVETO, TURN, TURNTO, GRAFMODE, TEXTMODE, VIEWPORT, FILLSCREEN, TURTLEX, TURTLEY, WCHAR, and CHARTYPE. And from APPLESTUFF the RANDOM, PADDLE, BUTTON, and NOTE functions.

My plan was to use as much as would fit in with my study of the beginning lessons in Bowles' textbook along with developing a program for a game suggested to me by Peter Hildebrandt, to whom goes my appreciation. Also thanks to Bill Stanley for his helpfulness. In these beginnings I found that it would have been very helpful to have had some real Apple-Pascal programs for examples. So my hope is that this real program will be helpful for those readers who are beginners as I. No claims are made as to the most efficient methods for programming and I am sure that others will be able to find improvements. I will be happy to hear from anyone who has comments and suggestions. I hope programmers and players will enjoy it.

The Program

BEGIN(*MAIN*) first draws the Pentagon War Games frame using the TURTLE, then proceeds to the MOVEPENT PROCEDURE. The program switches back and forth between MOVEPENT and IFPADDLE. MOVEPENT creates the pentagons beginning at a random start point (AX,AY) with SIDE = 1, and moving from there in random ways increasing by SIDE + 3 (*NOTE6*) each time for nine times. Here is a place to change the difficulty level for the player if you wish. NINE counts the times through to know when nine pentagons have been formed and also to know the score for adding up totals. IFPADDLE accesses the paddle position and moves the gun. At two places (*NOTE 4*) the TURTLEGRAPHICS procedure, CHARTYPE(6), is used to turn off the previous position of the gun and bullets by XORing the image. CHR(11) is the up arrow used for gun and bullet. If BUTTON(0) is pushed so is TRUE, the IFBUTTON PROCEDURE produces the four bullets with sound each. Hit or miss is determined (*NOTE 5*) by using the last value of X, the lower left corner of the pentagon and the last value of SIDE along with the paddle position. If a hit is made, NINE, SCORE and TSCORE are added up, destruction of the pentagon is shown along with sound (*NOTE 3*), and the message shown. The TURTLEX and Y functions are used (*NOTE 2*) to determine the X,Y value of the pentagon corner for the destruction picture and 20 lines are used here. The procedure FILLSCREEN is used (*NOTE 1*) to erase the last pentagon just before the destruction image. Finally, after five pentagon attacks, the end message is shown along with the total score.

```
PROGRAM PENTAWAR;
USES TURTLEGRAPHICS, APPLESTUFF;
VAR   SCORE,TSCORE,X,Y,
      NINE,SIDE,PENTA: INTEGER;

PROCEDURE THEEND;
BEGIN
  TEXTMODE;WRITELN;WRITELN;
  WRITELN('* * *   PENTAGON WARS   * * *  ');
  WRITELN;WRITELN;
  WRITELN('    YOUR TOTAL SCORE IS ',TSCORE);
  WRITELN;WRITELN;WRITELN;
  WRITELN('DIRECTIONS: ');
  WRITELN('YOU WILL SEE 5 PENTAGON ATTACKS.');
  WRITELN('YOU WILL GET ONLY 5 SHOTS.');
  WRITELN('MAXSCORE IS 45 IF YOU HIT THE');
  WRITELN('SMALLEST PENTAGON OF EACH ATTACK.');
  WRITELN('(9,8,7,..0 AS PENTAGONS ATTACK.)');
  WRITELN('USE APPLE GAME PADDLE 0.  ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('    PRESS RETURN THEN R FOR');
  WRITELN;
  WRITELN('    A NEW GAME.  GOOD LUCK!');
  WRITELN;
END;

PROCEDURE MISS;
VAR  TIME:  INTEGER;
BEGIN
  TEXTMODE;WRITELN;WRITELN;
  WRITELN; WRITELN;WRITELN;
  WRITELN('        YOU MISSED !');
  WRITELN;
  WRITELN('  ONLY ONE SHOT PER ATTACK.');
  WRITELN('   BETTER LUCK NEXT TIME.');
  WRITELN;
  WRITELN('  PRESS BUTTON TO CONTINUE.');
  WRITELN; WRITELN;WRITELN;
  WRITELN; WRITELN;WRITELN;
  FOR TIME:= 1 TO 800 DO
  BEGIN END;        (*WAIT BUTTON RELEASE*)
```

```
    REPEAT NINE:= 0
    UNTIL BUTTON(0);
  FOR TIME:= 1 TO 200 DO
  BEGIN END;                 (*WAIT AGAIN*)
END;

PROCEDURE HIT;
VAR  HITS,LENGTH,ANGLE,
     TX,TY,PITCH,DUR,TIME:   INTEGER;
BEGIN
  SCORE:= NINE + 1;
  TSCORE:= TSCORE + SCORE;
  VIEWPORT(2,277,90,180);
  FILLSCREEN(BLACK);           (*NOTE 1*)
  VIEWPORT(0,279,0,191);
  MOVETO(X,Y); TURNTO(90);
  LENGTH:=21; ANGLE:=120;
  DUR:=1; PITCH:=40;
  FOR HITS:= 1 TO 20 DO        (*NOTE 2*)
    BEGIN
      PENCOLOR(WHITE);
      MOVE(LENGTH);
      TX:=TURTLEX;  TY:=TURTLEY;
      PENCOLOR(BLACK);
      TURN(180); MOVE(LENGTH);
      MOVETO(TX,TY);
      TURN(ANGLE);
      LENGTH:=LENGTH-1; ANGLE:=ANGLE-2;
      NOTE(PITCH,DUR); PITCH:=PITCH-2;
    END;                       (*NOTE 3*)
    TEXTMODE;
    WRITELN;WRITELN;WRITELN;WRITELN;
    WRITELN('          A HIT ! ! !');
    WRITELN;WRITELN;
    WRITELN('          SCORE IS ',SCORE);
    WRITELN;WRITELN;
    WRITELN('   PRESS BUTTON TO CONTINUE.');
    WRITELN;WRITELN;
    WRITELN;WRITELN;WRITELN;WRITELN;
    FOR TIME:= 1 TO 800 DO
    BEGIN END;      (*WAIT BUTTON RELEASE*)
      REPEAT NINE:= 0
      UNTIL BUTTON(0);
    FOR TIME:= 1 TO 200 DO
    BEGIN END;                 (*WAIT AGAIN*)
END;

PROCEDURE IFBUTTON;
VAR  PX,BUL,TWO,PITCH,DUR:   INTEGER;
BEGIN
  PX:= (PADDLE(0)+19);
  MOVETO(PX,20);
  FOR PITCH:= 40 TO 50 DO
  BEGIN
    DUR:=1; NOTE(PITCH,DUR);
  END;
  FOR BUL:= 1 TO 4 DO      (*4 BULLETS*)
    BEGIN
      TURNTO(90);              (*TURN UP*)
      MOVE(20);
      FOR TWO:= 1 TO 2 DO
        BEGIN
          CHARTYPE(6);         (*NOTE 4*)
          WCHAR(CHR(11));
          TURNTO(180);
          MOVE(7);
        END;
    END;
    IF (PX > X) AND (PX < (X+SIDE))
      THEN BEGIN HIT;
      END                      (*NOTE 5*)
      ELSE BEGIN MISS;
      END;
END;
```

```
PROCEDURE IFPADDLE;
VAR  TIME:  INTEGER;
BEGIN
  PENCOLOR(BLACK);
  MOVETO((PADDLE(0)+19),20);
  FOR TIME:= 1 TO 8 DO        (*TIME*)
  BEGIN                       (*ADJUST*)
    CHARTYPE(6);
    WCHAR(CHR(11));           (*NOTE 4*)
    TURNTO(180);
    MOVE(7);
  END;
  IF BUTTON(0) THEN
  BEGIN
    IFBUTTON;
  END;
END;



PROCEDURE MOVEPENT;
VAR  DRAW,EACHONE,
          AX,BX,CX,NX,AY,BY:  INTEGER;
BEGIN
  REPEAT
  PENTA:= PENTA+1; SCORE:= 0;
  NINE:= 9; SIDE:= 1; BY:= 1; NX:= 1;
  AX:= 40+RANDOM MOD(200);
  AY:= 166;
  PENCOLOR(BLACK);
  MOVETO(AX,AY);          (* PENTAGON START*)
  WHILE NINE > 0 DO       (*9 PENTAGONS EACH*)
  BEGIN
    VIEWPORT(1,278,90,180);
    FILLSCREEN(BLACK);       (*CLEAR SCREEN*)
    VIEWPORT(0,279,0,180);
    GRAFMODE;
    BX:= RANDOM MOD(6+NX);
    CX:= RANDOM MOD(6+NX);
    BX:=BX-CX; BY:=BY+8; NX:=NX+4;
    X:=AX-BX; Y:=AY-BY;
    SIDE:= SIDE+3;              (*NOTE 6*)
    NINE:= NINE-1;
    PENCOLOR(BLACK);
    MOVETO(X,Y); TURNTO(0);
    PENCOLOR(WHITE);
    FOR EACHONE:= 1 TO 5 DO
    BEGIN                       (*5 SIDES*)
      MOVE(SIDE);
      TURN(72);           (*PENTAGON ANGLE*)
    END;
    IFPADDLE;
  END;
  UNTIL PENTA = 5;       (*AT END OF GAME*)
  THEEND;
END;



BEGIN                     (*MAIN*)
  INITTURTLE;
  MOVETO(0,0);
  PENCOLOR(WHITE);    (*DRAW THE FRAME*)
  MOVETO(279,0);
  MOVETO(279,191);
  MOVETO(0,191);
  MOVETO(0,0);
  PENCOLOR(BLACK);
  RANDOMIZE;
  PENTA:= 0; TSCORE:= 0;
  MOVEPENT;
  READLN;               (*WAIT FOR <RTN>*)
END.
```

©

Flipping Your Disk

M. G. Sieg

If you own an APPLE DISK II, you can double the storage capacity of a single mini-floppy at virtually no cost. The only things you need are at least two floppies, a hand held hole punch, and a colored pencil that will show on black.

The trick is simple, make your single sided floppy into a dual sided ''flipped'' floppy.

First, let's get acquainted with the anatomy of a floppy disk. Externally there is a black jacket with several holes cut into it. The inside of the jacket is lined with a white fabric which can only be seen by prying the jacket apart a bit at the center hole. Through the holes, the rust colored disk can be seen. The rust color is a coating on a mylar surface enabling the drive to read and record information much the same as the tape for your cassette recorder.

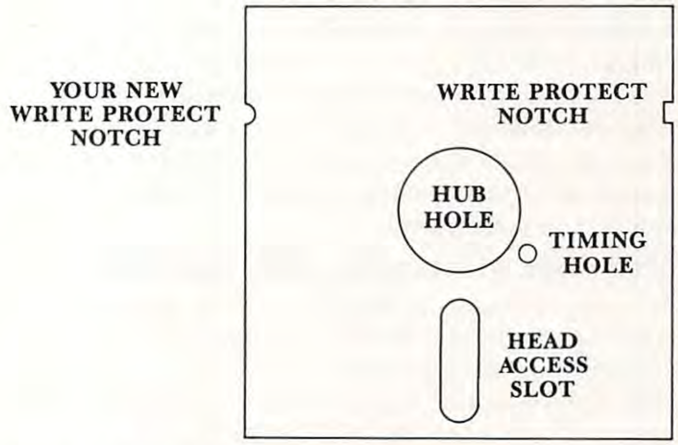
The hole in the center of the jacket is the hub hole. This permits the disk drive motor to engage the disk and spin it. The long wide slot just below the hub hole is called the head access slot. It permits the read/write head and the pressure pad to access the spinning disk. IMPORTANT: Avoid touching the disk surface through this slot. Fingerprints on the recording surface in this area can cause I/O errors. Just to the right of the hub hole is a small hole through which the disk surface can be seen at times or, at other times, a hole completely through the other side of the jacket appears. This is the timing hole. Finally, in the upper right corner (if you consider the head access slot the bottom) of the jacket, there is a rectangular slot. This is the write protect notch. When the floppy is inserted into the drive, a mechanical switch can slip into this notch signalling the drive that it is OK to write on this disk. If the notch is not present or is covered with a piece of tape the disk is ''write protected'' thereby preventing the APPLE from writing anything on this disk — even the initialization information.

By duplicating this write protect notch at the same position on the left side of the jacket, the disk can be turned over and the DISK II may write on the ''flip'' side. The APPLE DISK is different than most other drives because it ignores the timing hole, using the motor and 'soft' timing techniques instead.

If you follow these instructions carefully, a good 90% of major brand mini-floppies can be turned into ''flipped'' floppies. Place two disks in front of you face up on a very clean surface. Once again you are cautioned not to touch the recording surface through the head access slot. Take one of the disks and place it flipped over on top of the other, such that the head access slots are at the bottom. Align them both exactly and, with a light colored pencil, make a mark on the bottom disk along the inside edge of the flipped floppy's write protect notch. With a standard hole puncher, punch a half hole (i.e. no further into the jacket than your pencil mark) completely through both sides of the jacket at your pencil mark. This half hole is now the write protect notch for the flip side. The fact that this hole is round is of no consequence, since the only thing of importance is that the mechanical switch inside your drive can drop into a notch of some type.

Test your ''flipped'' floppy by inserting it into the drive (flipped side up naturally) and doing the normal INIT procedure. If you get several groans from your drive followed by an I/O ERROR, you may not have your notch deep enough or you may have run into one of the 10% or so disks that have flaws in the flipped surface. If you suspect your notch may not be deep enough, very carefully cut away a little more of the jacket. You must be careful not to cut into the disk surface, for that may ruin the disk completely. Assuming you have reasonable quality disks, the flipped surface having a flaw will be a rare problem but has no solution. If you should be unlucky enough to have this occur first time out, don't be discouraged; try another disk.

These flipped floppies may now be used exactly as you use all the normal disks in your collection.



This is what your flipped floppy should look like *after* following the procedures.

Editor's Note: While we've printed this article as a reader service, you should be ***well aware*** *of the risks involved. Disks made for single sided use may contain flaws on the reverse side. We can't vouch for the author's 10% figure. In essence, try this at your risk! RCL*

©

THE ATARI® GAZETTE

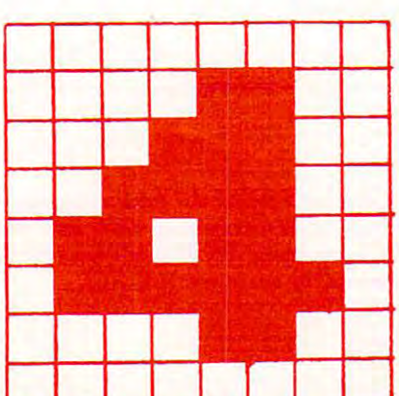
Designing Your Own Atari Character Sets

Craig Patchett
Greenwich, CT

If you want to draw boxes, or design a card game, then Atari's graphics characters are terrific. But what if you're writing an outer-space game or a music program? Wouldn't you prefer a rocket ship or a musical note to a vertical line? This article will explain not only how to change Atari's graphics characters to whatever you desire, but also how to change any Atari character at all, from letters to numbers to punctuation.

What does a character really look like?

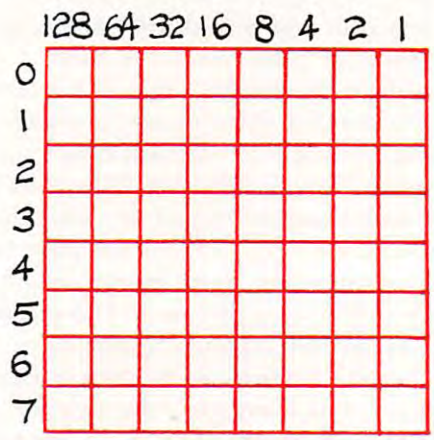
An Atari character, as you may already know, is made up of a bunch of small dots grouped close together. A total of 64 dots, arranged in an eight-by-eight square, can be used to make one character. An Atari "4", for example, really looks like this:



Here, the squares colored in represent the dots that are used. Notice that the outside squares are not used. If they were, then the characters would touch each other when printed side by side, and would be difficult to read. Graphics characters can be made to touch, however, since side by side they could be made to look like one large, continuous character.

How does the Atari know which dots to use for each character?

Somewhere in memory the Atari has a list of which dots are used for each character. Before we find out where this list is, let's see how the Atari represents each character in the list.



The Atari remembers each character as eight numbers, each representing a row of eight dots. These rows I have numbered above from 0 to 7. Row 0 is always the first number, row 7 the last. The Atari changes each row of dots into a number from 0 to 255 in the following way. Each dot in the row is assigned a multiple of two (fron 1 to 128) as its value, as shown above. To get the number for a given row, just add up the values of the dots used in that row. For example, let's look at the "4". The first number will be 12, since dots 4 and 8 are being used in row 1 (4 + 8 = 12). The third number will be 28, since dots 4, 8, and 16 are being used in row 2 (4 + 8 + 16 = 28), and so on down to row 7, which will be 0, since no dots are being used. Before going on, make sure you understand how to get the following eight numbers as representing the number "4": 0, 12, 28, 60, 108, 126, 12, 0.

Where does the Atari store the list?

Since there are a total of 128 Atari characters, not counting reverse characters (see Appendix C: ATASCII Character Set, in the BASIC Reference

Manual), the list will contain 1024 numbers (8 numbers per character X 128 characters = 1024 numbers), Look at Appendix D: Atari 400/800 Memory Map in the BASIC Reference Manual. This simply describes what some of the different memory locations are used for. We're interested in the first locations, containing the "Operating System ROM." The Operating System is just a program that tells the Atari how to do everything it can do in the "Memo Pad" mode, simple things such as putting a character on the screen when a key is pressed, etc. ROM means that the program will always be in the computer's memory, even when the computer is turned off, and can never be changed by the programmer (that's you). Unfortunately, the first 1024 locations in the Operating System ROM (locations 57344 to 58367) contain the list of numbers we are interested in. In order to change the characters we are going to have to change the list, which ROM won't let us do. There's an easy way out, however, and that's to move the list to a place where we can change it.

Where do we move the list to?

We need a place where the list will be safe from us accidentally changing it, but where we will be able to change it when we want to. Looking at Appendix D again, about halfway down the page is a box labeled "RAMTOP". RAMTOP points to the last location in user memory, the memory we have available for our use. What if we were to change RAMTOP so that it pointed 1024 locations before the end of user memory? Then the Atari would think that user memory ended at the new RAMTOP and would not try to put anything in memory after that location. We would still be able to use those locations ourselves though. Let's flip over to Appendix I: Memory Locations. If we look up decimal location 106, we see that it contains the value of RAMTOP. So if we change location 106, we can trick the Atari into staying away from our list. Before we do that, however, let me point out that adding one to the value in 106 actually adds 256 to RAMTOP. This is because of something called "paging", which is too complicated to explain here, and not really important for what we're doing anyway. Just be aware that to move RAMTOP back 1024 locations, we need to subtract four (4x256 = 1024) from location 106. To give us some extra space in case the Atari accidentally goes a little past RAMTOP, we'll subtract five instead. We do this using POKE and PEEK as such (finally some programming!):

```
10 POKE 106, PEEK (106)-5:GRAPHICS 0
```

The reason we use a GRAPHICS Ø right after changing RAMTOP is because the Atari normally stores screen data in the locations we'll be using for the list (see **Designing Your Own Atari Graphics Modes** in the Sept/Oct issue of COMPUTE). If we don't use a GRAPHICS command to move that list to a new location, the screen will do strange things when we move the character list into place, which we are now ready to do (yay!).

How do we move the list?

Moving the list is extremely simple; we just use a FOR/NEXT loop and POKE the values from ROM into their new locations. We first need to figure out the value of the location of the first number in the new list as such:

```
20 STARTLIST = (PEEK(106) + 1)*256
```

Remember, we subtracted an extra one from location 106 to be safe, so we have to add it back on to determine the start of the list. Also don't forget that we have to multiply the value in 106 by 256 because of paging. Now let's move (!):

```
30 ? "HOLD ON...":FOR MOVEME = 0 TO 1023:POKE
STARTLIST + MOVEME,PEEK(57344 + MOVEME):
NEXT MOVEME
```

All that's left now is to tell the Atari where the new list is. We do this by changing the value in location 756, which points to the starting location of the character set to be used (look at Appendix I). If you look at location 756 at this stage (use PRINT PEEK(756)), you'll see that it contains the value 224. Again, because of paging, this really means 224 x 256, or 57344 (surprise!), the starting location of the character set in ROM. So we go:

```
40 POKE 756,STARTLIST/256
```

A few words of warning about location 756. Everytime you use the GRAPHICS command, the Atari sets the value in location 756 back to 224. That means that after each GRAPHICS command, you'll have to execute the equivalent of line 4Ø. No big deal, but if you forget...

Let's change some characters!

Before we actually make any changes, let's look at the order the characters are stored in the list. For this we'll need Appendix C again (and you thought you'd never use the Appendices!). Unfortunately, Atari chose not to store the characters in memory exactly in the ATASCII order. Almost, but not exactly:

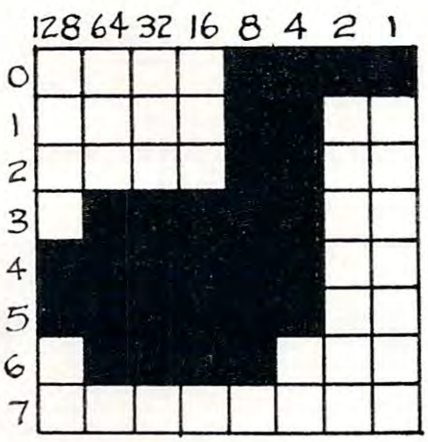
TYPE	ATASCII ORDER	MEMORY ORDER
uppercase, numbers, punctuation	32-95	0-63
graphics characters	0-31	64-95
lowercase, some graphics	96-127	96-127

As you can see, all that Atari did was to move the graphics characters between the uppercase and lowercase (they did this in order to be able to choose between uppercase and lowercase/graphics in modes one and two). In the meantime, they made our job harder for us. In order to determine where a character is stored in memory, we have to perform a *little* mathematical wizardry on its ATASCII value. In the following "formulas," keep in mind that each character is represented by eight numbers, which is why we multiply by eight:

ATASCII VALUE (AV)	MEMORY LOCATION (of first number)
32-95	(AV-32)*8 + STARTLIST
0-31	(AV + 64)*8 + STARTLIST
96-127	AV*8 + STARTLIST

Of course, to get the location of the original character (in ROM), we would add 57344 instead of STARTLIST.

With these mathematical manipulations in mind, let's try one of the original examples that I mentioned. We'll change one of the graphics characters, let's say ◄CTRL►T, to a musical note. First, let's design our note:



This may not look exactly like a note as is, but because of the size of the dots, it will look fine when printed on the screen, as we shall soon see. I'll leave it up to you to check for yourself that the note translates into the following eight numbers: 15, 12, 12, 124, 252, 252, 120, 0. We now want to replace the eight numbers already in memory for ◄CTRL►T with these eight. ◄CTRL►T has an ATASCII value of 20 (see Appendix C), which fits in the 0-31 category in the formula chart above. The first thing to do, therefore, is to add 64 (20 + 64 = 84) and multiply by eight (8x84 = 672) to give us a value of 672. So to change the ◄CTRL►T character we would have to change the eight numbers in memory beginning with location 672 + – STARTLIST. We make this change using a FOR/NEXT loop and DATA statements as such:

```
50 FOR MOVEME = 0 TO 7:READ VALUE:POKE 672
 + STARTLIST + MOVEME,VALUE:NEXT MOVEME
60 DATA 15, 12, 12, 124, 252, 252, 120, 0
```

Now, after this has been RUN, whenever we use a ◄CTRL►T, we will have a musical note. Try it!

As an informal kind of self-test, make sure you understand the following two lines. Try and work out which character they will change, and what the new character will look like, before you actually RUN them (with the rest of the program of course):

```
70 FOR MOVEME = 0 TO 7:READ VALUE:POKE 776
 + STARTLIST + MOVEME,VALUE:NEXT MOVEME
80 DATA 0, 0, 60, 102, 102, 102, 63, 0
```

As you can see, lines 50 and 70 are very much alike except for the initial value added to STARTLIST. This should light up a sign in your brain saying "SUBROUTINE!" If you have more than one or two characters to be redesigned, you should use a subroutine to save memory.

A few details and programming hints.

- In graphics modes one and two, to use lower case and graphics characters with your new character set, POKE 756 with STARLIST /256 +2. To go back to uppercase, etc., POKE 756 with STARTLIST/256.
- If you press the RESET button, the Atari will change the value of location 106 and put the display list back in place of your character set. Under such circumstances it is necessary to run the program over again in order to get your character set back again.
- If a character is too complicated to put in an eight by eight box, then use more than one box (and therefore more than one character), and combine them in a string. For example, using the Atari's regular graphics characters:

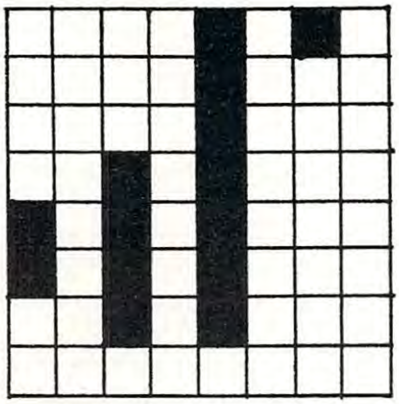
DIM BOX$(7):BOX$ = " (see below) ":PRINT BOX$
Type BOX$ as ◄CTRL►Q, ◄CTRL►E, ◄ESC► ◄CTRL► =, ◄ESC►◄CTRL► +, ◄CTRL►Z, ◄CTRL►C.

Bonus: Four Colors In Graphics Mode O!

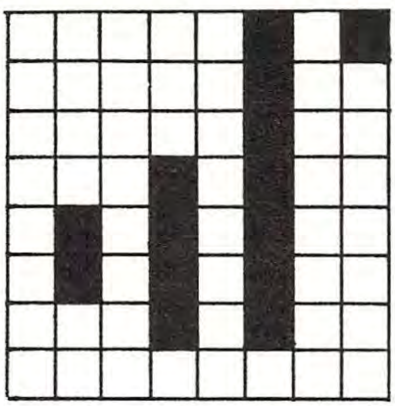
It is possible to define a character to be one of three different colors (4 = 3 + background). The only drawback is that once you have defined the letter "A" to be orange, for example, all "A"'s will be orange, not just the ones you would like to be.

How do we define the color of a character? It's really quite simple. Just as in graphics mode eight, a dot in an even numbered column will be a different color than a dot in an odd numbered column. Two dots side-by-side will produce yet another color. This is why an Atari "4" (and all other Atari characters) and my musical note have vertical lines that are two dots wide, compared to the horizontal lines that are only one dot wide (or thick if you prefer). If the vertical lines were only one dot wide, they would be a different color than the horizontal ones, unless the horizontal lines alternated one dot on and one dot off. Confused? Don't worry, just substitute the following variations of the musical note for the data in the sample program and see what they look like:

60 DATA 10,8,8,40,168,168,32,0



80 DATA 5,4,4,20,84,84,16,0



Such characters will, of course, look unusual in graphics modes one and two, just as they look unusual in the above diagrams.

You can't do a lot of experimenting with this "phenomena" to get such effects as multicolored characters. Changing the background color will change the colors of the columns, and thus the colors of the characters. Finally, if you only need one "A", or whatever, to be a different color, define it as a graphics character.

Bonus: Upper and Lowercase in Graphics Modes 1 and 2.

By now, after hopefully running things over in your mind, you might even suspect already how to mix upper and lowercase in modes one and two. If not, it is a painfully simple trick. Since modes one and two allow use of lowercase and graphics characters together, just redefine the graphics characters to be uppercase letters! You can do this by moving the uppercase character descriptions from the ROM list to your own list like so:

```
35 FOR MOVEME = 256 TO 472:POKE STARTLIST +
MOVEME + 256,PEEK(57344 + MOVEME):NEXT
MOVEME
```

Typing a ◄CTRL►A will now give us an uppercase "A" and so on. Of course, this is not the best way to do it, since we no longer have any graphics characters. If we know that we will only need certain uppercase letters in our program, then it would be better to move just those letters, one by one, using the tables given earlier in the article. In any case, we are now able to mix almost any combination of characters we wish in graphics modes one and two.

And as the sun sets slowly in the west...

I realize that I have attempted to cover quite a bit of information over the course of this article, and most likely was not able to explain everything to everyone's satisfaction. If you have any *pressing* questions with regard to what I have covered here, please feel free to send them to me at the above address, along with any constructive criticism you might also have, and I'll do my best to answer them. Good luck, and always remember; the Atari is your slave and you its master. ©

Atari BASIC
A Line Renumbering Utility

D. M. Gropper, Newburgh, Indiana

Most programmers developing a program need to insert lines of code into what they have already written. The current version of Atari BASIC does not have a "RENUM" or "RESequence" command available to the user. The following short program was written to give the capability of renumbering BASIC language programs.

Listing 1.

```
9000 PRINT "RENUM UTILITY ACTIVE"
9010 ADDR = PEEK(136) + PEEK(137) * 256
9020 PRINT "INPUT STARTING NUMBER AND"
9030 PRINT "INCREMENT (FORMAT X,Y)"
9040 INPUT START,INCR
9050 LNUM = PEEK(ADDR) + PEEK(ADDR + 1) * 256
9060 NADDR = ADDR + PEEK(ADDR + 2) *256
9070 IF (LNUM = 32768) OR (LNUM = 9000)
  THEN GOTO 9110
9080 LOWNUM = INT(START / 256) :
  HINUM = INT(START - (LOWNUM *256))
9090 POKE ADDR,HINUM : POKE ADDR +
  1,LOWNUM
9100 START = START + INCR: ADDR = NADR :
  GOTO 9050
9110 PRINT "RENUMBERING ENDED AT
  ";START - INCR
9120 END
```

LINE 9010; The address of any BASIC programs' first line is given by the contents of locations 136 and 137.

LINE 9050: The first two bytes, starting at the address from line 9010, contain the actual line number.

LINE 9060: The third byte contains the length of the line in bytes, so adding this to the address given by locations 136, 137 will give the address of the next line.

LINE 9070: Here we are testing for the end of the program, L = 32768, or the start of the utility, L = 9000

LINE 9080: The new line number is broken into two bytes,

LINE 9090: And 'POKEd' back into the line number bytes.

LINE 9100: Update the line number and address and repeat.

LINE 9110: All done...Let's get out of here and tell the programmer what the last line change was.

Listing 2.

```
9010 D = 256 : S = 100 : I = 10 : X = PEEK(136) +
  PEEK(137) * D
9020 L = PEEK(X) + PEEK(X + 1) * D : N = X +
  PEEK(X + 2) : IF (L = 32768) OR (L = 9010) THEN
  END
9030 HN = INT(S/D) : LN = INT(S - (HN * D)) :
  POKE X,LN : POKE X + 1,HN : S = S + I :
  X = N : GOTO 9020
```

This program occupies 534 bytes. Listing 2 is the same thing only compacted down to 289 bytes for those of us who get tight on memory space. A point to note is that listing 2 assumes starting the line renumbering with a line number of 100 and incrementing by 10.

A not so obvious point is that only the line numbers are changed and **not** the sequence of execution. For example:

Taking

```
100 X = 100: ? X
110 Y = 200: ? Y
120 Z = 300: ? Z
```

If we now use listing 1 to resequence starting with 120 and using an increment of -10 (in other words decrementing) then the end result is:

```
120 X = 100: ? X
110 Y = 200: ? Y
100 Z = 300: ? Z
```

The output on the screen on running the renumbered program would be:

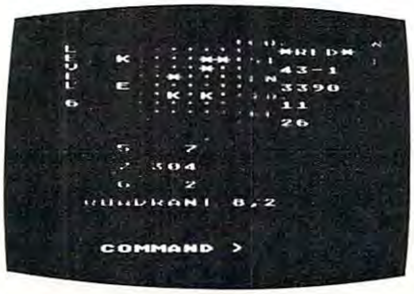
```
100
200
300
```

, because the locations 136 and 137 still point to the first line which is now line *number* 120. The third byte at this address is still the **length** of the line in bytes so the next line to be executed is the *now-numbered* 110, etc.

For those of you who like to experiment — take the above example and renumber starting at number 1 and use an increment of 2 and then "LIST" the result. If you bear in mind the editing capability of the Atari then the reason for the "LISTed" result becomes obvious.

One last point — if you do use this utility then please remember that you will have to manually change any "GOSUB" or "GOTO" line references. ©

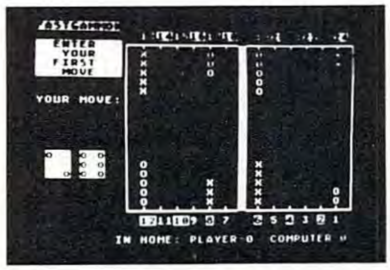
SOFTWARE FOR THE ATARI 800* AND THE ATARI 400*



TARI TREK™
By Fabio Ehrengruber

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

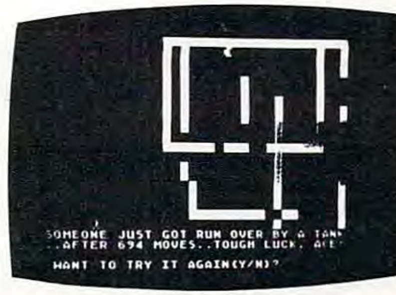
Cassette - $11.95 Diskette - $14.95



FASTGAMMON™
By Bob Christiansen

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

On cassette only - $19.95



TANK TRAP
By Don Ursem

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

Cassette - $11.95 Diskette - $14.95

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

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

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

On diskette only - $79.95

★ ★ ★ ★ ★ ★ ★ ★ ★

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

On cassette only - $24.95

★ ★ ★ ★ ★ ★ ★ ★ ★

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

Cassette - $11.95 Diskette - $14.95

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

WHERE TO GET IT: Call us at (213) 344-6599 for the name of the Quality Software dealer nearest you. If necessary you may order directly from us. Mastercard and Visa cardholders may place orders by telephone. Or mail your check or bankcard number to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335. California residents add 6% sales tax. SHIPPING CHARGES: Within North America orders must include $1.50 for first class shipping and handling. Outside North America the charge for airmail shipping and handling is $5.00. Pay in U.S. currency.

*Indicates trademarks of Atari, Inc.

ATARI Memory Dump and Dissassembler

Robert W. Baker
Atco, New Jersey

Here's a handy little utility program for the Atari 400/800 systems. It lets you examine any area of memory, either RAM or ROM, in one of two formats. You can select whether you want a straight memory dump or a dissassembly listing. In both formats, the memory locations are given as both decimal and hexadecimal values. The data can be displayed on the television/monitor screen or printed on a printer if available.

When first run, the program takes a minute or two to initialize but from then on it is relatively fast. The starting address for the dump/dissassembly can be entered as either a decimal or hexadecimal number. When entering it as a hexadecimal number, precede the number by a dollar sign ($). You're then asked if a dissassembly is desired. Answering N for no will cause the standard memory dump to be displayed. Answering Y for yes will generate the dissassembly listing.

Before the dump/dissassembly is displayed you are given the option to have the output printed if desired. Answering N for no causes the output to be displayed as normal, using the entire display (24 lines). At the end of each screen you are given the option to continue (C), restart (R), or stop (S). Continue will display the next screen in sequential order. Restarting will return to select the starting address and allow specifying dump/dissassembly and printer options. When printing the data output, the printer will print continuously. Just press any key on the Atari keyboard to halt the printer. When the printer stops you will see the prompt for continue, restart, or stop as mentioned above.

Memory Dump

The memory dump simply displays the contents of eight bytes of memory on each line displayed or printed. The values are given in hexadecimal to conserve display space and to correspond with the dissassembly listings. This feature is very useful for examining pointers or various values stored in memory, that do not happen to be executable machine code instructions. You might want to play around with looking at how BASIC variables or even BASIC lines themselves are stored in memory on the Atari.

For those with 80 column printers (Atari 825, etc.) you can change the FOR-NEXT loop count in line 600 to get 16 bytes per line to conserve paper. Just change line 600 to:

```
FOR X = 1 TO 16:V = PEEK(A)
```

You might even want to change the heading line in line number 302 to print the numbers 0 to 9 plus A to F.

Dissassembly Listing

This feature is much more powerful and can provide a wealth of information. When a dissassembly is requested, the program displays one 6502 machine instruction per line. It indicates the hexadecimal value of one to three bytes of memory that make up the instruction. It also displays the instruction and operand in the standard assembly code forms.

Any unrecognized values are indicated by a "*?*" instead of an instruction mnemonic. You may have to try different starting locations to get the dissassembly to function properly. If you specify an address that happens to be the middle of an instruction, it may dissassemble as a different instruction and/or cause following instructions to be displayed incorrectly. This is always a problem with a dissassembly program of this kind. It is extremely difficult to "sync-up" with the machine instructions whenever there are data bytes between various routines, etc. If you should see a high number of *?*'s displayed, try another starting address, possibly one to two higher or lower than the original address. This should correct the situation.

The dissassembly gives each instruction using standard MOS Technology mnemonics and addressing nomenclatures. Operand values and addresses are shown in hexadecimal and are prefixed by a dollar sign ($) as a reminder. All immediate values are preceded by parenthesis and indexed values are suffixed by a ",X" or ",Y" as appropriate. Zero page addressing is implied by the length of the operand being only a single byte. All branch instructions show the actual computed target address in the dissassembly for added convenience. If required, the relative offset is shown in the object code.

The dissassembly function is extremely useful for examining machine language routines such as those used within the BASIC cartridge itself, or in the operating system ROMs. I'll let you know if I come across anything interesting hidden in the Atari system. Before anyone asks — if you'd like a copy on cassette tape instead of doing all the typing, send $2 to cover costs.

Just a quick note concerning the program listings. The heading lines printed by the BASIC statements in lines 302 and 305 were actually in reverse image to enhance the display. Unfortunately this does not show up in the program listings. I have

tried to use CHR$(xx) functions in the print statements for clearing the display, etc. to make things easier to read. The Atari printers do not print the graphics and/or control characters that can be included in PRINT statements. Actually they can cause problems if a program is LISTed with these control characters imbedded in PRINT statements. The control characters will be decoded and acted upon by the printer.

```
10 REM ****************************
20 REM
25 REM MEMORY DUMP/DISASSEMBLER
30 REM
35 REM BY: ROBERT W. BAKER
40 REM 15 WINDSOR DR, ATCO NJ 08004
50 REM
60 REM ***** V1.0 ******* 1/4/81 ****
65 REM
70 GRAPHICS 0:POKE 752,1
80 PRINT CHR$(125);"          M E M O R Y
  D U M P   ":? :?
90 PRINT "INITIALIZING ....."
100 DIM H$(16),A$(6),S$(6),M$(1536)
110 H$="0123456789ABCDEF"
120 S$="      "
150 OPEN #1,4,0,"K"
160 FOR X=1 TO 1531 STEP 6
170 READ A$
175 IF A$(2,2)="*" THEN A$(2,4)="*?*"
180 N=LEN(A$):IF N<6 THEN A$(N+1)=S$
185 M$(X,X+5)=A$:NEXT X
200 PRINT CHR$(125);"          M E M O R Y
  D U M P   ":? :?
201 PRINT "ENTER DECIMAL STARTING ADDRES
S":PRINT
202 PRINT "OR HEX ADDRESS PRECEDED BY '$
'":PRINT
203 POKE 752,0
204 INPUT A$:IF A$="" THEN 800
205 IF A$(1,1)="$" THEN 209
206 FOR X=1 TO LEN(A$)
207 IF A$(X,X)<"0" OR A$(X,X)>"9" THEN 2
00
208 NEXT X:A=INT(VAL(A$)/8)*8:GOTO 240
209 A=0:IF LEN(A$)<2 THEN 200
210 FOR X=2 TO LEN(A$)
211 IF A$(X,X)<"0" THEN 200
212 IF A$(X,X)<="9" THEN A=A*16+VAL(A$(X
,X)):GOTO 220
215 IF A$(X,X)<"A" OR A$(X,X)>"F" THEN 2
00
218 A=A*16+ASC(A$(X,X))-55
220 NEXT X
240 PRINT :PRINT "WANT DISSASSEMBLY (Y/N
) ?";
242 GET #1,X:D=0:IF X=78 THEN 245
244 D=1:IF X<>89 THEN 240
245 PRINT CHR$(X)
250 PRINT :PRINT "WANT PRINTED COPY (Y/N
) ?";
252 CLOSE #2
255 P=0:GET #1,X
260 IF X=78 THEN OPEN #2,8,0,"E":GOTO 29
0
270 IF X<>89 THEN 255
280 P=1:OPEN #2,8,0,"P"
290 IF P=0 THEN PRINT CHR$(125);:GOTO 30
0
295 PRINT CHR$(125);"DEPRESS ANY KEY TO
HALT PRINTER":PRINT #2
300 PRINT #2;"LOC-DEC/HEX  ";
302 IF D=0 THEN PRINT #2;"0  1  2  3  4
 5  6  7  ":GOTO 310
305 PRINT #2;" OBJECT   DISSASSEMBLY  "
310 PRINT #2
320 POKE 764,255
330 IF P=0 THEN FOR N=1 TO 20
340 IF A>65535 THEN A=A-65536
350 A$=STR$(A):L=LEN(A$)
360 PRINT #2;S$(1,6-L);A$;" ";
370 Y=A:GOSUB 950
380 PRINT #2;": ";
400 IF D=0 THEN 600
410 V=PEEK(A)
411 GOSUB 1000:PRINT #2;" ";
415 A=A+1:X=(6*V)+1:A$=M$(X,X+5)
420 IF A$(1,1)="0" THEN PRINT #2;"
";A$(2,4):GOTO 630
430 V=PEEK(A):GOSUB 1000
432 PRINT #2;" ";:A=A+1
435 IF A$(1,1)="2" THEN 500
440 PRINT #2;"    ";A$(2,4);" ";
445 IF A$(5,5)<>"R" THEN 470
450 IF V>127 THEN V=V-256
460 Y=A+V:GOSUB 900:GOTO 590
470 IF A$(5,5)="#" THEN PRINT #2;"#$";:G
OSUB 1000:GOTO 590
475 IF A$(6,6)=")" THEN PRINT #2;"(";
480 PRINT #2;"$";:GOSUB 1000
482 IF A$(5,5)=" " THEN 590
485 IF A$(5,6)="Y)" THEN PRINT #2;"),Y":
GOTO 630
490 PRINT #2;",";A$(5,6):GOTO 630
500 V1=V:V=PEEK(A):GOSUB 1000:A=A+1
510 PRINT #2;"  ";A$(2,4);" ";
515 Y=V1+(256*V)
520 IF A$(5,5)=")" THEN PRINT #2;"(";:GO
SUB 900:PRINT #2;")":GOTO 630
525 GOSUB 900
530 IF A$(5,5)=" " THEN 590
540 PRINT #2;",";A$(5,5):GOTO 630
590 PRINT #2:GOTO 630
600 FOR X=1 TO 8:V=PEEK(A)
610 GOSUB 1000:PRINT #2;" ";
620 A=A+1:NEXT X:PRINT #2
630 IF P=0 THEN NEXT N:GOTO 700
640 IF PEEK(764)=255 THEN 340
```

```
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 200
740 IF X<>83 THEN 710
800 POKE 752,0:CLOSE #1:CLOSE #2:END
900 PRINT #2;"$";
950 V=INT(Y/256):GOSUB 1000
960 V=Y-(V*256)
1000 H=INT(V/16):L=V-(H*16)
1010 PRINT #2;H$(H+1,H+1);H$(L+1,L+1);
1020 RETURN
9000 DATA 0BRK,1ORAX),0*,0*,0*,1ORA,1ASL
,0*
9010 DATA 0PHP,1ORA#,0ASL,0*,0*,2ORA,2AS
L,0*
9020 DATA 1BPLR,1ORAY),0*,0*,0*,1ORAX,1A
SLX,0*
9030 DATA 0CLC,2ORAY,0*,0*,0*,2ORAX,2ASL
X,0*
9040 DATA 2JSR,1ANDX),0*,0*,1BIT,1AND,1R
OL,0*
9050 DATA 0PLP,1AND#,0ROL,0*,2BIT,2AND,2
ROL,0*
9060 DATA 1BMIR,1ANDY),0*,0*,0*,1ANDX,1R
OLX,0*
9070 DATA 0SEC,2ANDY,0*,0*,0*,2ANDX,2ROL
X,0*
9080 DATA 0RTI,1EORX),0*,0*,0*,1EOR,1LSR
,0*
9090 DATA 0PHA,1EOR#,0LSR,0*,2JMP,2EOR,2
LSR,0*
9100 DATA 1BVCR,1EORY),0*,0*,0*,1EORX,1L
SRX,0*
9110 DATA 0CLI,2EORY,0*,0*,0*,2EORX,2LSR
X,0*
9120 DATA 0RTS,1ADCX),0*,0*,0*,1ADC,1ROR
,0*
9130 DATA 0PLA,1ADC#,0ROR,0*,2JMP),2ADC,
2ROR,0*
9140 DATA 1BVSR,1ADCY),0*,0*,0*,1ADCX,1R
ORX,0*
9150 DATA 0SEI,2ADCY,0*,0*,0*,2ADCX,2ROR
X,0*
9160 DATA 0*,1STAX),0*,0*,1STY,1STA,1STX
,0*
9170 DATA 0DEY,0*,0TXA,0*,2STY,2STA,2STX
,0*
9180 DATA 1BCCR,1STAY),0*,0*,1STYX,1STAX
,1STXY,0*
9190 DATA 0TYA,2STAY,0TXS,0*,0*,2STAX,0*
,0*
9200 DATA 1LDY#,1LDAX),1LDX#,0*,1LDY,1LD
A,1LDX,0*
9210 DATA 0TAY,1LDA#,0TAX,0*,2LDY,2LDA,2
LDX,0*
9220 DATA 1BCSR,1LDAY),0*,0*,1LDYX,1LDAX
,1LDXY,0*
9230 DATA 0CLV,2LDAY,0TSX,0*,2LDYX,2LDAX
,2LDXY,0*
9240 DATA 1CPY#,1CMPX),0*,0*,1CPY,1CMP,1
DEC,0*
9250 DATA 0INY,1CMP#,0DEX,0*,2CPY,2CMP,2
DEC,0*
9260 DATA 1BNER,1CMPY),0*,0*,0*,1CMPX,1D
ECX,0*
9270 DATA 0CLD,2CMPY,0*,0*,0*,2CMPX,2DEC
X,0*
9280 DATA 1CPX#,1SBCX),0*,0*,1CPX,1SBC,1
INC,0*
9290 DATA 0INX,1SBC#,0NOP,0*,2CPX,2SBC,2
INC,0*
9300 DATA 1BEQR,1SBCY),0*,0*,0*,1SBCX,1I
NCX,0*
9310 DATA 0SED,2SBCY,0*,0*,0*,2SBCX,2INC
X,0*
```

```
LOC-DEC/HEX  0  1  2  3  4  5  6  7

 40992 A020: 95 00 E8 94 00 E8 E0 92
 41000 A028: 90 F6 A2 86 A0 01 20 7F
 41008 A030: A8 A2 8C A0 03 20 7F A8
 41016 A038: A9 00 A8 91 84 91 8A C8
 41024 A040: A9 80 91 8A C8 A9 03 91
 41032 A048: 8A A9 0A 85 C9 20 F8 B8
 41040 A050: 20 41 BD 20 72 BD A5 92
 41048 A058: F0 03 20 99 BD 20 57 BD
 41056 A060: A5 CA D0 9C A2 FF 9A 20
 41064 A068: 51 DA A9 5D 85 C2 20 92
 41072 A070: BA 20 F4 A9 D0 EA A9 00
 41080 A078: 85 F2 85 9F 85 94 85 A6
 41088 A080: 85 B3 85 B0 85 B1 A5 84
 41096 A088: 85 AD A5 85 85 AE 20 A1
 41104 A090: DB 20 9F A1 20 C8 A2 A5
   ***** SAMPLE MEMORY DUMP *****
```

```
LOC-DEC/HEX   OBJECT    DISSASSEMBLY

 40992 A020: 95 00      STA $00,X
 40994 A022: E8         INX
 40995 A023: 94 00      STY $00,X
 40997 A025: E8         INX
 40998 A026: E0 92      CPX #$92
 41000 A028: 90 F6      BCC $A020
 41002 A02A: A2 86      LDX #$86
 41004 A02C: A0 01      LDY #$01
 41006 A02E: 20 7F A8   JSR $A87F
 41009 A031: A2 8C      LDX #$8C
 41011 A033: A0 03      LDY #$03
 41013 A035: 20 7F A8   JSR $A87F
 41016 A038: A9 00      LDA #$00
 41018 A03A: A8         TAY
 41019 A03B: 91 84      STA ($84),Y
 41021 A03D: 91 8A      STA ($8A),Y
 41023 A03F: C8         INY
 41024 A040: A9 80      LDA #$80
```

```
41026 A042: 91 8A      STA ($8A),Y
41028 A044: C8         INY
41029 A045: A9 03      LDA #$03
41031 A047: 91 8A      STA ($8A),Y
41033 A049: A9 0A      LDA #$0A
41035 A04B: 85 C9      STA $C9
41037 A04D: 20 F8 B8   JSR $B8F8
41040 A050: 20 41 BD   JSR $BD41
41043 A053: 20 72 BD   JSR $BD72
41046 A056: A5 92      LDA $92
41048 A058: F0 03      BEQ $A05D
41050 A05A: 20 99 BD   JSR $BD99
41053 A05D: 20 57 BD   JSR $BD57
41056 A060: A5 CA      LDA $CA
41058 A062: D0 9C      BNE $A000
41060 A064: A2 FF      LDX #$FF
41062 A066: 9A         TXS
***** SAMPLE DISSASSEMBLY *****
```

©

Formatted Output for ATARI Basic

Joseph J. Wrobel

Many folks tell me that they must struggle to produce nicely formatted output when using ATARI Basic due to the lack of the TAB function and the PRINT USING command. Struggle no more. When used together, the two subroutines presented in this article can provide formatted output simply and directly in ATARI Basic. Both numerics and strings are supported. The type, arrangement and format of variables which appear on one output line are controlled on a line-by-line basis by the main program. The number of variables in one output is limited only by the character width of the output device. The output device can be the TV screen or any type of printer, ATARI or otherwise.

The approach to formatted output used here employs two subroutines. The purpose of the first is to construct a line for output in a string variable set aside for this purpose. Each time the subroutine is called, it inserts the data sent to it by the main program at the selected position in the string and in the format requested. When all the data to be printed in the current line has been positioned, the second subroutine is called. This subroutine merely prints the output line string on the appropriate device, then clears the string (fills it with spaces) to prepare it for the next line of data.

A sample program using the routines is given in Listing 1. Line 10 is required to set aside the strings to be used in the subroutines. LINE$ is the string which will ultimately contain the formatted output line. It is dimensioned to a size one less than the character width of the output device. In the example, for the 38 character wide default screen size, it is dimensioned to hold up to 37 characters by setting NC to 37. Line 20 initializes LINE$ to a string of spaces. N$ is a string used to temporarily store each data item. It should be big enough to hold the largest of your data items. To be on the safe side, its length is set equal to that of LINE$.

The actual subroutines of interest reside in lines 1000-1070 and lines 2000-2010 respectively. The latter routine, as noted above, simply prints LINE$ (line 2000) then fills it with spaces (line 2010) in preparation for the next output line. If, instead of the screen, a printer is the output device, then the PRINT of line 2000 may simply be replaced by an LPRINT or a PRINT # command, whichever is appropriate.

The routine starting at line 1000 actually does the formatting and has two different entry points depending on whether the data item is a string or a numeric. The case of a string is the simpler of the two, so let's consider it first. To position a string you must place the string in variable N$, specify the column position (RC) against which it is to be right justified, then GOSUB 1060. Three examples of this calling sequence are given in lines 100-120. At line 1060 the program first calculates LC, the leftmost character position of the data item. Then, if the column boundaries are acceptable, it inserts N$ in LINE$ and RETURNs.

To position a numeric, you must put the data item into variable N, specify ND, the number of digits to the right of the decimal point you wish printed, specify RC as defined above, then GOSUB 1000. See lines 150-170 for examples of this calling sequence. In lines 1000-1010 N is rounded to the appropriate number of decimals. In 1020 the string N$ is defined. If the number is to be printed as an integer (ND = 0), N$ is correct as is and the jump is taken to 1060 to insert N$ into LINE$. For non-integer formats, the decimal point and any trailing zeroes which were dropped in forming the string representation of the number must be restored. This is done in lines 1030-1050. N$ is then ready for insertion into LINE$.

Output from the sample program as printed on an ATARI 820 printer (using LPRINT in 2010) is shown in Figure 1. Note that the numbers are rounded for presentation with the requested precision, that the decimal points of each column neatly line up, that all trailing zeroes are present and that negative numbers are also accommodated. Also note how easy it is to line up the column headings with the data by simply specifying the appropriate value of RC when printing them (see lines 100 & 150, 110 & 160, 120 & 170).

The routines run fairly rapidly, but if you need some extra speed, the loop in line 2010 can be avoided. To do this, dimension a string, let's call it

MT$, the same length as LINE$ and fill it with spaces just once at the start. Then, line 2010 can be replaced by **2010 LINE$ = MT$:RETURN**

If you have a slow printer like a teletype, you may also gain some speed by stripping the trailing spaces from LINE$ before printing it. This can be done by replacing line 2000 with the three lines given below.

```
2000 FOR I = NC TO 1 STEP -1:IF LINE$(I,I)<>" "
  THEN 2004
2002 NEXT I
2004 PRINT LINE$(1,I)
```

```
1 REM ** FORMATTED OUTPUT EXAMPLE **
2 REM JOE WROBEL, ROCHESTER, NY
3 REM SUBROUTINE VARIABLES - I,LC,N,NC,N
D,NZ,RC,LINE$,N$
10 NC=37:DIM LINE$(NC),N$(NC)
20 GOSUB 2010
100 N$="X":RC=7:GOSUB 1060
110 N$="X/32":RC=17:GOSUB 1060
120 N$="SIN(PI*X/8)":RC=33:GOSUB 1060
130 GOSUB 2000
140 FOR X=0 TO 15
150 N=X:ND=0:RC=7:GOSUB 1000
160 N=X/32:ND=3:RC=17:GOSUB 1000
170 N=SIN(4*ATN(1)*X/8):ND=7:RC=32:GOSUB
 1000
180 GOSUB 2000:NEXT X
190 STOP
1000 I=INT(10^ND+0.5)
1010 N=INT(I*N+0.5)/I
1020 N$=STR$(N):IF ND=0 THEN 1060
1030 IF N=INT(N) THEN N$(LEN(N$)+1)="."
1040 NZ=ND+1+LEN(STR$(INT(N)))-LEN(N$)
1050 IF NZ>0 THEN FOR I=1 TO NZ:N$(LEN(N
$)+1)="0":NEXT I
1060 LC=RC+1-LEN(N$):IF LC<=RC AND RC<=N
C AND LC>=1 THEN LINE$(LC,RC)=N$
1070 RETURN
2000 PRINT LINE$
2010 FOR I=1 TO NC:LINE$(I,I)=" ":NEXT I
:RETURN
```

X	X/32	SIN(PI*X/8)
0	0.000	0.0000000
1	0.031	0.3826834
2	0.063	0.7071068
3	0.094	0.9238795
4	0.125	1.0000000
5	0.156	0.9238795
6	0.188	0.7071068
7	0.219	0.3826834
8	0.250	0.0000000
9	0.281	-0.3826834
10	0.313	-0.7071068
11	0.344	-0.9238795
12	0.375	-1.0000000
13	0.406	-0.9238795
14	0.438	-0.7071068
15	0.469	-0.3826834

©

Random Color Switching While Idle

R. A. Howell

Have you ever been involved in a scenario similar to the following? This has happened to me several times. I get the program listing of a new game from a friend or from the pages of a magazine. The game sounds really exciting, so I anxiously begin typing the program into my Atari 800 computer system. Of course, my 10 year old son is busily watching over my shoulder because he is also anxious to try out the new program. When I finally finish, we try it. After a few corrections for typing mistakes, the game is running and we get deeply engrossed in playing it. About the time we have mastered the rules and are into the full action and excitement of the game, ZAP!!, the Atari 800 goes into its random color switching routine on the screen. This usually makes the playing field difficult to see because the random colors selected by the computer are either too dark or are all of similar shades so they blend together. The solution is simple for those of us who have used the Atari; just hit a key, any key on the keyboard and the original program's colors will be restored. Right! Wrong! The problem with this is that if I take either hand from the joystick or fire button to hit a key, my son gains the advantage in the game and vice versa. Have you also found yourself in this dilemma?

The problem occurs because of a hardware feature on the Atari computer. When a key has not been pressed for a little over 9 minutes, the system automatically starts to vary the colors on the screen. It will continue to randomly vary the colors on the screen until a key on the keyboard is pressed. At that point it will return the screen to the correct colors and begin the 9 minute time-out sequence again. The purpose of this feature is to prevent the image on the screen from being permanently burned into the phosphor on the cathode ray tube when the computer is left for a long time without being changed. However, many programs (games in particular) do not require any key-presses. All inputs come from trigger buttons and game paddles or joysticks. When running such a program, it is inconvenient to have this color switching occur every 9 minutes, so let's look at what triggers this feature and how to prevent it from happening.

Type the following one line program into your Atari computer and run it:

10 PRINT PEEK(77): GOTO 10

This program repeatedly displays the contents of RAM (Random Access Memory) location 77 on the screen. As you can see by watching it run, RAM location 77 starts at 0 and increments by 1 every 4 to 4.5 seconds. Now while this program is still running, press any key on the keyboard (except BREAK or the 4 special function keys). As soon as you pressed the key, notice that location 77 returned to 0 and started incrementing all over again. Now let the program run for a while without pressing any keys. After approximately 9 minutes, the count will be close to 128. As soon as RAM location 77 reaches 128, you will see that it gets set to 254 and the screen colors immediately begin to vary randomly. Now with the program still running, press any key again. Normal colors are returned to the screen and location 77 begins all over incrementing from 0.

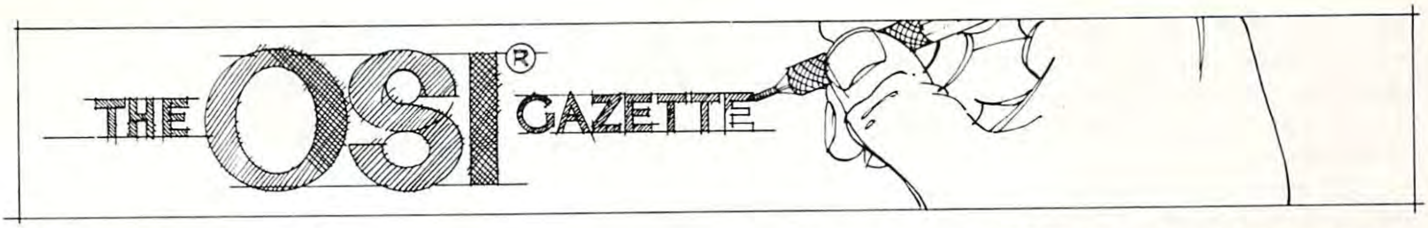
Each byte of memory in the Atari consists of 8 BITS (BInary digiTS - 1's or 0's). The lower 7 bits of memory location 77 are used to count from 0 to 127. When the count reaches 127, all of these 7 bits are binary 1's. Adding 1 more causes the 8th (uppermost) bit to change from 0 to 1 and this triggers the random color switching hardware. At 4 to 4.5 seconds per increment, it takes about 9 minutes for the computer to count from 0 to 127. Any number in location 77 from 128 to 255 will cause the upper bit to be set to 1 and the random color switching to occur. To see this, stop the program from running (with the BREAK key) and enter the following:

POKE 77,217

When you type this and press RETURN, the screen immediately starts switching colors because 217 is between 128 and 255 and has caused the upper bit of memory location 77 to be set to 1.

So any BASIC program can be modified to prevent the random color switching from occurring by periodically executing a POKE 77,0 to reset location 77 to zero and prevent it from reaching 128. Now you say, 'Won't this stop the computer from doing its color switching if I leave it for over 9 minutes with that program running and thus defeat the purpose of this hardware feature?' The answer is no, not if the POKE statement is placed in the program properly. When the program is idle, it will probably be in a loop waiting for paddle or joystick inputs. Just make sure the POKE statement is not put in this loop. Place it elsewhere in the program where it will be executed frequently. Then, if the computer is left idle while the program is running, it will not execute the POKE statement and random color switching will take place after 9 minutes. When the player returns, as soon as the joystick or paddle is moved or a trigger button pressed, the idle loop will be terminated and the POKE statement executed, returning the screen to normal colors. If the POKE statement is put in the right place, it can be made to appear as if the paddle or joystick acted just like a keypress in restoring the screen to normal.

Now that the function of RAM location 77 has been revealed, the random color switching feature of the Atari computer can be put back into its proper place. It won't need to be a bother any longer by popping up unexpectedly at the wrong moment! Happy programming. ©



A Small Operating System: OS65D The Kernel

Part 3 of 3

Tom R. Berger
School of Math
University of Minnesota
Minneapolis, MN

Concluding Remarks

OS65D is a very small operating system. It is in no sense 'generalized' to run with a large variety of software or peripherals as, say, Digital Research's CP/M is for the Z80. If software and peripherals other than those supplied by OSI are to be used, then the operating system must be modified. There are advantages and disadvantages to such an operating system. Disadvantages result from its inherent inflexibility and lack of generalized commands. On the other hand, because the operating system is so very small and easy to understand, for those who choose to understand it, it is easy to modify to suit personal needs: a definite advantage.

Let's look now at some 'features' not available in OS65D. Essentially all the operating system is in memory at all times. This creates minor problems with peripherals and INPUT/OUTPUT. For example, the original conception by OSI of I/O leads to a sequence of routines exactly filling the I/O space. Time has shown that OSI did not make the perfect choice for all situations. In particular, the real time version of OS65D requires that certain of the I/O routines be partially overlaid or omitted to make room for expansions of other I/O routines. The missing routines are not easily returned except by special allocation. A more generalized system would have an area of memory for I/O routines (just as OS65D does), but this area would not have fixed routines in it. I/O routines would be written to run at any location and would be loaded into the special space from the disk when they were needed, and where a niche was available. After they had served their purpose, the space they occupy would become available for other routines. This 'generalized' approach eases I/O problems, but requires much additional coding to handle all the loading and space allocation.

The disk handling routines could not be made much more compact. In particular, many user functions are left out. Thus the user must do a large amount of housekeeping not required on larger systems. The most glaring deficiency is the file creation process. You cannot create a file until you know its size. Usually, you cannot know its size until it is in memory; but the file creation utility occupies the same space as the file. As a result, a scratch file must be created in order to temporarily save programs while a permanent file of the correct size is created. The process becomes even more involved if you wish to expand a current file beyond its current size.

If you use BASIC programs which process many files, then the error recovery process of OS65D is far too simple. If BASIC calls an operating system command (say DISK!"blah blah") and an error occurs, this error is often nonrecoverable. That is, the stack is reset and return to BASIC occurs through the WARM START. This often means your program will bomb if you try to CONTINUE. If you have a large amount of information stored in BASIC strings and in the process of saving it encounter a disk error, then without a great deal of knowledge about the internal working of BASIC, your information is lost.

Most file handling is done with BASIC utilities. If you are programming in assembly language, this leads to endless shuffling back and forth from BASIC to the Assembler and back.

The operating system lacks an adequate editor. Thus the Assembler and BASIC must contain their own editors. As a consequence, all input must be acceptable to one of these two editors if it is to be processed. In particular, line numbers are needed. A

BASIC program can be created to solve this numbering problem, but BASIC may be too slow. Solving this new problem leads to further complications which would not be necessary with a good operating system editor.

There are certain philosophical advantages to a small operating system. OS65D is small enough that its entire operation can be understood at once. This means hackers can modify and alter the system, not just by POKES and patches, but fundamentally, to suit their own needs. In my experience, most hobby OSI computer owners aspire to or already fall in this hacking category. The smallness of the system puts the user in direct contact with the most fundamental operating system commands and operations. Even though it is slightly more involved, this gives the user the very maximum of control over the system.

This article was written using disassemblies of OS65D V3.2 (NMHZ) Release November 1979. Future articles will cover: (1) the I/O routines; (2) the Disk routines; (3) the ROM, and (4) miscellaneous bits and pieces. The disassemblies I have made are fully annotated (by hand) and are available for those who would like to use them. Send a stamped, self-addressed postcard to me to determine availability.

Tom Berger
10670 Hollywood Blvd.
Coon Rapids, MN 55433

©

A Six-Gun Shootout Game For The OSI C1P

Charles L. Stanford

The Six Gun Shootout game is a very pleasant and fun activity, particularly for the six to twelve or so age group. But this article concerns more than just the mechanics of writing another BASIC game for the C1P. When I originally wrote the program almost two years ago, we were reasonably satisfied with it. Sure, it was slow. Every time a player moved his gunfighter up or down the screen, the graphics POKEs took a lot longer than desired. And remembering that the "1" key was UP and the "2" key was down took a lot away. Those of you who have seen my articles on Fast Graphics (COMPUTE II Issue 3) and on interfacing the Atari Joystick to the C1P (COMPUTE Issue 7) can grasp what happened. Making that program work like it should has taught us more about the workings of the machine, over the past year, than any dozen manuals or articles.

This article, then, is a summing up of the methods we used to speed up both the software and the hardware to make BASIC games both more fun and much more saleable in the not inconsiderable Software marketplace.

BASIC Program Description

The game runs much as the early Arcade versions did. Each player has his gunfighter, who can shoot across the screen. Three Cacti obstruct some of the view, and move to a new location after each shot. Each player can move up or down, and shoot. Each gets 15 shots, and 5 hits wins.

The BASIC program shown in Listing 1 is fairly well annotated with REMs, but a few of the routines bear some discussion. The initialization starting at Line 5 sets the screen up as though no joysticks were available. This was deliberate, and makes the game more universally useful. It is a good idea to do this on all games, whether for paddles or for joysticks. The scoring from Line 200 is handled indirectly through the Fast Graphics Machine Language subroutine. Thus the POKEs of the ASCII characters are to that program rather than to the

screen. One routine, the "man dead" sequence, is still done in BASIC POKEs as a delay is called for here anyway.

The Joystick Interface

As described in the previous article mentioned above, the joysticks are interfaced to the keyboard such that any position can be directly related to the pressing of one or two keys. While the Atari Joysticks have eight positions around the center, only two of them are used in this game, and the others are "masked" out of existance. This is done with the routines beginning at Lines 700 and 750. Line 710's POKE K, 127 activates only Row 7 of the Keyboard. The first statement of Line 720 ORs away any columns except 5, 4, and 3, by forcing 1s into the others. Thus only keys 3, 4, and 5 are accepted as valid inputs. The next two statements of that line mask all but Column 5, so that a "shoot" command gets precedence over a "move" command. Finally, the other two keys are examined in Lines 725 and 730, and one of the move routines is addressed.

The routine at 750 works the same, except that Row 6 is activated, and the other player gets his chance. Each player is queried in turn, so one guy can't stand there and fill the other full of holes. The joystick works the same way as the keyboard, and is certainly a lot easier to use without a lot of practice, especially by the younger players.

Fast Graphics for the Six Gunners

The machine language graphics is done exactly the same way as the Choo Choo Collision demonstration program of the other previous article. A standard routine, shown in Listing 2, addresses a table of graphics symbols. These symbols are tailored for any game or other graphics screen display as shown in Figure 1. First, the Graphics Reference Manual is used to "draw" the characters desired, using a grid of sufficient size. Don't worry about screen location. The BASIC program takes care of that, by POKEing the table. Just lay out the characters, Determine the addresses of each of the elements of each character relative to the upper left corner of its grid, and couple that with the character code in making up the table. Each character should be ended with an #$FE (if there are more characters) or an #$FF (for the last character, or to end the routine).

Going Farther

You can just enter this program as-is, and have another nice game for your collection. Or you can dig a lot deeper, and quite possibly learn some techniques that will improve both your programming ability and some of those other games that run a bit slow, or get tiresome because the keyboard sequence is hard to use and remember. Anyone wishing to gain a deeper understanding of either the hardware or the software concepts described here should most certainly look to the other articles referenced.

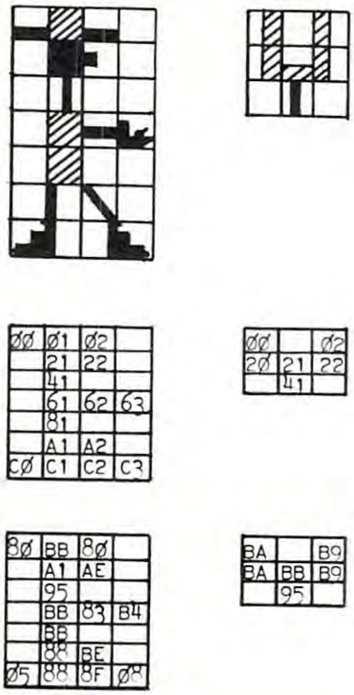


Figure 1. Graphics Development

Listing 1. Basic Program

```
5 FORS=ØTO9:PRINT:NEXT
7 REM-FOR ATARI JYSTKS
1Ø PRINT" SIX-GUN DUEL GAME
15 PRINT:PRINT:PRINT
2Ø PRINT"LEFT","RIGHT"
25 PRINT"----","-----
3Ø PRINT"  4      UP","    :
35 PRINT:PRINT"  3      DOWN","   Ø
4Ø PRINT:PRINT"  5      SHOOT","    -
45 PRINT:PRINT:PRINT"YOU HAVE 15 SHOTS EACH.
5Ø PRINT:PRINT:PRINT"FIVE HITS WINS!
55 PRINT:PRINT:PRINT"HIT SPACEBAR TO START
6Ø GOSUB1ØØ
65 GOSUB8ØØ
7Ø Z=1:L=Ø:R=Ø:SL=15:SR=15
75 POKE53Ø,1:K=57ØØ8:C=53445:D=54ØØ9
8Ø POKEK,253:IFPEEK(K)=239THEN9Ø
85 GOTO8Ø
9Ø X=USR(X):GOSUB2ØØ
95 GOTO7ØØ
1ØØ REM-MACH GRAPHICS
11Ø RESTORE
115 POKE11,34:POKE12,2:POKE254,96:POKE255,2
12Ø FORP=ØTO61:READM:POKE546+P,M:NEXT
13Ø DATA16Ø,Ø,169,32,153,Ø,211,153,Ø,21Ø,153,Ø,2Ø9,153,Ø
135 DATA2Ø8,2ØØ,2Ø8,241,234,234,234,16Ø,Ø,177,254,141,86,2,2ØØ
14Ø DATA177,254,141,87,2,2ØØ,177,254,17Ø,2ØØ,224,254,24Ø,236,224,255
145 DATA24Ø,8,177,254,2ØØ,157,68,2Ø9,2Ø8,236,96,234,234,234,234,234
148 REM-FIGURES NEXT
149 POKE133,255:POKE134,31:FORP=ØTO152:READM:POKE6Ø8+P,M:NEXTP
15Ø DATA197,2Ø8,Ø,128,1,187,2,128,33,161,34,174,65,149,97,187
155 DATA98,131,99,18Ø,129,187,161,136,162,19Ø,192,5,193,136,194,143
16Ø DATA195,8,254,17Ø,2Ø8,Ø,186,2,185,32,186,33,184,34,185,65
165 DATA149,254,174,2Ø9,Ø,186,2,185,32,186,33,184,34,185,65,149
17Ø DATA254,178,21Ø,Ø,186,2,185,32,186,33,184,34,185,65,149,254
175 DATA249,21Ø,1,128,2,187,3,128,33,173,34,161,66,149,96,181
18Ø DATA97,131,98,187,13Ø,187,161,189,162,143,192,5,193,136,194,143
185 DATA195,8,254,136,211,Ø,49,1,53,4,48,1Ø,48,13,49,14
19Ø DATA53,255,255,255,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
195 DATAØ,1,2,3,4,5,6,7,8,9,1Ø,11,12,13,14,15
199 RETURN
2ØØ REM-SCORE
21Ø POKE73Ø,48+L:POKE732,48+R
22Ø IFL=5ORR=5THEN29Ø
23Ø M=INT(SL/1Ø):N=SL-M*1Ø:IFM=ØTHENM=-16
24Ø U=INT(SR/1Ø):V=SR-U*1Ø:IFU=ØTHENU=-16
25Ø POKE726,48+M:POKE728,48+N
26Ø POKE734,48+U:POKE736,48+V
27Ø IFSL=ØANDSR=ØTHEN29Ø
28Ø RETURN
29Ø X=USR(X):GOTO6Ø
3ØØ REM-LEFT MAN DEAD
3Ø5 C=C+16Ø
31Ø POKEC+2,32:POKEC-31,32:POKEC-61,32:POKEC-62,32:POKEC-63,32
315 POKEC-95,32:POKEC-126,32:POKEC-127,32:POKEC-158,32:POKEC-159,32
32Ø POKEC-16Ø,32
325 POKEC,143:POKEC+1,171:POKEC+3,14Ø
33Ø POKEC+32,187:POKEC+33,161:POKEC+34,148
335 POKEC+35,187:POKEC+36,187:POKEC+37,128
34Ø POKEC+38,176:POKEC+64,143
345 FORT=ØTO5ØØ:NEXT:C=C-16Ø:RETURN
35Ø REM-RIGHT MAN DEAD
355 D=D+156
36Ø POKED+3,32:POKED+4,32:POKED-28,32:POKED-6Ø,32:POKED-61,32
365 POKED-62,32:POKED-92,32:POKED-124,32:POKED-125,32
```

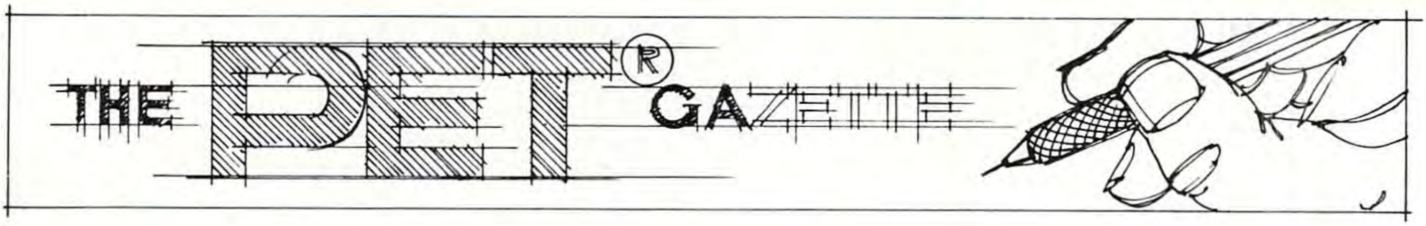
```
370 POKED-155,32:POKED-156,32:POKED-157,32
375 POKED+3,139:POKED+5,171:POKED+6,136
38Ø POKED+32,178:POKED+33,128:POKED+34,187:POKED+35,187
385 POKED+36,148:POKED+37,161:POKED+38,187
39Ø POKED+7Ø,136
395 FORT=ØTO5ØØ:NEXT:D=D-158:RETURN
4ØØ IFSL=ØGOTO75Ø
4Ø5 Q=Ø
41Ø FORX=1TO3
415 IFC<E(X)-32ANDC>E(X)-128THENG=2:GOTO445
42Ø NEXT
425 IFC<D-96ORC>D+32THENG=23:GOTO445
43Ø G=23:GCSUB46Ø
435 GOSUB35Ø
44Ø L=L+1:GOTO45Ø
445 GOSUB46Ø
45Ø SL=SL-1
455 GOSUB82Ø:GOSUB2ØØ:GOTO69Ø
46Ø FORX=C+1ØØTOC+1ØØ+G
465 POKEX,45
47Ø NEXT
475 FORT=ØTO2ØØ:NEXT
48Ø RETURN
5ØØ IFSR=ØGOTO7ØØ
5Ø5 Q=1
51Ø FORX=1TO3
515 IFD<E(X)ANDD>E(X)-96THENG=2:GOTO545
52Ø NEXT
525 IFD<C-64ORD>C+64THENG=23:GOTO545
53Ø G=23:GOSUB56Ø
535 GOSUB3ØØ
54Ø R=R+1:GOTO55Ø
545 GOSUB56Ø
55Ø SR=SR-1
555 GOSUB8ØØ:GOSUB2ØØ:GOTO64Ø
56Ø FORX=D+94TOD+94-GSTEP-1
565 POKEX,45
57Ø NEXT
575 FORT=ØTO2ØØ:NEXT
58Ø RETURN
6ØØ REM-LEFTMOVE
61Ø Q=Ø:IFC<53443THEN75Ø
62Ø C=C-32:GOTO64Ø
63Ø Q=Ø:IFC>54Ø19THEN75Ø
635 C=C+32
64Ø H=C:GCSUB9ØØ:POKE6Ø9,J:POKE6Ø8,M:X=USR(X):IFQ=1THEN7ØØ
645 GOTO75Ø
65Ø REM-RIGHTMOVE
66Ø Q=1:IFD<53443THEN7ØØ
67Ø D=D-32:GOTO69Ø
68Ø Q=1:IFD>54Ø19THEN7ØØ
685 D=D+32
69Ø H=D:GOSUB9ØØ:POKE689,J:POKE688,M:X=USR(X)
695 IFQ=ØTHEN75Ø
7ØØ REM-LEFT INPUT
71Ø POKEK,127
715 Y1=PEEK(K)
72Ø Y1=Y1OR199:Y2=Y1OR247:IFY2=247THEN4ØØ
725 IFY1=239THEN6ØØ
73Ø IFY1=223THEN63Ø
735 GOTO75Ø
75Ø REM-RIGHT INPUT
76Ø POKEK,191
765 Z1=PEEK(K)
77Ø Z1=Z1OR199:Z2=Z1OR247:IFZ2=247THEN5ØØ
775 IFZ1=239THEN65Ø
78Ø IFZ1=223THEN68Ø
785 GOTO7ØØ
8ØØ REM-CACTUS LOCATOR
81Ø EE=53414
82Ø FORX=1TO3
83Ø E(X)=EE+32*INT(22*RND(1))
84Ø E(X)=E(X)+X*4
85Ø H=E(X):GCSUB9ØØ
86Ø ONXGOSUB87Ø,88Ø,89Ø
865 NEXTX:RETURN
87Ø POKE644,J:POKE643,M
875 RETURN
88Ø POKE659,J:POKE658,M
885 RETURN
89Ø POKE674,J:POKE673,M
895 RETURN
9ØØ REM-CHAR LOC SUB
91Ø J=INT(H/256)
92Ø M=H-J*256
93Ø RETURN
99Ø END

OK
```

```
Ø222 AØ ØØ    LDY,I
Ø224 A9 2Ø    LDA,I
Ø226 99 ØØ D3 STA,A,Y
Ø229 99 ØØ D2 STA,A,Y
Ø22C 99 ØØ D1 STA,A,Y
Ø22F 99 ØØ DØ STA,A,Y
Ø232 C8       INY
Ø233 DØ F1    BNE
Ø235 EA       NOP
Ø236 EA       NOP
Ø237 EA       NOP
Ø238 AØ ØØ    LDY,I
Ø23A B1 FE    LDA,(I),Y
Ø23C 8D 56 Ø2 STA,A
Ø23F C8       INY
Ø24Ø B1 FE    LDA,(I),Y
Ø242 8D 57 Ø2 STA,A
Ø245 C8       INY
Ø246 B1 FE    LDA,(I),Y
Ø248 AA       TAX
Ø249 C8       INY
Ø24A EØ FE    CPX,I
Ø24C FØ EC    BEQ
Ø24E EØ FF    CPX,I
Ø25Ø FØ Ø8    BEQ
Ø252 B1 FE    LDA,(I),Y
Ø254 C8       INY
Ø255 9D 88 D3 STA,A,X
Ø258 DØ EC    BNE
Ø25A 6Ø       RTS
```

Listing 2. Machine Language Subroutine

```
Ø26Ø  C5 DØ          Ø2A1  32 D1
Ø262  ØØ 8Ø          Ø2A3  ØØ BA
Ø264  Ø1 BB          Ø2A5  Ø2 B9
Ø266  Ø2 8Ø          Ø2A7  2Ø BA
Ø268  21 A1          Ø2A9  21 B8
Ø26A  22 AE          Ø2AB  22 B9
Ø26C  41 95          Ø2AD  41 95
Ø26E  61 BB          Ø2AF  FE
Ø27Ø  62 83          Ø2BØ  F9 D2
Ø272  63 B4          Ø2B2  Ø1 8Ø
Ø274  81 BB          Ø2B4  Ø2 BB
Ø276  A1 88          Ø2B6  Ø3 8Ø
Ø278  A2 BE          Ø2B8  21 AD
Ø27A  CØ Ø5          Ø2BA  22 A1
Ø27C  C1 88          Ø2BC  42 95
Ø27E  C2 8F          Ø2BE  6Ø B5
Ø28Ø  C3 Ø8          Ø2CØ  61 83
Ø282  FE             Ø2C2  62 BB
Ø283  8A D1          Ø2C4  82 BB
Ø285  ØØ BA          Ø2C6  A1 BD
Ø287  Ø2 B9          Ø2C8  A2 8F
Ø289  2Ø BA          Ø2CA  CØ Ø5
Ø28B  21 B8          Ø2CC  C1 88
Ø28D  22 B9          Ø2CE  C2 8F
Ø28F  41 95          Ø2DØ  C3 Ø8
Ø291  FE             Ø2D2  FE
Ø292  2E D2          Ø2D3  88 D3
Ø294  ØØ BA          Ø2D5  ØØ 31
Ø296  Ø2 B9          Ø2D7  Ø1 35
Ø298  2Ø BA          Ø2D9  Ø4 3Ø
Ø29A  21 B8          Ø2DB  ØA 3Ø
Ø29C  22 B9          Ø2DD  ØD 31
Ø29E  41 95          Ø2DF  ØE 35
Ø2AØ  FE             Ø2E1  FF
```

Listing 3. Machine Language Graphics Table ©



Keyprint Revisited

Eric Brandon
Islington, Ontario
Canada

When I first saw KEYPRINT by Charles Brannon in the NOV/DEC 1980 issue of COMPUTE! I thought my printer problems were over. Not so! As I read on, I discovered that KEYPRINT was only for the new upgrade ROMs, and I had the old original ROMs. Furthermore, I knew no machine or assembly language. I could:

a) Get new ROMs (and give up half my program library).

b) Learn assembly language and modify KEYPRINT for the old PETs.

c) Give up.

I chose solution b. I purchased books, assemblers, disassembler, and all the paraphenalia associated with assembly language programming. Here is the result of my efforts: KEYPRINT for old ROM PETs.

```
C*    PC   SR AC XR YR SP
.:   C6ED 00 38 00 32 FE
.    M 033A 03CF
           0  1  2  3  4  5  6  7
.:   033A 78 A9 03 8D 1A 02 A9 47
.:   0342 8D 19 02 58 60 AD 03 02
.:   034A C9 45 D0 03 20 54 03 4C
.:   0352 85 E6 A9 80 85 20 A9 00
.:   035A 85 1F A9 04 8D 64 02 85
.:   0362 F1 20 BA F0 20 32 F1 A9
.:   036A 19 85 22 A9 0D 85 21 20
.:   0372 D2 FF A9 11 AE 4C E8 E0
.:   037A 0C D0 02 A9 91 20 D2 FF
.:   0382 A0 00 B1 1F 29 7F AA B1
.:   038A 1F 45 21 10 0B B1 1F 85
.:   0392 21 29 80 49 92 20 D2 FF
.:   039A 8A C9 20 B0 04 09 40 D0
.:   03A2 0E C9 40 90 0A C9 60 B0
.:   03AA 04 09 80 D0 02 49 C0 20
.:   03B2 D2 FF C8 C0 28 90 CB A5
.:   03BA 1F 69 27 85 1F 90 02 E6
.:   03C2 20 C6 22 D0 A6 A9 0D 20
.:   03CA D2 FF 4C CC FF 67 54 00
.     \
```

First, type in the hexadecimal (base 16) code with your monitor. If you don't know how to do this, consult your (or anyone else's PET manual.

Once you have entered it, type:

M 033A 03CF

and compare what you see with what is on this page. If they don't correspond exactly, either fill in what doesn't match (remember to hit RETURN at the end of each line), or start over.

When KEYPRINT is in memory correctly, type:

S 01, KEYPRINT, 033A, 03CF

to save it on tape.

An 'X' command will get you out of the monitor. Type:

SYS 826

to initialize KEYPRINT. The cursor should reappear almost instantly. If it doesn't, you have made a mistake in the first 12 bytes (numbers). LOAD it, and check again.

Hopefully, your cursor came back. If it did, hit the '\' key and your screen should be dumped on the printer. If it doesn't, you have made one of the following mistakes:

a) Typing error.
SOLUTION: Start again.

b) Your printer has a secondary address other than 4. SOLUTION: POKE 861,SA

c) You forgot to initialize.
SOLUTION: SYS 826 and hit the key again.

d) You hit the wront key.
SOLUTION: Hit the key to the right of the ampersand.

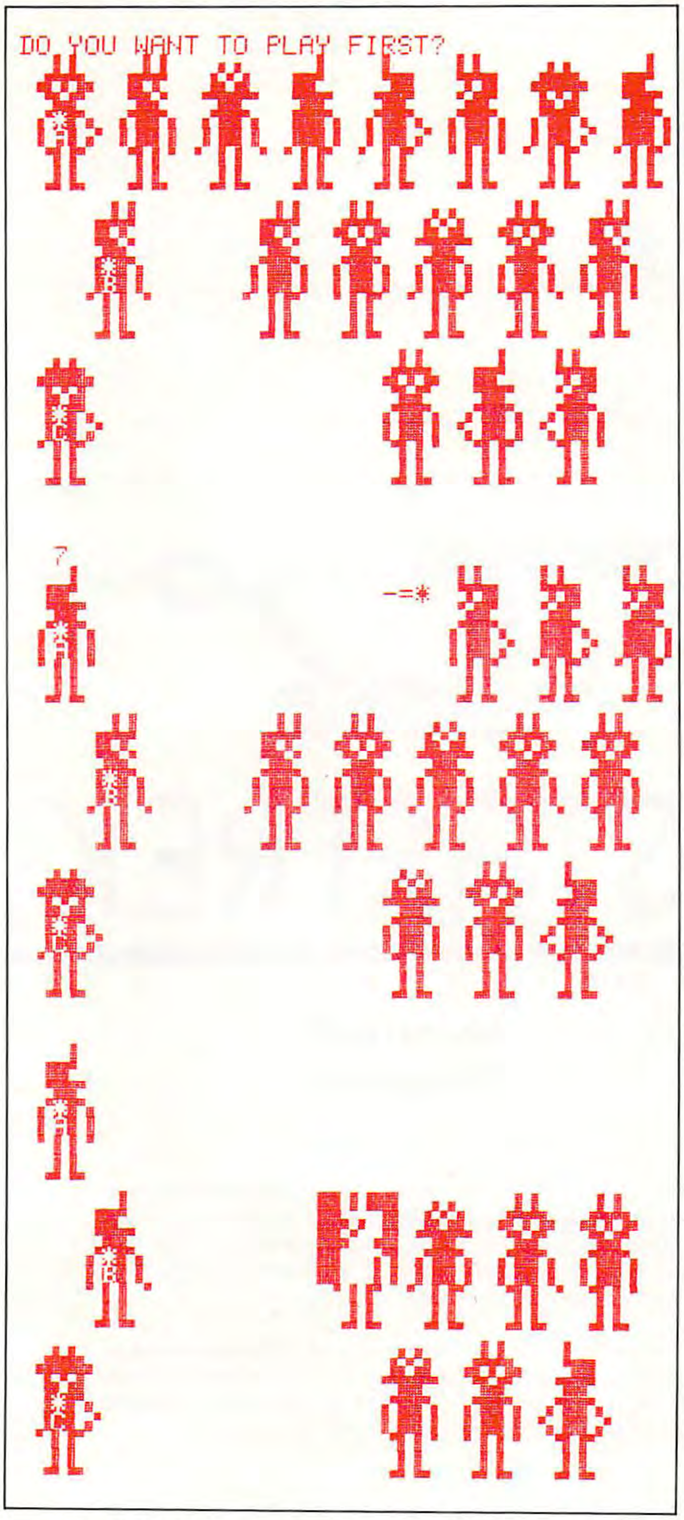
e) You tried this program on something other than an old ROM PET.
SOLUTION: Move on to the next article.

You can also make KEYPRINT work without hitting a key (it doesn't even have to be initialized) with an SYS 852.

When KEYPRINT is active (hitting ' ' will make it work), the PET will neither LOAD nor SAVE properly. There are two ways of deactivating KEYPRINT:

a) POKE 537,133:POKE 538,230

b) Typing LOAD or ◄SHIFT► RUN/STOP, pushing PLAY, FAST FORWARD, or REWIND on the cassette and BREAKing it with the stop key. If the cassette motor doesn't run before you BREAK the LOAD, KEYPRINT will not be deactivated.



KEYPRINT can always be revived with an SYS 826.

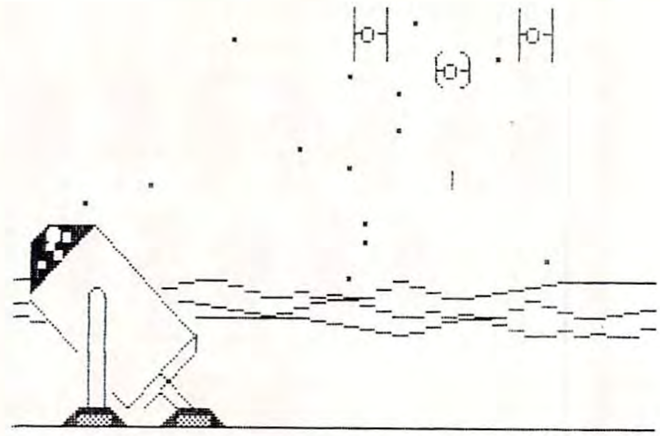
Finally, memory location 843 contains the number representing the key that must be hit to dump the screen. To change the key, Type:

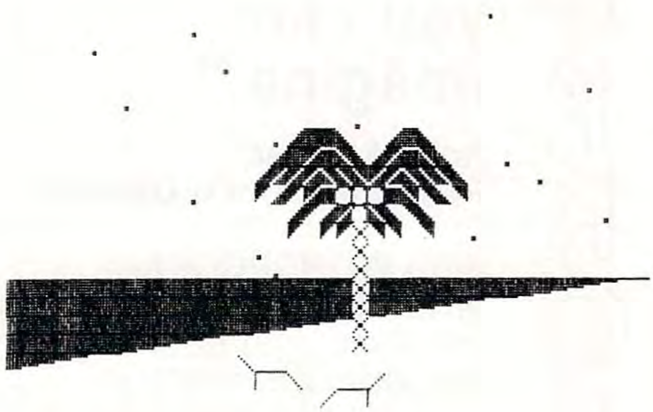
FOR T = 1 TO 1E6:PRINT PEEK(515):NEXT T

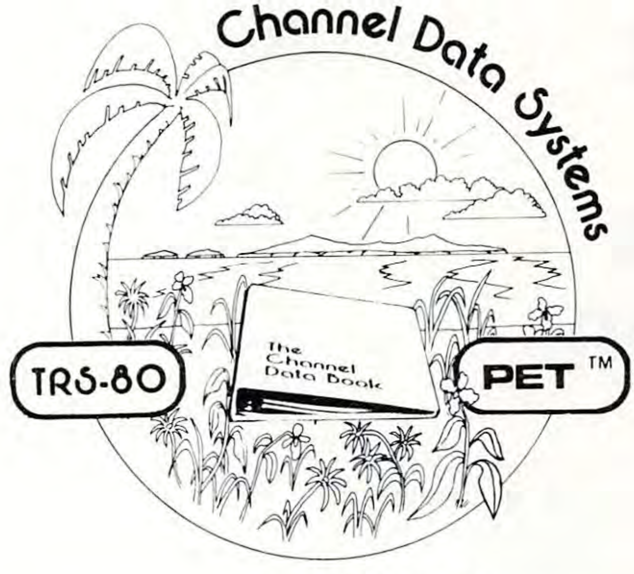
You will see a column of 255s going up the screen. Hit the key you wish to assign as THE key. The 255s will change into another number, Remember that number. BREAK the loop with the STOP key, and POKE 843,n where n is the number you saw when you hit the key.

I hope you find the program useful. I would like to thank Jim Butterfield for sharing some memory locations with me. If anyone wishes more information on the program, or if anyone wants to trade their programs with me please write to:

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







OMNIFILE — CBM or TRS-80 — $30

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

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

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

EXPLORE — CBM — $15

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

CHANNEL DATA BOOK for PET — CBM — $20

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

ACCESSORIES

PORT NOISE COMPLETE — Speaker-amplifier for Commodore PET or CBM with connector (M & N pin) all ready to plug in—$20.

IEEE/USER PORT CONNECTOR — 24 pin connector. With backshell—$7. Without backshell—$4.

C-10 TAPE CASSETTES — High quality AGFA tape with screw type case. $10/10 or $75/100.

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

CHANNEL DATA SYSTEMS

5960 Mandarin Ave. Goleta, CA 93117 805-964-6695

Learning About Garbage Collection

Jim Butterfield

If you are blessed with Commodore's newest ROM 4.0 system, you won't need to worry about garbage collection. But users with Original and Upgrade ROMs will run into it, and they will find it worthwhile to learn more about how it works.

Garbage collection is misnamed. It should be called garbage disposal or preferably memory reclaiming. Whatever you call it, the symptoms are highly visible and annoying: a long pause during which the computer appears to be dead.

There are methods to overcome many garbage collection delays. First, however, it's worthwhile looking into what causes it and how it behaves. We'll perform a series of experiments to disclose the characteristics of garbage collection.

Part 1: Experiments

Type in the following program:

```
100 DIM A$(255)
110 FOR J = 1 TO 255
120 A$(J) = "A" + "B"
130 NEXT J
500 REMARK: FORCE COLLECTION WITH FRE(0)
510 PRINT "STARTING"
520 Z = FRE(0)
530 PRINT "FINISHED"
```

Type RUN. There will be a pause of over five seconds between the printing of the words STARTING and FINISHED. This is the infamous garbage collection pause; while it's in progress, the RUN/STOP key doesn't work and the computer appears to be dead.

Note that there is in fact no garbage to be collected: all the strings we have manufactured are still live. But the delay is still there.

Conclusion #1: *You can have substantial garbage collection delays even when you have little or no garbage.*

Now that the program has run, type GOTO 500. Garbage collection will take place again on the same strings. It's just as long as the first time.

Conclusion #2: *You don't save time on a garbage collection even though your strings were collected recently.*

Add the following lines to the above program:

```
200 FOR J = 1 TO 255
210 A$(J) = "AB"
220 NEXT J
```

Type RUN. The words STARTING and FINISHED print with very little delay between them. The garbage collection delay has vanished!

What has happened here? The string AB in line 210 is used exactly where it lies in the Basic program; there's no need to repack it into "dynamic string memory". As a result, this type of string doesn't need collection.

In contrast, the string built in line 120 had to be manufactured by concatenation, and thus needed to be stored in general memory.

Conclusion #3: *Strings supplied within the program don't contribute to garbage collection delays. This also applies to strings supplied within DATA statements.*

If you listed the program as we have run it so far, you'll see that we have created a good deal of garbage. All of the strings generated by line 120 were later thrown away and replaced by the strings in line 210. Yet there was almost no garbage collection delay.

Conclusion #4: *Garbage (abandoned strings) don't contribute much to garbage collection delay. Only the strings you keep cost you time.*

Now let's change two lines of our program to increase the number of strings we are generating. Change the following lines:

```
100 DIM A$(255), B$(255)
 .. .. .. .. ..
210 B$(J) = LEFT$("HELLO",4)
```

This time, we're going to manufacture twice as many strings. Should we expect the garbage collection time to double over our previous five seconds?

Type RUN and see.

This time, garbage collection took over twenty seconds.

Conclusion #5: *Garbage collection time is proportional to the square of the number of dynamic (manufactured) strings.*

Now for the final experiment. Type in the following lines:

Original ROM:

```
450 X1 = PEEK(134) : X2 = PEEK(135)
460 Y1 = PEEK(130) : Y2 = PEEK(131)
470 POKE 134,Y1 : POKE 135,Y2
600 POKE 134,X1 : POKE 135,X2
```

Upgrade ROM:

```
450 X1 = PEEK(52) : X2 = PEEK(53)
460 Y1 = PEEK(48) : Y2 = PEEK(49)
470 POKE 52,Y1 : POKE 53,Y2
600 POKE 52,X1 : POKE 53,X2
```

What will these additions do? Just before garbage collection begins, it sets the top-of-Basic memory

pointer lower. After garbage collection completes, it restores the pointer to its original value.

There are the same number of strings as previously, so it seems that garbage collection time should not be affected, and should stay at twenty seconds or so.

Type RUN. Surprise! Garbage collection time drops to zero!

Conclusion #6: *Garbage collection is not performed on strings located above the top-of-Basic-memory.*

The strings are not affected — but no garbage collection took place up there either, so that unwanted strings would not be discarded.

Part 2: Techniques For Reducing Garbage Collection Time

Case 1: Eliminating concatenation garbage.

Suppose we're inputting a string and using concatenation to put it together. Sample coding might be:

```
800 REMARK: INPUT STRING
810 A$ = "" : rem start with null string
820 GET B$: IF B$ = "" GOTO 820
830 IF B$ = CHR$(13) GOTO 850
840 A$ = A$ + B$ : GOTO 820
850 REMARK: A$ CONTAINS OUR INPUT
```

The problem here is that this type of concatenation lays waste a lot of memory. If our input is HELLO, ROBERT, the variable A$ will first be set to H, then to HE and so on until the full thirteen characters are received. Over seventy locations will end up containing abandoned strings; and if our input string were fifty characters long we'd create over a thousand bytes of garbage. This kind of thing can trigger automatic garbage collection very quickly.

A little perspective: if A$ and B$ were our only string variables, we'd have nothing to worry about. Garbage collection would be almost instantaneous. But if we had hundreds of other strings lying about, they would all go through the collection process, and we'd be in time trouble.

Solution: Before we enter this string-wasting routine, insert (at line 805) coding to move the top-of-Basic-memory pointer down. Let the concatenation program run; when it is finished (line 850), force a tiny collection with Z = FRE(0) and then restore the top-of-Basic-memory pointer. Refer back to the experiments for the technique.

Case 2: Reading in a batch of new strings from a file.

Suppose we read in a whole flock of strings dealing with a customer account and place them in one or more arrays. No problem so far: the strings will read in neatly from a file and there will be little waste space.

Now assume that we've finished with that customer and the program goes back to read in material for the next account. Danger! The old strings are still there, taking up waste space. As we read in new material, we may run short of room, and garbage collection will automatically be called in. It will collect the new strings and quite a few of the old ones that we haven't discarded yet. Help!

Solution: Get rid of the old strings as soon as they are not needed by setting them to null strings (e.g., A$(J) = ""). Then, when your strings are at a minimum — just before reading in the new batch — force a collection with Z = FRE(0). Collection will be quick, since there are few live strings left, and the new information will read into freshly liberated memory.

Case 3: Shuffling strings around

There are times when you have a lot of strings in an array, and you want to change their order. The most usual case is that you want to sort them into some kind of order.

To exchange strings four and seven, you would tend to code something like:

```
700 X$ = X$(4)
710 X$(4) = X$(7)
720 X$(7) = X$
```

Unfortunately, this simple swap leaves three abandoned strings in memory: the old value of X$(4), the old value of X$(7), and X$, which will probably not be used again. We don't need to do much of this before garbage collection kicks in again.

Solution: Use a technique called an index array. Instead of changing the strings and causing garbage, change the index instead. The above coding will change to:

```
700 I% = I%(4)
710 I%(4) = I%(7)
720 I%(7) = I%
```

We must be careful to set up array I% at the start, so that I%(4) = 4, for example. At any time, we can call up string number four by referring to X$(I%(4)). Here's a simple example:

```
100 REMARK: SIMPLE BUBBLE SORT
110 DIM N$(20),I%(20)
120 PRINT "INPUT 20 NAMES:"
130 FOR J = 1 TO 20
140 I%(J) = J        : rem set up index
150 INPUT N$(J)      : rem get string input
160 NEXT J
200 F = 0
210 FOR J = 1 TO 19
220 IF N$(I%(J))< = N$(I%(J + 1))GOTO 250
230 F = 1
240 I% = I%(J) : I%(J) = I%(J + 1) : I%(J + 1) = I%
250 NEXT J
260 IF F = 1 GOTO 200
300 FOR J = 1 TO 20 : PRINT N$(I%(J)) : NEXT J
```

You can see that we never move a string, but the sort is performed. ©