Double-Density Graphing On The OSI C1P

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

Character Generation On The Atari

COMPUTE!

$2.50
February, 1981
Issue 9
Vol. 3, No. 2
63379

The Journal For Progressive Computing™

Ticker Tape Messages On The Atari

The 25¢ Apple II Real Time Clock

Mixing And Matching Commodore Disk Systems

Crash Prevention For Your PET

02
74470 63379

IF YOU'RE WAITING FOR THE PRICE OF WORD PROCESSORS TO FALL WITHIN REASON,

Everyone expected it would happen sooner or later... with WordPro PLUS™ it already has! Now all the marvelous benefits of expensive and advanced word processing systems are available on Commodore computers, America's largest selling computer line. WordPro PLUS, when combined with the new 80 column CBM 8032, creates a word processing system comparable to virtually any other top quality word processor available—but at savings of thousands of dollars!

New, low cost computer technology is now available at a fraction of what you would expect to pay. This technology allowed Commodore to introduce the new and revolutionary CBM 8032 Computer.

WordPro PLUS turns this new CBM 8032 Computer into a sophisticated, time saving word processing tool. With WordPro PLUS, documents are displayed on the computer's screen. Editing and last minute revisions are simple and easy. No more lengthy re-typing sessions. Letters and documents are easily re-called from memory storage for editing or printing with final drafts printed perfectly at over five hundred words per minute!

Our nationwide team of professional dealers will show you how your office will benefit by using WordPro PLUS. At a price far less than you realize.

Invest in your office's future...
Invest in WordPro PLUS...
Call us today for the name of the WordPro PLUS dealer nearest you.

Professional Software Inc.
166 Crescent Road
Needham, MA 02194
(617) 444-5224
TELEX: 95 1579

NOW THE SOFTCARD™ CAN TAKE YOU BEYOND THE BASICS.

You probably know about the SoftCard — our ingenious circuit card that converts an Apple II® into a Z-80® machine running CP/M®.

You may even know that with the SoftCard, you get Microsoft's powerful BASIC — extended to support Apple graphics and many other features.

Now, whenever you're ready to get beyond the BASICs, the SoftCard can take you into whole new realms. Starting with two advanced language packages from Microsoft.

FORTRAN AND COBOL TO GO.

Now you can run the world's most popular engineering/scientific language and the most popular business language on your Apple. Think what that means: you can choose from literally thousands of "off-the-shelf" applications programs, and have them working with little conversion. Or design your own programs, taking advantage of all the problem-solving power these specialized languages give you.

FORTRAN-80

A complete ANSI-standard FORTRAN (except COMPLEX type), with important enhancements. The extremely fast compiler performs extensive code optimization, and, since it doesn't require a "P-code" interpreter at run time, your programs will typically execute 2-3 times faster than with Apple FORTRAN.

FORTRAN is easy to learn if you know BASIC, and the package includes a huge library of floating point, math, and I/O routines you can use in all your programs.

COBOL-80

Virtually the only choice for serious business data processing. It's ANSI 1974 standard COBOL, with many user-oriented features added: formatted screen support for CRT terminals, simple segmenting of very large programs, powerful file handling capability, trace debugging, and much more. A separate Sort package is coming soon.

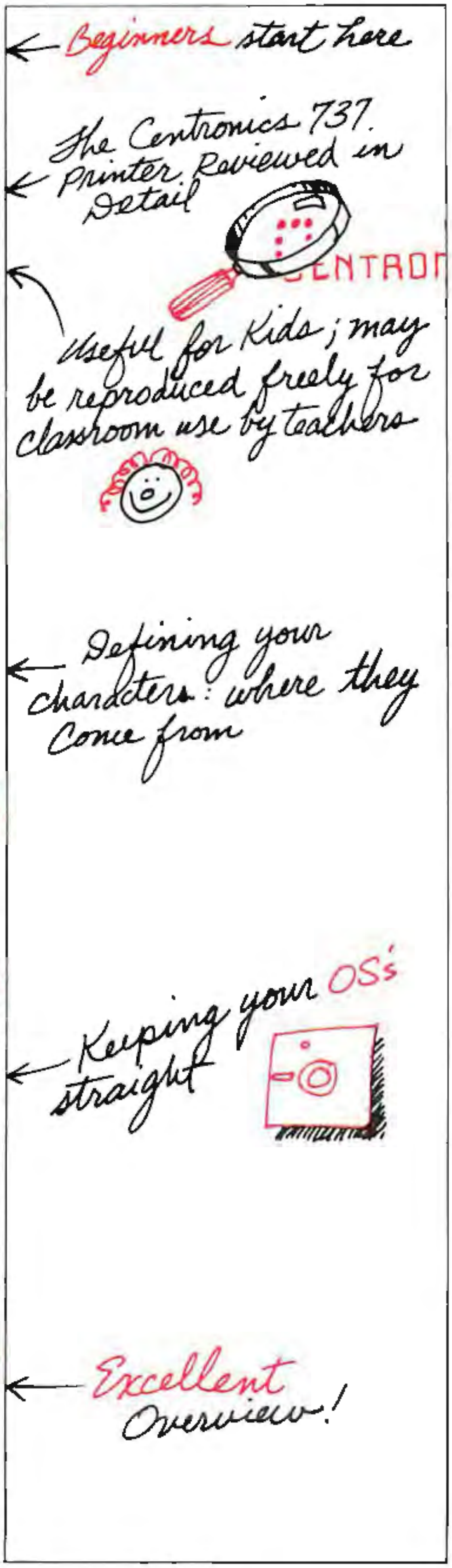
FORTRAN-80 and COBOL-80 are just two more reasons why the Apple with SoftCard is the world's most versatile personal computer. Get all the exciting details from your Microsoft dealer today. And start getting beyond the BASICs.

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

MICROSOFT

Table of Contents

February, 1981, Vol. 3. No. 2



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, $16.00. 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.

Robert C. Lock, Publisher/Editor
Joretta Klepfer, Manager, Dealer Marketing
Carol Holmquist Lock, Circulation Manager
Kathleen Martinek, Publication Assistant
J. Gary Dean, Art Direction/Production Assistance

COMPUTE receives continuing editorial assistance from the following persons:
Harvey Herman, University of North Carolina at Greensboro
Jim Butterfield, Toronto, Canada
Larry Isaacs, Raleigh, NC

The following writers contribute on a regular basis as Contributing Editors:
Al Baker, 2327 S. Westminster, Wheaton, IL 60187
Gene Beals, 115 E. Stump Road, Montgomeryville, PA 18936
Len Lindsay, 5501 Groveland Terrace, Madison, WI 53716
Roy O'Brien, P.O. Box 426, Beaumont, CA 92223

Subscription Information (12 Issue Year):
COMPUTE. Circulation Dept.
P.O. Box 5406
Greensboro, NC 27403 USA
U.S. $16.00
Canada $18.00 (U.S. funds)
Europe: Surface Subscription, $20.00 (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

The Editor's Notes

Robert Lock, Publisher/Editor

Too Few Ataris

Such a problem... We've been saying Atari sales are picking up, a more than gradual creep that's been in evidence since summer. The trickle has apparently turned into a roar: it seems the pipeline effectively ran dry in mid-December when dealers across the country were selling machines faster than they could get them. This translates, of course, into not selling machines, since many dealers were unable to obtain enough. We've heard from some who said they could have sold many more, given sufficient supply. Don't give up Santa; it would still be quite appropriate for Valentine's day. And I suspect Atari corporate won't be caught short again.

The International Commodore, Or, Bye Jack

We've been persisting in these pages with claims that Commodore's getting it together in marketing. With the help of Dr. Chip, we've been trying to track the rapid changes in mid to upper level management. Commodore has been growing up as a corporate entity, and such growth is invariably replete with problems in working out directions, helmsmen/women, and the like.

Jack Tramiel, President and founder of Commodore, has stepped out of the position of President. He will become Vice-Chairman of the Board of Directors and Chief Executive Officer. We suspect Jack's skills will be more directed to the long-range growth of the company, and less to the day to day operations and intermediate planning. In short, the move looks like a logical, progressive step in the growth of the company.

The new President appears to be exceptionally well qualified to head a company such as Commodore. His name is James Finke, and he comes to Commodore with a background that seems ideally suited. You have to understand that Commodore is truly an international company. The US has, in the past, made up a small portion of their overall market. Thus, they're relatively unique at the moment among the competitive 6502 machine vendors.

Their strength outside the US places them at number one in installed machines in Canada, England, Germany, and so on. In the US they've been running number three behind Tandy and Apple. In spite of efforts to the contrary, their steps for improving the marketing channels in the US have been slow going, with problems with dealer support and supply being foremost.

The point of all this is that the new President, for corporate stability, will have to be able to get things rolling in the US, while maintaining the superiority in Europe. Mr. Finke appears to have such a background. A 1951 Physics graduate of Williams College, a Masters Degree in International Economics from Oxford University, and a Harvard Law degree. A history of experience leading from Vice President and General Manager of Motorola's communication division European Operations to General Manager of General Electric Medical Divisions International Operations to Vice President and General Manager-Europe for Data General Corporation. We applaud Jack and the rest of the Commodore board, and wish Mr. Finke much success in his new position. We hope this move portends a new stability in product relations with customers and dealers.

A Major CAPUTE!

Remember that annual awkwardness when you have to change from one year to the next? You write checks with the wrong year on them, etc.? **COMPUTE!** is apparently no exception to the year-end transition oops. Our January issue proudly claimed it was 1980. Hopefully the cover on this one matches. And, oh yes... missile is not spelled missle. I don't suppose we could claim we wanted to see if you were on your toes?

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!**:

Organization

The front section of the magazine contains articles of general interest. These will vary from issue to issue with columns, business applications articles, general programming hints and educational articles. While an article may appear in this section that is machine specific, it's generally here because it has material of interest to other readers.

The balance of the magazine is organized into five Gazettes. These are, in order of appearance, Atari, OSI, Pet, and the Single-Boards (Aim, KYM, and SYM). Even if you're not the owner or user of a computer covered by a particular Gazette, you'll still find useful information there.

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.

The Readers Feedback and CAPUTE!

These two continuing features provide channels of communication with readers and authors. **The Readers Feedback** grows out of your comments provided via **The Editor's Feedback** card. You'll find one bound into every issue. Please use it. **CAPUTE!** is our collection ground for past errors and omissions. Here you'll find updates to previous program problems, etc.

©

The Reader's Feedback

Robert Lock and Readers

Our best article vote will take a one month sabbatical. Now that we're monthly, we're adding an extra month for vote gathering.

Why We're Here

The Reader's Feedback serves several purposes. The principal one is self-explanatory. I read every Editor's Feedback card that comes in, and your comments help me in defining/refining the direction and goals of the magazine.

We use the feedback as a means of showing authors and potential authors what kinds of material we're looking for. Frequently you as a reader, or as a group of readers, are quite precise at defining needs.

The Feedback cards are also a means of cluing me in on problems with vendors, problems with hardware and software, and specifically problems with any of our advertisers. Although we can't look into every possible problem, we do use the Feedback cards to show us potential problem areas. Our measure of this is generally quantity of responses.

Keep writing, and we'll keep reading. Thanks for your continued support. From our end we'll try to remain the best resource magazine around.

And Now Our Readers

I am a high school science teacher. I am a novice Apple Computer programmer. I would appreciate COMPUTE! articles designed to enhance the programming ability of novice Apple programmers... In-depth articles of Apple Poking, Peeking and Calls would be very helpful...

We are constantly looking for good material oriented at beginning and intermediate programmers. Tutorial articles are especially welcomed. I know there are experienced Apple programmers out there that could write the kinds of articles, short programming notes, and such that our reader above is talking about. Well group?

I'd like to see more articles on larger OSI Systems.

As with the Apple reader above, we're always looking for good OSI material. Educational and business users should remember that their applications articles can help other readers, even if they don't share a common machine. An article describing the method of developing a specific applications program can be of as much use to others as the specific program itself.

I have had an Atari for six months and if it wasn't for the computer magazines I would still be trying to count votes, etc. or measure a bicycle wheel... I bought a computer to expand knowledge and not play games. Why don't the software people realize this — if it wasn't for your writers I would feel I had a white elephant with 1 leg.

We try to present a mix of material in every issue that will be of use to our broad range of consumers of computers. Thus, an article on Player Missile Graphics, while immediately relevant to its title, is relevant to programmers developing applications programs that can become more useful by implementing these concepts. Atari is slowly releasing a business oriented applications library, and other vendors are getting involved as well. We would certainly like to see more applications programs submitted here.

A Call For Generality

In reflecting on the now final mix of this issue, I realize (as always) a few things to change next time around. The article on the line-oriented text editor is discussed for both Apple and PET. The program presented is for PET, with the author comment that the Apple version requires only I/O changes. The article wasn't supplied to us with those changes, and by the time I realized it, it was too late to get them... or to hold the article back. If you send us an article that's applicable to more than one 6502 machine (and that's the kind of article we dearly love to get), please make sure you include the versions for the various machines.

If you translate a program written for one machine in COMPUTE! so that it will run on your (different) machine, send it in. It helps make the magazine more useful for all readers. We don't have the programming staff here to do it automatically, but with thousands of programmers out there reading the magazine, I'm sure some of you must be translating.

Until next time... ©

Keep Those Cards and Letters Coming

COMPUTE! Needs You!

Address articles, programming notes and comments to:

The Editor
COMPUTE!
P.O. Box 5406
Greensboro, NC 27403

Writing For COMPUTE!

Robert Lock, Editor/Publisher

We are always seeking good material for publication in **COMPUTE!**. I cannot overstress our interest in material for the beginner; in short (e.g. 1 page or so) programming hints; in material that crosses "machine boundaries". We present a mix of long articles and short ones. Length is not a criteria of success. Frequently our most favored articles have been simple, provocative programs.

Remember The Beginner

Every time an issue of **COMPUTE!** goes out, there are new readers, with new machines, trying to get started with documentation that may or may not meet their needs. That's one of the reasons we stress good solid introductory material. Many of our readers are interested in simple programming assistance and support. Many are interested in useful programs that allow them to get more practical use from their machines.

Guidelines for Potential Authors

Take a look at The Readers Feedback column this time. It's devoted to reader comments on content. Then sit down and write up a brief article describing that program you've been using at home for six months that you think nobody else would be interested in. You might be surprised.

Submitting Articles To COMPUTE!

Manuscripts should be double spaced, typed with both upper and lower case (please!). Program listings should be provided in printer output form as well as machine readable form. If you don't have a printer, that shouldn't stop you from submitting an article. I'm sure your local store or a friend would be more than happy to let you run off a listing for **COMPUTE!** If that isn't feasible, send it anyway. Many excellent articles don't even contain programs.

Address your articles to:

The Editor
COMPUTE! Magazine
P.O. Box 5406
Greensboro, NC 27403 USA

The Follow-Up

We pay for accepted articles based on their number of pages in the magazine. You'll receive payment after the article appears. Thanks to you all for writing for **COMPUTE!** ©

Guest Commentary

BUSINESS APPLICATIONS ANALYSIS--THE MISSING STEP

Editor's Note: This article, originally printed in the July/August, 1980 COMPUTE!, is reprinted because of its usefulness.

Hal Wadleigh

Business applications analysis seems to be the most neglected element of the microcomputer industry today. The shame of it is that the principles of business analysis affect almost every phase of the use of microcomputers in the business environment--from the initial choice of equipment to evaluating programs in use. The root of the problem appears to lie in the history of microcomputer software.

A short time ago, there was little or no business software available for the smaller microcomputer systems. The software market was flooded with games, but programs that do anything useful for businesses were few and far between. When business programs could be found, they were unfortunately lacking in the qualities that make "good" software distinct from "bad" software. Now that the systems have been out for a while, the quantity of business packages available is greatly improved. The bad news is that the quality of this software (with a few notable exceptions) is as poor as ever.

Both of these situations--the plethora of games and the low quality of business software--seem to be related to the way in which most microcomputer programs are developed. The programmer gets an idea and sits down to start coding. This approach is ideal for games because any interesting oddities that occur during this rather non-objective procedure can be incorporated into the game to make it more interesting. This is also the worst procedure possible for business programming.

The nature of games is that they don't have to do anything in particular (except hold the player's interest) and the job itself can be redefined to accommodate any discoveries made during the programming process. In this case, the program is more important than the job it is supposed to do!

Business programs, however, are the exact opposite--the job is everything and elegant programming is almost meaningless. A good business program is one that does the job well. A bad business program is one that does the job poorly. The elegance and sophistication of the program does not matter. Successful games are usually programs that continually surprise and amaze the user. Business programs had better NOT surprise and amaze the user.

The principles of business applications analysis are really quite simple. It does not take a great deal of intelligence or education--just a little control. It is a five step process:

STEP #1: Define The Job

It is not too unusual to hear a small businessman say something like "I bought one of those little computers last year. What do you think I ought to do with it." It's a rather amazing statement when you think about it. The man has a tool and would like to know what kind of job to do with it. The proper procedure is to buy the tool that best fits the job that needs to be done--it doesn't matter if we're talking about hammers or computers.

Any computer is a tool for processing information. Defining the job for a computer is usually a simple matter of completing the sentence "I want to get. . ." with a detailed description of what will be the output of the system.

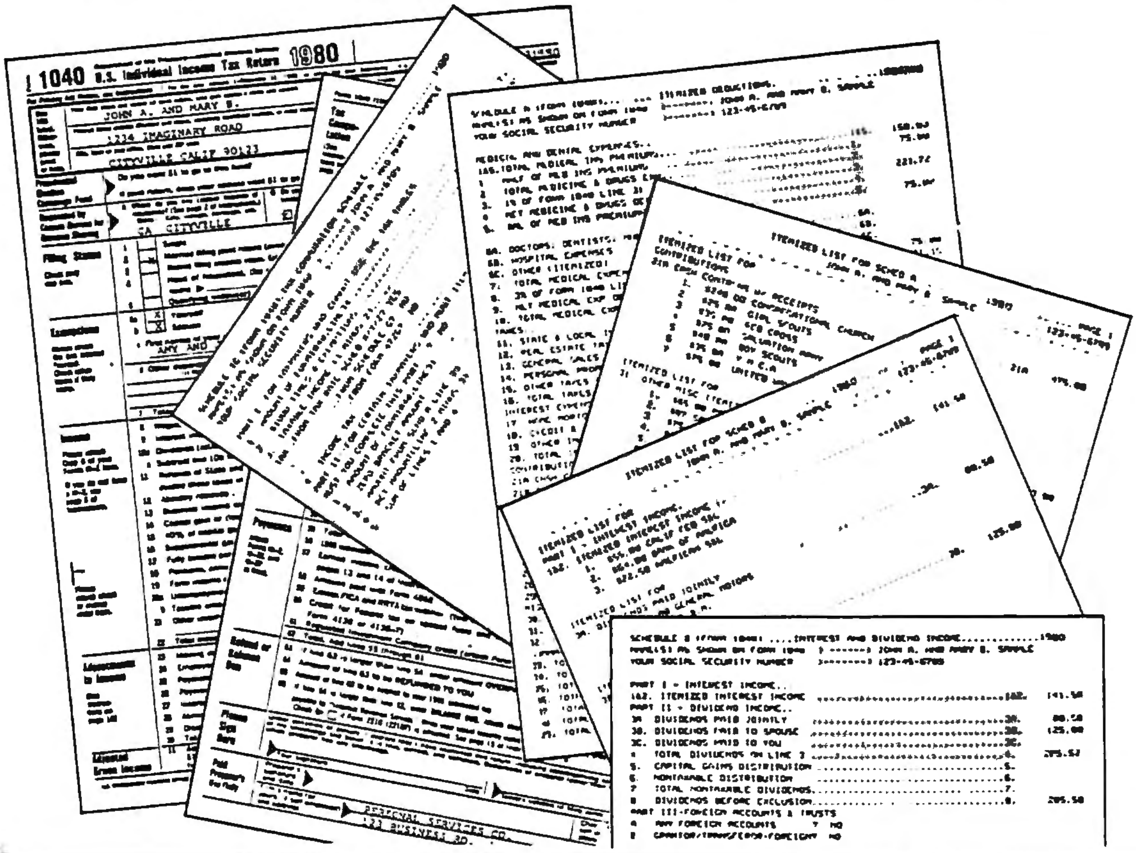
This step is often called the *OUTPUT SPECIFICATION* phase.

STEP #2: Define The Information Necessary To Do The Job

No computer will create new information. A computer will, however, change the form of information that is available to it into a more useful form. For example, a file available to the computer might have a lot of records on items in a business' inventory. Each one of these items has the information on what the value of the items are individually and a count of how many of these items are in stock. The computer can, whenever necessary, take these individual items of information and produce that information in the form of a statement that, "Current inventory is worth $9875.42" on a display. This is not really a matter of producing new information--since the information is already contained in the individual inventory items. The computer has simply changed the form of that information into something more desirable.

Since we have already defined the job we want the computer to do, we now have to define the information that the computer will need to do that job. This often involves a bit of research. The person who does this part of the anlysis has to know how to do the job itself. It also usually involves

THIS YEAR . . . DO IT THE EASY WAY!



. . . for the SERIOUS Apple II User

1981 TAX PREPARER by Howardsoft™

(COMPUTER-AIDED TAX FILING & RECORD-KEEPING)

EASY-TO-USE — *Self-instructing with built-in sample return • Prompted by questions & menus • On-screen preparation uses facsimile of IRS forms • Easily created itemized lists with sums computed & entered on proper lines • Full-feature editing of all data • Error-tolerant disk filing for retrieval & updating.*

TIME-SAVING — *All math & linking of forms performed automatically • Changes in one form automatically reflected in all others • Reenter preparation at any point • Once data entered & editing complete, computer produces complete printout of forms, schedules, itemized lists • File forms directly with the IRS.*

CONVENIENT — *Year-long record-keeping makes year-end tax filing easy • Interactive & on-line, no prior organizing necessary • Use with or without line printer • No special printer or paper • Creates own facsimile of IRS forms • Paginated with IRS-type header • Fully compatible with all line printers.*

COMPLETE — *All schedules (A,B,C,D,E,F,G,R&RP,SE,TC) and many forms (1040,2210,2106,3468,4562,4726,4797,5695) • Low-cost annual updates to stay current with new tax laws • Designed for future expansion to include additional forms • Full documentation and instructions in handsome notebook*

SOPHISTICATED — *Alternative tax strategies easily compared • Special handling of gain/loss, depreciation, rental income • All files coded for use by multiple taxpayers • Used by tax professionals for numerous clients • Extra features when used with 2 disk drives • All tax rate schedules are built in.*

AVAILABLE NOW for Apple Computers with 48K, Applesoft ROM, & 1 or 2 disk drives . . . $99

(Inquire at your dealer or order direct. We pay shipping on prepaid orders. All others sent C.O.D. with handling charges added. California residents add 6% sales tax.)

Also ask about our Real Estate Analyzer & Creative Financing packages.

HOWARD SOFTWARE SERVICES

7722 Hosford Avenue
Los Angeles, CA 90045
(213) 645-4069

finding out the exact form the information is in when it becomes available to the people who will be operating the computer.

This step is often called the *INPUT SPECIFICATION* phase.

STEP #3: Define The Information To Be Stored

Some of the information necessary to do the job will be needed over and over again. It is silly and wasteful to require operators to enter this information every time it is needed. Sometimes the job itself is simple data retrieval--looking at stored information. This is the step where the information that should be stored is defined. In this step, you decide the number of data files, the form of each data record in the file, and even the size of the file.

This step is often called the *FILE SPECIFICATION* phase.

STEP #4: Determine The Physical Flow Of The Information

Business applications are a matter of getting the right information to the right place at the right time. If the computer is going to be printing reports in the accounting office and the information is needed at the loading dock, then the system specifications have to include a means of getting that printed report to the loading dock. This step will be almost meaningless in some applications--but it will be the most critical step in others. In either case, it cannot be ignored--even if it seems to be unimportant at first glance.

This step is often called the *WORKFLOW SPECIFICATION* phase.

STEP #5: Define The Time Contraints Of The Operation

Since we are dealing with a system that has to get the right information to the right place at the right time, we need to make some rather exact definitions of the tolerable delays for each step of the job. It would be silly to define a system that has to sort large files in many different ways without allowing enough time for these sorting operations. It would also be silly to try to function without such sorting operations if they are critical to the operation itself. This final step is often called the *RESPONSE TIME SPECIFICATION PHASE.*

This constraints defined in this stage may show that the previous steps have resulted in a system design that simply cannot work fast enough to do the job. This could necessitate doing one or more of the earlier steps over until all five steps conclude with a acceptable applications design.

The Final Result--System Specifications

Now that you have completed these five steps, you have some idea of what you are looking for. You still haven't chosen any equipment and you haven't even designed any programs--but you DO have a complete definition of the exact job to be done--and that is the most critical point:

YOU CANNOT BUY AND PROGRAM A COMPUTER TO DO A JOB UNLESS YOU KNOW EXACTLY WHAT THE JOB IS!!!!

Unless you have gone through this process, you don't really know what the job is and you can't really make any informed decisions about equipment or programming. The end results are all too often either comical or tragic.

The general impression of many computer professionals is that micro systems are toys and that micro software is limited to games and junk. There is an uncomfortable amount of truth to that view--due to the haphazard way in which micros have been used. If people in the microcomputer industry begin using their tools properly, that attitude will change. It will soon become obvious that mainframe systems are needlessly expensive behemoths and that mainframe software is archaic and oversensitive to small errors.

The *real* microcomputer revolution will begin when microcomputers are used properly--and defining the job to be done is *always* the first step to proper use. ©

LED
A Line-Oriented Text Editor

Arnie Lee,
ABACUS SOFTWARE

A compiler, unlike the BASIC interpreter in your personal computer converts program source statements written in an English-like language, into a format acceptable for execution by your computer. This article deals not with compilers, but with a general purpose utility that is used to create and maintain the program source language statements that are input to compilers.

While designing the TINY PASCAL System for the PET and APPLE II it became apparent very early in the development stage that we would need a utility program to maintain the PASCAL source language statements. The utility we wrote for this purpose is called the LINE EDITOR (LED).

The LED is line-oriented as opposed to word-oriented. As such, it cannot be considered a true word processor although it does provide many of the same capabilities as many of the other commercially available word processors. In fact, a slightly modified version of the LED was used to create this article. Although we wrote the LED to maintain program source statements, its usefulness is by no means limited to that application.

The LED is a line oriented text editor. The entire source program must be in memory while the user is modifying it. Modifications allowed include appending source to the end of the text, inserting lines of text into the middle of existing text, changing occurance of a character string to another string, and printing the text to a hardcopy device. After creating or modifying the text, the user may then save it onto tape or diskette. Some of the key points to note when using the LED are:

- each line is numbered
- each line can contain up to 80 characters
- when entering a line, the line must be terminated by RETURN key
- a maximum of 500 lines of text may be entered (this is subject to the memory capacity of your particular computer)
- as lines are inserted or deleted from the source program, the remaining lines are automatically renumbered
- a line of source may extend more than one screen line on your crt

Commands

The following are the descriptions of each of the commands:

'F' enter FILER portion of LED
This command allows you to use the LOAD or SAVE commands which are described below:

'L' load file from tape or diskette
This command allows you to load a previously edited source program. The source program may have been saved on tape or diskette.
After keying 'L' the LED will prompt you for the name of the source program. Key in the filename and depress RETURN. Do not key in the suffix '.SOURCE'. If you decide that you really don't want to load a file, then enter a null line instead of a filename. At this point you will be asked if the file was saved on tape or diskette (for the PET version of LED). Type 'T' or 'D' as appropriate. If the source program is on tape, then you must put the source file tape into cassette #1. For either tape or diskette, the filename that is keyed in must match the filename that is on the storage medium.

'S' save file onto tape or diskette
This command allows you to save the current source program onto tape or diskette.
After keying 'S', the LED will prompt you for the name of the file to be saved. Key in the filename and then depress RETURN. The filename is limited to twelve characters. The suffix '.SOURCE' will be added to the filename by the LED. If you decide that you really don't want to save a file, then enter a null line instead of a filename.
At this point you will be asked if you want to save the file onto tape or diskette (for PET version of the LED). Type 'T' or 'D' as appropriate. If the source is to be saved onto tape, then you must put the tape into cassette #1.

***Note that tape is supported only in the PET version.

'A' append the end of source
This command allows you to add lines to the end of the current source program. If you have not loaded any source program, then this command will allow you to create a new source program. You may append one or as many lines as you desire. To signal the end of append mode press RETURN when the cursor is sitting at the first character after the line number prompt (null line).

'C' change string
This command allows you to change an existing string to a new string. It will make changes to either a single line or to a range of lines. Indicate a single line by keying in its line number followed by RETURN. Indicate a range of lines by keying the line number of the first line to be searched followed by '-' and finally followed by the line number of the last line to be searched followed by RETURN. You will then be prompted for the change string. The format for the changed string is:

+ from-string + to-string +

where:
+ is a delimiter—any character may be used but it must not be contained in either the from-string or the to-string.
from-string is the string of characters which are to be replaced
to-string is the string of characters which are to replace the from-string in the original source line
e.g. /abc/xyz/
in the above example all occurances of 'abc' will be replaced by 'xyz'.
e.g. /abc//
in the above example all occurances of 'abc' will be eliminated (replaced by nulls).

'D' delete line or range
This command allows you to delete a line or a range of lines from the source program in memory.
DELETE range(low,high)-> 80
will delete line 80
DELETE range(low,high)-> 80-90
will delete lines 80 thru 90
DELETE range(low-high)-> -20
will delete all lines thru 20

'I' insert lines into source program
This command will allow you to insert lines into the existing source program. LED will prompt you for the line number before which you want to insert the new source statements. You may enter one or as many new lines as you desire. Follow each line with RETURN. To signal the end of INSERT mode press RETURN when the cursor is setting at the first character in the line (null).

'L' list source program
This command allows you to list a line or range of lines.
LIST range(low-high)-> 80
will list line 80
LIST range(low-high)-> 80-100
will list lines 80 thru 100
LIST range(low-high)-> -20
will list all lines thru 20
LIST range(low-high)-> null
will list all lines

With the LIST command only the following features are available:

PET
RUN/STOP key - suspends the listing awaiting the depression of the RETURN key.
SPACE BAR - scrolls the listing one line at a time
OFF/RVS key - slows the speeds of the listing

APPLE II
ESC key - suspends the listing awaiting the depression of the RETURN key.
RETURN KEY - reverts to normal speed listing after ESC
SPACE BAR - slows the speed of the listing

'M' menu display
This command allows you to see a more complete explanation of the commands than the abbreviated version which prompts you.

'P' print source program
This command allows you to print a line or range of lines to a hardcopy printer. The PET is supported thru the IEEE interface as device 4. The APPLE II is supported thru slot 2. The range specifications are identical as LIST.

'Q' quit LED
This command allows you to gracefully exit from the LED. The LED gives you a chance to change your mind so that if you accidentally keyed 'Q', then you will have another opportunity to save your source file.

'R' replace a line
This command allows you to replace a single line only. After keying in the line number to be replaced, the LED will prompt you with that line number. Key in the replacement text and press return.
e.g.
REPLACE -line#-> 108
allows you to replace line 108

Listing

The listing which follows is the version of the LED for the PET/CBM machines. The version for the APPLE II is very similar to the PET/CBM version and runs under APPLESOFT. The major differences between the two versions are in the routines that handle the disk and printer I/O.

```
0 REM LINE EDITOR (C)1980 ABACUS ¬
    ¬SOFTWARE
10 DIMT$(500):REM BUFFER SPACE
20 L$="":REM CURRENT LINE
30 LL=1:REM LAST LINE #
40 SP$=" ":DL$=CHR$(20)
50 EE=0:REM DISK ERROR CHANNEL CLOSED
60 PR=0:REM PRINT CHANNEL
90 POKE144,49:REM DISABLE STOP KEY
100 PRINT"ħ        ɪABACUS SOFTWARE LINE ¬
    ¬EDITOR"
110 PRINT"↓↓      FUNCTIONS:"
130 PRINT
140 PRINTTAB(8);"A)PPEND-TO END OF TEXT"
150 PRINTTAB(8);"C)HANGE-STRING
160 PRINTTAB(8);"D)ELETE LINE(S)
170 PRINTTAB(8);"F)ILER COMMANDS
180 PRINTTAB(8);"I)NSERT BEFORE LINE
190 PRINTTAB(8);"L)IST LINE(S)
200 PRINTTAB(8);"M)ENU DISPLAY
210 PRINTTAB(8);"P)RINT LINE(S)
220 PRINTTAB(8);"Q)UIT LEAVE EDITOR
230 PRINTTAB(8);"R)EPLACE LINE
240 PRINT:PRINT"    ENTER SELECTION-> ";
250 GOTO510
500 PRINT:PRINT"ɪENTERr̂ A,C,D,F,I,L,P,Q,
    ¬R,M)ENU->";
510 GET A$:IFA$=""THEN510
520 J=0:FORI=1TO10
530 IFA$=MID$("ACDFILRMQP",I,1)THENJ=I:
    ¬I=10
```

```
540 NEXTI
550 PRINTA$
560 IFJ=0THEN500
570 ONJGOTO1000,2000,3000,4000,5000,
     ¬6000,7000,100,8000,9000
1000 PRINT
1005 PRINT"£APPENDȓ TO END OF TEXT"
1010 PRINT:PRINTLL">";
1020 GOSUB10000:REM GO READ LINE
1030 IFLEN(L$)=0THEN500
1040 T$(LL)=L$
1050 LL=LL+1
1060 GOTO1010
2000 REM CHANGE STRING
2010 PRINT:PRINT"£CHANGEȓ";:GOSUB16000:
     ¬REM GET RANGE
2020 IFHI=0THEN500
2025 PRINT"£CHANGEȓ STRING->";:GOSUB1000
     ¬0:REM GET STRING
2030 L=LEN(L$)
2040 IFL=0THEN500
2050 IFL<4THEN2000
2060 DM$=LEFT$(L$,1):REM DELIMITER
2070 IFRIGHT$(L$,1)<>DM$THEN2000
2080 J=0:FORI=2TOL-1
2090 IFMID$(L$,I,1)=DM$THENJ=I
2100 NEXTI
2110 IFJ=0THEN2000
2120 IFJ=2THEN2000
2130 FR$=MID$(L$,2,J-2)
2140 IFJ+1=LTHENTS$="":GOTO2160
2150 TS$=MID$(L$,J+1,L-J-1)
2160 F=LEN(FR$)
2170 FORI=LOTOHI
2180 T=LEN(T$(I)):S=1:NL$=""
2190 FORJ=1TOT-F+1
2200 IFMID$(T$(I),J,F)<>FR$THEN2230
2210 NL$=NL$+MID$(T$(I),S,J-S)+TS$
2220 S=J+F:J=S-1
2230 NEXTJ
2240 IFS<>1THENNL$=NL$+RIGHT$(T$(I),
     ¬T-S+1):T$(I)=NL$
2250 NEXTI
2260 GOTO500
3000 REM DELETE LINE(S)
3005 PRINT:PRINT"£DELETEȓ ";:GOSUB16000:
     ¬REM GET RANGE
3010 IFNOTDFTHEN3015:REM NOT DEFAULT ON ¬
     ¬ENTIRE FILE
3011 PRINT"£DELETEȓ ENTIRE FILE? ";
3012 GETA$:IFA$=""THEN3012
3013 PRINTA$:IFA$="N"THEN500
3014 IFA$<>"Y"THEN3011
3015 IFHI>LL-1THEN500
3020 IFHI=LL-1THENLL=LO:GOTO500
3030 J=HI-LO+1
3040 FORI=LOTOLL-J-1
3050 T$(I)=T$(I+J)
3060 NEXTI
3070 LL=LL-(HI-LO)-1
3080 GOTO500
4000 REM FILER
4010 PRINT"ψ£FILERȓ ENTER L)OAD OR ¬
     ¬S)AVE-> ";
4020 GETA$:IFA$=""THEN4020
4030 IFA$<>"L"ANDA$<>"S"THENPRINT:
     ¬GOTO4000
4040 PRINTA$:M$=A$
4050 PRINT"£ENTERȓ FILENAME-> ";
4070 GOSUB10000
4075 IFLEN(L$)=0THEN500
4076 IFLEN(L$)>12THEN4050
4080 FI$=L$
4090 PRINT"£ENTERȓ D)ISK OR T)APE-> ";
4100 GETA$:IFA$=""THEN4100
4110 PRINTA$
4120 IFA$<>"D"ANDA$<>"T"THEN4090
4130 IFA$="D"THEN4160:REM DISK ROUTINES
4140 IFM$="L"THEN4400
4150 GOTO4200
4160 DR$="":IFLEFT$(FI$,2)<>"0:"ANDLEFT$
     ¬(FI$,2)<>"1:"THENDR$="0:"
4170 GOTO4600
4200 REM TAPE SAVE
4210 IFLL=1THENPRINT"NO FILE TO SAVE":
     ¬GOTO500
4220 OPEN2,1,2,FI$+".SOURCE"
4230 FORI=1TOLL-1
4240 FORJ=1TOLEN(T$(I))
4250 PRINT#2,MID$(T$(I),J,1);
4260 NEXTJ
4270 PRINT#2,CHR$(255);
4280 NEXTI
4290 CLOSE2
4300 PRINTSPC(6);FI$;" SAVED"
4310 GOTO500
4400 REM TAPE LOAD
4410 OPEN2,1,0,FI$+".SOURCE"
4430 LL=0:REM LINE COUNT
4440 LL=LL+1:T$(LL)=""
4450 GET#2,A$
4460 IFST=64THEN4500:REM END OF FILE
4465 IFST<>0THENPRINT"*** LOAD ERROR ¬
     ¬***":GOTO500
4470 IFA$=CHR$(255)THEN4440:REM END OF ¬
     ¬LINE
4480 T$(LL)=T$(LL)+A$
4490 GOTO4450
4500 CLOSE2
4510 PRINTSPC(6);FI$;" LOADED"
4520 LL=LL+1
4530 GOTO500
4600 REM DISK SAVE
4610 IFM$="L"THEN4800
4620 IFLL=1THENPRINT"NO FILE TO SAVE":
     ¬GOTO500
4630 FL$="@0"+DR$+FI$+".SOURCE,S,W"
4640 OPEN2,8,2,FL$
4650 GOSUB20000:REM ERROR CHECK
4655 IFE1<>0THEN500
4660 FORI=1TOLL-1
4670 FORJ=1TOLEN(T$(I))
4680 PRINT#2,MID$(T$(I),J,1);
4690 NEXTJ
4700 PRINT#2,CHR$(255);
4710 NEXTI
4720 CLOSE2
4730 PRINTSPC(6);FI$;" SAVED"
4740 GOTO500
4800 REM DISK LOAD
4810 FL$=DR$+FI$+".SOURCE,S,R"
4820 OPEN2,8,2,FL$
4830 GOSUB20000:REM ERROR CHECK
4835 IFE1<>0THEN500
4840 LL=0:REM LINE COUNT
4850 LL=LL+1:T$(LL)=""
4860 GET#2,A$
4870 IFST=64THEN4500:REM END OF FILE
4880 IFST<>0THENGOSUB20000:GOTO500
4890 IFA$=CHR$(255)THEN4850:REM END OF ¬
     ¬LINE
```

```
4900 T$(LL)=T$(LL)+A$
4910 GOTO4860
4920 CLOSE2
4930 PRINTSPC(6);FI$;" LOADED"
4940 LL=LL+1
4950 GOTO500
5000 REM INSERT LINE
5010 PRINT:PRINT"ɼINSERTr̂ BEFORE ";:
     ¬GOSUB17000:REM GET LINE#
5015 IFLO>LLORLO<1THEN5000
5020 PRINT:PRINTLO;">";
5030 GOSUB10000:REM READ LINE
5040 IFLEN(L$)=0THEN500
5050 LL=LL+1
5060 FORI=LLTOLOSTEP-1
5070 T$(I)=T$(I-1)
5080 NEXTI
5090 T$(LO)=L$
5100 LO=LO+1
5110 GOTO5020
6000 REM LIST LINES
6010 PRINT:PRINT"ɼLISTr̂ ";:GOSUB16000:
     ¬REM GET RANGE
6020 IFHI=0THEN500
6030 SS$="N":PRINT:FORI=LOTOHI:REM ¬
     ¬PERFORM LIST
6040 PRINTI;">";T$(I)
6050 GETA$:IFA$=CHR$(18)THENFORJ=1TO1024
     ¬:NEXTJ
6060 IFA$<>CHR$(3)THEN6110
6070 SS$="Y"
6080 GETA$:IFA$=CHR$(13)THENSS$="N":
     ¬GOTO6110
6090 IFA$<>CHR$(32)THEN6070
6100 GOTO6120
6110 IFSS$="Y"THEN6070
6120 NEXTI
6130 GOTO500
7000 REM REPLACE LINE
7010 PRINT:PRINT"ɼREPLACEr̂ ";:GOSUB17000
     ¬:REM GET LINE#
7020 IFLO>=LLORLO<1THEN7000
7030 PRINT:PRINTLO;">";
7040 GOSUB10000:REM READ LINE
7050 IFLEN(L$)=0THEN500
7060 T$(LO)=L$
7070 GOTO500
8000 REM QUIT
8010 PRINT:PRINT"     ɼLEAVE EDITOR-ARE ¬
     ¬YOU SURE?r̂ ";
8020 GETA$:IFA$=""THEN8020
8030 PRINTA$
8040 IFA$<>"Y"ANDA$<>"N"THEN8000
8050 IFA$="N"THEN500
8060 PRINT:PRINT"        ɼ** END LINE ¬
     ¬EDITOR **r̂"
8070 POKE144,46:REM ENABLE STOP KEY
8080 END
9000 REM PRINT LINE
9010 IFPR=0THENPR=4:OPENPR,PR
9020 PRINT"ɼPRINTr̂ ";:GOSUB16000:
     ¬REM GET RANGE
9030 IFHI=0THEN500
9040 FORI=LOTOHI:REM PERFORM PRINT
9050 PRINT#PR,I;": ";T$(I)
9060 NEXTI
9070 PRINT#PR
9080 PRINT#PR,"***";LL-1;"LINES IN ¬
     ¬BUFFER ***"
9090 PRINT#PR
9100 GOTO500
10000 REM INPUT A LINE OF TEXT
10010 L$=""
10020 PRINT"$<";
10030 GETA$:IFA$=""THEN10030
10040 IFA$=CHR$(13)THENPRINT" ":RETURN
10050 IFLEN(L$)>80THENGOTO15000
10060 IFA$>=SP$ANDA$<=CHR$(95)THEN10100
10065 IFA$>=CHR$(161)ANDA$<=CHR$(223)THE
      ¬N10100
10070 IFA$<>DL$THENGOTO10030
10080 IFLEN(L$)>0THENPRINTA$;:L$=LEFT$(L
      ¬$,LEN(L$)-1)
10090 GOTO10020
10100 L$=L$+A$:PRINTA$;:GOTO10020
15000 REM LINE INPUT ERROR
15010 PRINT:PRINT"ɼERRORr̂ LINE TRUNCATED
      ¬"
15020 RETURN
16000 PRINT"RANGE(LOW,HIGH)-> ";
16010 GOSUB10000:REM INPUT RANGE
16020 LO=1:HI=LL-1:REM DEFAULT LIST ALL
16025 L=LEN(L$)
16030 DF=0:IFL=0THENDF=-1:GOTO16150
16040 J=0:FORI=1TOL
16050 A$=MID$(L$,I,1)
16060 IFA$>="0"ANDA$<="9"THEN16090
16070 IFA$="-"THENJ=I:GOTO16090
16080 J=99:I=99
16090 NEXTI
16100 IFJ=99THEN16000
16110 IFJ=0THENLO=VAL(L$):HI=LO:RETURN
16120 IFJ>1THENLO=VAL(LEFT$(L$,J-1))
16130 IFJ<LTHENHI=VAL(RIGHT$(L$,L-J))
16140 IFLO>HITHEN16000
16150 RETURN
17000 PRINT"-LINE#->";
17010 GOSUB10000:REM INPUT LINE#
17020 L=LEN(L$)
17030 IFL=0THEN17000
17040 J=0
17050 FORI=1TOL
17060 A$=MID$(L$,I,1)
17070 IFA$>="0"ANDA$<="9"THEN17090
17080 J=99:I=L
17090 NEXTI
17100 IFJ=99THEN17000
17110 LO=VAL(L$)
17120 RETURN
20000 IFEE=0THENEE=15:OPENEE,8,EE
20010 INPUT#EE,E1,E2$,E3,E4
20020 IFE1=0THENRETURN
20030 PRINTE1;",";E2$;",";E3;",";E4
20040 PRINT"*** DISK ERROR ***"
20050 CLOSE2
20060 RETURN
```

©

The Atari 825 Printer

Robert W. Baker
Atco, NJ

The Atari 825 printer is a dot matrix impact printer that can print lines up to eight inches long in three different character sets. The printer operates under complete control of an Atari 400 or Atari 800 Personal Computer System or other compatible host device. The Atari 825 printer is in fact a Centronics 737 printer, well known for its quality and durability. The same mechanism is also used by Radio Shack for their TRS-80 system, so obtaining supplies locally should not be a problem.

When used with the Atari computer, the Atari 850 Interface Module is required for operation of the printer. The Interface Module converts the Atari Input/Output protocol serial data into 7-bit parallel data for operation of the printer. Additionally, the Interface Module also provides four RS232C serial ports for connection of an Atari 830 Modem and other RS232C compatible peripheral devices.

As mentioned earlier, the printer will print in any of three character sets:

1) **Monospaced** (uniformly spaced) characters at 10 characters per inch (cpi).

2) **Monospaced condensed** characters at 16.7 cpi

3) **Proportionally spaced** characters at an average of 14 cpi

Each of these character sets consists of 96 standard ASCII characters. The default character set is the 10 cpi set and cannot be mixed with characters in the other two character sets. The condensed and proportionally spaced characters must be programmed by means of control codes and can be mixed on the same print line. When a character set selection code has been transmitted to the printer, the printer prints all characters in that set until it receives a different character set selection code or the printer is powered off.

The monospaced characters are formed in a dot matrix 7 dots wide by 8 dots high. The spacing between characters is uniform: 3 dot spaces between normal monospaced characters, and 2 dot spaces between condensed monospaced characters. Including the dot spaces between characters, the normal monospaced characters are considered to be 10 dot spaces wide and the condensed monospaced characters 9 dot spaces wide. The proportionally spaced characters are formed in a dot matrix N dots wide by 9 dots high, where N is a variable number of dots from 6 to 18. The numeric characters in this character set do not vary in width. They're always 12 dots wide and are monospaced at 12.5 cpi to allow tabulating columns of numbers. With the increased dot density in the porportionally spaced character set, print quality is extremely good and comes close to that of "letter quality" printers.

Elongated characters can be programmed by control codes and all characters can be elongated. The elogated characters have twice the dot width of normal size characters and can be mixed with normal size characters on the same print line. Elongated printing terminates when the "stop elongated printing" control code is programmed or the print line is terminated.

The printer can print up to 80 ten character per inch monospaced characters on an eight inch print line. When the characters are elongated, it can print half as many characters per line. The maximum number of proportionally spaced characters per eight inch line varies with the dot width of the characters. Therefore, the line limit must be computed in dot columns with a limit of 1200 per eighth inch line.

By decreasing or increasing the number of dot spaces between characters and/or words in the print line, you can even justify lines at the right margin. The printer responds to six dot spacing control codes that set from 1 to 6 dot spaces between words or characters. This feature combined with the proportionally spaced character set should make the printer ideal for word processing applications.

One special control code recognized by the printer activates a ninth pin in the print head to provide automatic underlining. Both characters and spaces are underlined until a "stop underlining" control code is encountered or the printer is powered off.

Backspacing is initiated by a control consisting of the ASCII backspace code (BS) followed by a character specifying the number of dot spaces to be backspaced. This feature is especially useful for printing overstrike characters such as a slashed-zero or a not-equal sign. The printer does not actually backspace in the sense that the carriage moves backward. The carriage can only move from left to right. When the printer receives a BS code, it returns the carriage to the left margin with no paper motion (no line feed) and then moves the carriage out to the last print column counted minus the specified backspace dot spaces. Then it prints the next character and continues the line. Remember that the number of dot spaces to specify in order to backspace to the desired print position depends on the character set being used.

The printer generates an automatic line feed each time the print line is terminated and the carriage is returned to the left margin. In addition, the printer responds to four different line feed codes to advance one full line, reverse one full line, reverse a half line, and advance a half line. This allows prin-

DYNACOMP

Quality software for:*

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

GAMES, SIMULATIONS and EDUCATION

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.

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 and GAMES PACK II — Price: $ 9.95 each, Cassette, $13.95 each, Diskette

GAMES PACK I contains BLACKJACK, LUNAR LANDER, CRAPS, HORSERACE, SWITCH and more. GAMES PACK II includes CRAZY EIGHTS, JOTTO, ACEY-DUCEY, LIFE, WUMPUS and others. Available for all computers!

Why pay $5.95 or more per program when you can buy a DYNACOMP collection for just $9.95?

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.

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 accompany 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.

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 micro computers. 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 hold under pressure. Practice with POKER PARTY before going to that expensive game tonite!

Availability

DYNACOMP software is supplied with complete documentation containing clear explanations and examples. 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

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

BUSINESS, UTILITIES and MISCELLANEOUS

MAIL LIST II (North Star only) — Price: $21.95

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; over 2200 with double density systems)!

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 indenting. 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!

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.

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 ANALZYER may be pruchased 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

REGRESSION I and II may be purchased together for $36.95 (cassettes) and $44.95 (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.

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

VISA

DYNACOMP, Inc.
6 Rippingale Road
Pittsford, New York 14534
(716) 586-7579

master charge

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

A T A R I   8 2 5   P R I N T E R   P R I N T   S A M P L E S

xxxxxxxxxx -- Standard character set, 10 cpi
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
ELONGATED CHARACTERS

xxxxxxxxxx -- Condensed character set, 16.7 cpi
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
ELONGATED CHARACTERS

xxxxxxxxxx -- Proportionally spaced character set
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890
ELONGATED CHARACTERS

Back to normal printing with automatic underlining illustration.

The formula for water is H_2O - not - H^2O xxxxx <-- super & sub scripts!

PROPORTIONAL *** CONDENSED xxx PROPORTIONAL MIXED ON SAME LINE

X ≠ Y *** <--- SHOWS USE OF THE BACKSPACE TO CREATE SPECIAL CHARACTERS

ting special mathematics and chemical formulas requiring subscripts or superscripts. The auto-linefeed function can be disabled if desired but is not recommended when used with the Atari computer systems.

The printer has a special line feed buffer that can store up to 255 line feed codes. This allows you to program consecutive line feed codes for paper handling and page formatting purposes. You can also mix line feed codes with character data, since the printer handles line feed codes and print characters on a "first-in-first-out" basis. Any characters before a line feed code are printed before performing a line feed, then any characters after the line feed are printed after the line feed code is executed. A carriage return does not occur with a line feed.

Manual switches on the front of the printer provide power on/off, online/local modes, and manual paper feeding in forward or reverse. The printer can handle three types of paper:

- **roll paper,** 8.5 inches wide
- **fanfold paper,** 9.5 inches wide with pin feed holes
- **cut sheets and forms** up to 8.5 inches wide

An adjustment on the printing head allows uniform printing of multi-part forms as well as single sheets.

A special ribbon is required for ribbon replacement. The ribbon is not on a typical ribbon spool. Instead, the ribbon used is referred to as a "zip pack". The continuous ribbon loop is provided in a plastic bag which is removed after the ribbon is installed in the printer. A pair of plastic gloves are usually supplied with each ribbon to avoid getting your hands covered with ink.

Several BASIC commands provide easy access to the prnter on the Atari computer systems. The LIST"P:" command will list a BASIC program on the printer rather than on the tv screen. The LPRINT command is used to print any data instead of displaying on the screen. Use of the LPRINT command does not require an OPEN statement and can be used in either direct (typed from the keyboard) or deferred mode (within a program). An OPEN command allows opening a "logical file" to the printer and then using the PRINT#... command to print to the printer. More details on these commands are provided in the Atari manuals.

All data to be sent to the printer is normally enclosed in quotes. ASCII control codes (like linefeed -LF) are generated on the Atari keyboard by pressing the CTRL key and holding it while pressing the next character key. Escape code sequences (like ESC SOH) are generated by pressing and releasing the ESC key (as many times as needed) and then keying CTRL followed by the desired character. When control codes and escape code sequences are keyed on the Atari keyboard, Atari graphics characters are displayed on the TV screen. These graphics

characters are not printed by the printer, but the printer responds to the control codes generated.

If your BASIC program includes graphics characters that generate printer control codes, the printer will act on the codes when you try to list the program on the printer. This can be avoided to some extent by using the CHR$(nn) string function to include control codes in a print line. Most control codes will require two CHR$(nn) functions, such as: CHR$(27);CHR$(14) to start enlongated printing. Note that the CHR$ argument (nn) is a decimal number representing the value to be sent to the printer.

All in all, the printer is a very good quality, dependable unit manufactured by a highly respected printer manufacturer. Supplies and repairs should be easily obtainable from a number of local sources. The printer includes a number of features that make it ideally suited for most word processing and small business applications. It would certainly appear to be a very worth while investment for the serious Atari user. By the way, the price of the Atari 825 is very close to the price of a normal Centronics 737 so you don't pay a penalty by buying it from an Atari dealer. However, remember that you do need the Atari 850 Interface module to use the printer with the Atari system.

TABLE 1. PRINTER CONTROL CODES

Keying Sequence	ASCII Mnemonic	Decimal Code	Hex. Code	Function
This group is keyed by pressing CNTR and this letter:				
J	LF	10	0A	Forward line feed
M	CR	13	0D	Carriage return
O	SI	15	0F	Start underline
N	SO	14	0E	Stop underline
H	BS	08	08	Backspace (must be followed by character defining number of dot spaces)
This group is keyed by ESC ESC then CTRL and this letter:				
A	ESC SOH	27 01	1B 01	1 dot space
B	ESC STX	27 02	1B 02	2 dot spaces
C	ESC ETX	27 03	1B 03	3 dot spaces
D	ESC EOT	27 04	1B 04	4 dot spaces
E	ESC ENQ	27 05	1B 05	5 dot spaces
F	ESC ACK	27 06	1B 06	6 dot spaces
J	ESC LF	27 10	1B 0A	Full reverse line feed
N	ESC SO	27 14	1B 0E	Start elongated print
O	ESC SI	27 15	1B 0F	Stop elongated print
Q	ESC DC1	27 17	1B 11	Select proportionally spaced character set
S	ESC DC3	27 19	1B 13	Select 10 cpi mono-spaced character set
T	ESC DC4	27 20	1B 14	Select 16.7 cpi condensed characters
This group by ESC ESC ESC then CTRL and this letter:				
↑	ESC FS	27 28	1B 1C	Half-line forward line feed
←	ESC RS	27 30	1B 1E	Half-line reverse line feed

©

Simulated PRINT USING

Jim Butterfield

It's handy to be able to arrange numbers neatly in columns. Computers having the PRINT USING statement help you do this. If your machine hasn't got a PRINT USING, however, you'll need to do it some other way.

There are many methods of producing this kind of output. One of the better ones involves extracting the digits, one at a time, and then printing them; this method is a little slow in Basic because of the arithmetic involved.

I've put together a quick and fairly fast subroutine to help you do the job. The actual coding is eight Basic lines, so it won't take up too much space. To allow for maximum flexibility, you are permitted to name how many digits you want to allow before the decimal point, and how many after.

The subroutine takes your value V and gives you back a string, V$, which you can then print. String V$ contains the leading spaces and trailing zeros to fit the space you have specified.

The length of V$ can be worked out this way: You will have specified how many digits you want before the decimal point as variable V1 and after the decimal point as variable V2. Add these two values together; then add one for the sign and one more for the decimal point. Exception: If you've specified V2 as zero, meaning you want no digits after the decimal point, the decimal point itself will be dropped.

Images, Pictures and Patterns

There are many possible features of a PRINT USING system that are *not* included in this short subroutine. You should know about them; perhaps you would like to try your hand at adding some of them.

A **floating dollar sign** allows the dollar sign to move up snugly against the number itself. A **fill** character fills up all the spaces before the first digit of the number; it's most often used with the asterisk character, to give an output that looks like *****12.47 for printing cheques.

Comma insertion allows you to punctuate large numbers, to give an output like 3,827,149. **Negative numbers** often have many ways of display: examples are -437.22, (437.22) and 437.22CR. **Variable Zero Suppression** allows you to choose whether to print a value of five cents as .05, 0.05, or 00.05.

The above features, if included, would make the subroutine bigger and slower. Apart from a floating minus sign, they are not there; but a couple of features have been included which are important for financial printouts.

All numbers are carefully rounded, so that a value of 12.387 will convert to 12.4 if you choose to show one place after the decimal.

Overflow is tested: it would be annoying or disastrous to have a value of 12345 printed as 345 just because you asked for three digits before the decimal! Situations like this are flagged by the printing of asterisks instead of the number.

There's one type of overflow that doesn't cause asterisks to be printed, but in this case you're unlikely to mistake it for a genuine value. Occasionally, when you have a number like one million, the STR$ function will convert it to a string like "1E+09" rather than the "1000000000" we might expect. (Why the extra zeros? They are intended to go behind the decimal point). This causes an odd-looking output of something like "1E. +09" which won't be mistaken for a real number. If this bothers you, you could add extra coding to spot it. It's probably better, however, to think of overflow as a debugging tool — it must never, never happen in your final polished program.

You should try to keep the number of digits (V1 plus V2) not greater than 9. If you really want to print amounts well over a million dollars with accuracy to the penny, you're starting to push against the limits of 32-bit Basic; rounding errors will start to steal the occasinal penny away from you.

The Program

Line 50020 changes V to a rounded integer, and 50030 converts to a string. At this point, 3.14159 becomes the string " 3142" with spaces at the front padding out the string to the right length.

Line 50040 skips decimal point insertion if we don't need it. Otherwise, line 50050 checks to see if there are any spaces behind where we want to place the decimal point. For example, the value .014 might be held as " 14", and we'll need to re-insert the missing zero. Variables V5 and V6 will do this for us, if needed.

Note that line 50050 leaves an "unclosed" loop on the stack. So long as this is a subroutine, it won't give us any problem: the loop will be closed when the subroutine performs RETURN.

Line 50070 puts in the decimal point and any needed zeros. Finally, line 50080 checks for overflow and substitutes asterisks if needed.

The test program, lines 100-170, produces both very large and very small numbers, both positive and negative.

```
100 REM DEMO PROGRAM FOR SUB-
ROUTINE
110 FOR J = 1 to 20
120 V = EXP(RND(1)*14-6)*SGN(RND(1)-.2)
130 V1 = 4:V2 = 0:GOSUB 50000:PRINTV$;" ";
140 V1 = 3:V2 = 1:GOSUB 50000;PRINTV$;" ";
150 V2 = 4:GOSUB 50000:PRINT V$
160 NEXT J
170 END
```

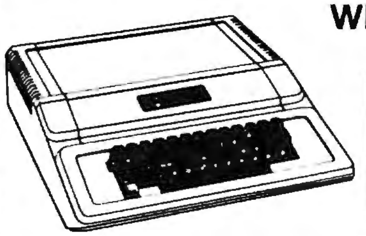
```
50000 REM 'USING' ARRANGE IN COLUMNS
50010 REM V IS VALUE; V1.V2 PRINTS
50020 V4 = INT(V*10↑V2 +.5)
50030 V$ = RIGHT$("          " + STR$(V4),
V1 + V2 + 1)
50040 IF V2<1 GOTO 50080
50050 FOR V5 = V1 + 2 TO V1 + V2 + 1:IF ASC
  (MID$(V$,V5))<48 THEN NEXT V5
50060 V6 = V5-V1-1
50070 V$ = MID$(V$,V6,V1 + 1) + LEFT$(".00000",
  V6) + MID$(V$,V5)
50080 IF ASC(V$)>47 THEN V$ = LEFT$("******
  ******",V1 + V2 + 2 + (V2 = 0))
50090 RETURN
READY.
```

©

THE STAR MODEM
From Livermore Data Systems

RS232 MODEM	SALE	$135
IEEE 488 MODEM	SALE	$245
RS232 CCITT		$170
IEEE 488 CCITT		$280

STAR Modem is the price performance leader with a full 2 YEAR FACTORY WARRANTY



WE CARRY THE BLACK APPLE

BELL & HOWELL
MADE EXCLUSIVELY FOR BELL & HOWELL BY
apple computer inc.

EBS Business System for PET/CBM $795
Extremely comprehensive package for small business. Fully integrated inventory and accounts receivable system including invoices, packing slips, mail labels, statements, bank deposit slips, and 17 reports. Allows any of 10 standard letters to be merged with customer record info on either a selective or complete file basis. Demo disk and system description available for $3.00

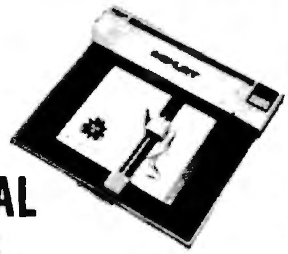
OZZ Data Base System for CBM 8032 $335
Flexible file handling and report writing package for such applications as: Inventory Control; Management Information, Mailing List; Scheduling Medical Record Keeping, Accounting.

KMMM Pascal for PET $75
Subset of standard Pascal with true machine language translator for faster execution 16K with tape or disk

EARL for PET (disk file based) $65
Editor, Assembler, Relocater, Linker to generate relocatable object code

fullFORTH+ for PET/CBM $65
A full-featured FORTH with extensions conforming to Forth Interest Group standards. Includes assembler, string processing capabilities, disk virtual memory multiple dimensioned arrays, floating point and integer processing.

MIPLOT Intelligent Plotter
by Watanabe Instruments (Digiplot)



SPECIAL $1145

- Incorporates all intelligent functions required for producing graphs and drawings including 8 vector and 4 character commands.
- Distance accuracy within 1%, repetition accuracy within .01 inches, programmable step size .004 inches, internal interpolation in .002 steps.
- Solid and broken line types can be specified
- Character generator for letters, numerals and symbols. Characters can be enlarged and rotated in four orientations
- Coordinate axes can be drawn by simply specifying the graduation interval and number of repetitions.
- Self test mode automatically draws complete test pattern.
- Printer mode outputs character data in 16 sizes with 4 orientations
- Connection to any microcomputer using parallel 7-bit ASCII code
- Can use simple cable to parallel port, or special interface to IEEE or other.
- Modular control circuit and mechanical construction
- Uses any hard fiber-tip pen
- Uses 11 x 17 paper.

Includes power supply, I/O connector, 2 pens, 50 sheets of paper, and complete manual.

6502	7.45	10/6.95	50/6.55	100/6.15
6502A	8.40	10/7.95	50/7.35	100/6.90
6520 PIA	5.15	10/4.90	50/4.45	100/4.15
6522 VIA	6.45	10/6.10	50/5.75	100/5.45
6532	7.90	10/7.40	50/7.00	100/6.60
2114-L450		3.45	20/3.35	100/3.25
2114-L200		4.15	20/3.95	100/3.75
2716 EPROM (5 volt)		10.45	5/9.90	10/9.50
TMS 2532 EPROM				29.00
4116-200 as RAM (NEC)				8 for 35.00
S-100 Wire Wrap				2.65

CASSETTES — AGFA PE-611 PREMIUM
High output, low noise, 5 screw housing, labels.

C-10	10/5.65	50/25.00	100/48.00
C-30	10/7.30	50/34.00	100/66.00

All other lengths available. Write for price list.

ATARI 800 $777
All Atari Modules 20% OFF
SPECIAL—purchase ATARI 800, receive extra 8K memory FREE.
EDUCATIONAL PLAN - buy 2 ATARI Computers, receive 1 ATARI 400 FREE!

DISKS
(write for quantity prices)

SCOTCH (3M) 5"	10/2.85	50/2.75	100/2.65
SCOTCH (3M) 8"	10/2.95	50/2.85	100/2.75
Maxell 5" Double Dens.	10/4.25	50/4.10	100/3.95
Maxell 8" Double Dens.	10/4.65	50/4.50	100/4.35
Verbatim 5"	10/2.45	50/2.40	100/2.35
(add .75 for 5" Verbatim plastic storage box)			
Verbatim 8" Dbl. Dens.	10/3.35	50/3.25	100/3.15
BASF 5" soft	10/2.60	20/2.50	100/2.40
BASF 8" soft	10/2.40	20/2.35	100/2.30
Diskette Storage Pages			10 for 3.95
Disk Library Cases	8" - 2.85		5" - 2.15

commodore

CBM-PET SPECIALS

FREE Up to $235 free merchandise with purchase of one of following CBM-PET items!

		FREE
8032 32K - 80 Column CRT	$1795	235
8016 16K - 80 Column CRT	$1495	205
8050 Dual Disk Drive-1,020,000 Bytes	$1795	235
CBM Modem - IEEE Interface	$395	50
CBM Voice Synthesizer	$395	50
8N Full size graphics keyboard	$795	100
16K Full Size Graphics or Business Keyboard	$995	150
32K Full Size Graphics or Business Keyboard	$1295	205
2040 Dual Disk Drive - 343,000 bytes	$1295	205
2022 Tractor Feed Printer	$795	100
C2N External Cassette Deck	$95	12
Used PETs (8, 16, and 32K)		CALL

WRITE FOR SYSTEM PRICES

****EDUCATIONAL DISCOUNTS****
Buy 2 PET Computers, get 1 FREE

CBM Full Size Graphics Keyboard	$ 74
WordPro III - 32K CBM, disk, printer	$170
WordPro IV - 8032, disk, printer	$255
VISICALC for PET, ATARI	$170
BPI General Ledger, A/P, A/R for PET	$270
Programmers Toolkit - PET ROM Utilities	$34.90
PET Spacemaker Switch	$24.90
Dust Cover for PET	$6.90
IEEE-Parallel Printer Interface for PET	$65.00
IEEE-RS232 Printer Interface for PET	$149.00

EPSON MX-80 Printer	$545
STARWRITER Daisy Wheel Printer	$1500
Centronics 737 Printer	$790
NEC Spinwriter - parallel	$2500
XYMEC HI-Q 1000 Intelligent Daisy Wheel	$2150
Leedex Video 100 12" Monitor	$129
ZENITH DATA SYSTEMS	
Zenith Z19 Terminal (factory asm.)	$735
Zenith Z89 with 48K	$2150
SYM-1	$209
SYM BAS-1 BASIC or RAE-1/2 Assembler	$ 85
KTM-2/80 Synertek Video Board	$349
KIM-1 (add $34 for power supply)	$159
Seawell Motherboard - 4K RAM	$195
Seawell 16K Static RAM - KIM, SYM, AIM	$320
KL-4M Four Voice Music Board and Visible Music Monitor (4 Voice) for PET	$59.90

FLEXFILE Database-Report Writer by Michael Riley
Flexible file handler for PET/CBM

MICRO-REVERSI for PET by Michael Riley $9.95
Machine language version—you can't win at Level 5.

PAPER-MATE 60 Command PET Word Processor $29.95
Full-featured version by Michael Riley

A P Products 15% OFF
A P Hobby-Blox 15% OFF

ALL BOOK and SOFTWARE PRICES DISCOUNTED

The 8086 Book (Osborne)	$12.75
Z8000 Assembly Language Programming	$10.60
PET Personal Computer Guide (Osborne)	$12.75
PET and the IEEE-488 Bus (Osborne)	$12.75
6502 Assembly Language (Osborne)	$ 9.90
Programming the 6502 (Zaks)	$10.45
6502 Applications Book (Zaks)	$10.45
6502 Software Cookbook (Scelbi)	$ 9.45
CP/M Handbook (w/ MP/M) Zaks	$11.85

115 E. Stump Road
Montgomeryville, PA 18936 215-699-5826 **A B Computers**

WRITE FOR CATALOG.
Add $1.25 per order for shipping. We pay balance of UPS surface charges on all [illegible] Prices listed are on cash discount basis. Regular prices slightly higher

Part 2 of several

The Mysterious And Unpredictable RND

Bob Albrecht and
George Firedrake

Integer RND Numbers

It's true. RND numbers are greater than zero and less than one.

0<RND(1)<1

Another way to say it: RND numbers are decimal fractions *between* 0 and 1.

But what if we want random *integers* from 1 to 6 (as in rolling a die) or random digits (0,1,2,3,4,5,6,7,8,9) or random integers from 1 to 100?

Well, if RND(1) is a number between 0 and 1, then 10 times RND(1) must be a number between 0 and 10. OK? Hmmm ... not so sure? Try this program.

```
100 REM****RND NUMBERS BETWEEN ZERO AND TEN

110 PRINT "[CLR]" ;
120 INPUT "HOW MANY RND NUMBERS" ; N
130 PRINT

200 REM****PRINT N RND NUMBERS (10*RND(1))
210 FOR K = 1 TO N
220   PRINT 10*RND(1),
230 NEXT K
240 PRINT

999 END
```

Here is a sample run.

```
HOW MANY RND NUMBERS? 16

 3.34464508          .328904955
 3.69228884          6.31052523
 7.74506208          4.35766491
 5.07949568          9.26821156
 9.78314249          1.95072511
 8.04495845          7.15665136
 .0878570662         1.47048625
 7.02014795          2.49452329

READY
■
```

Yes, all 16 numbers are between 0 and 10. In the above sample the smallest number is .0878570662 and the largest number is 9.78314249.

Now, think of each number as having an integer part to the left of the decimal point and a fractional part to the right of the decimal point.

6.731052513

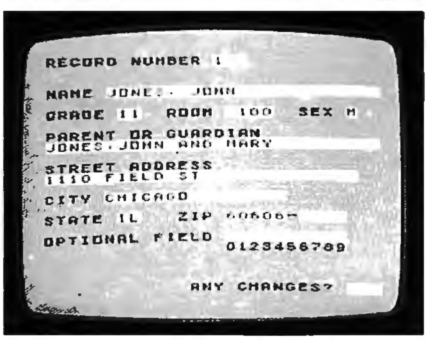
integer part: 6 — fractional part: .731052513

Moving with You into the '80s

Microcomputer Courseware From Scott, Foresman, the Education Expert

Reading Skills Courseware Series (Grades K-6) and **Mathematics Courseware Series** (Grades K-8) feature:

- content correlated with basal series—to supplement instruction, provide enrichment
- individualizing capabilities that meet each pupil's learning needs
- motivating animation, color graphics, music, and speech
- comprehensive Teacher's Guides



School Management Applications for administrators feature:

- fourteen modules, each programmed to streamline a specific area of school administrative work
- easy information storage and access, clearly organized printed reports
- program instructions in plain English, helpful Reference Manuals

Scott, Foresman Module Packages (Grades 1-3) for Speak & Read™ Educational Model, complete with Pupil Books, Teacher's Guides, and optional Workbooks, can enhance any reading curriculum by reinforcing essential skills.

Scott, Foresman and Company
Electronic Publishing
Glenview, Illinois 60025

Learn more. Mail this coupon to receive free information. CO2

To Scott, Foresman Electronic Publishing

Please send me free, full-color brochures on:

- ☐ Reading Skills Courseware Series and Mathematics Courseware Series (E0103)
- ☐ School Management Applications (E0101)
- ☐ Scott, Foresman Module Packages for Speak & Read™ (E0141)

School ____________________

Name ____________________

Position ____________________

School Address ____________________

City __________ State __________ ZIP Code __________

Here is a program to print random digits. Each number printed will be a single digit, 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9. The random digit is computed and printed in line 220.

```
1ØØ REM****RANDOM DIGITS: Ø 1 2 3 4 5 6 7 8 9

11Ø PRINT "[CLR]" ;
12Ø INPUT "HOW MANY RANDOM DIGITS" ; N
13Ø PRINT

2ØØ REM****PRINT N RANDOM DIGITS
21Ø FOR K = 1 TO N
22Ø   PRINT INT(1Ø*RND(1)) ,
23Ø NEXT K
24Ø PRINT

999 END
```

A sample run might look like this.

```
HOW MANY RANDOM DIGITS? 2Ø

3         6         5         4
5         5         7         6
6         Ø         8         1
2         8         Ø         3
5         8         9         5

READY.
■
```

Do you understand how the program works? The key is line 220.

It goes like this.

- RND(1) is a number between 0 and 1.
- 10*RND(1) is a number between 0 and 10.
- The integer part of 10*RND(1) is a single digit, 0 through 9.
- Aha! INT(10(RND(1)) is a single digit, 0 through 9.

Exercise 4.

(a) What is the integer part of 7.15665136? ________
    What is the fractional part? ________

(b) What is the integer part of 5.07949568? ________
    What is the fractional part? ________

(c) Beware! This one is tricky (but you can do it).
    What is the integer part of .328904955? ________
    What is the fractional part? ________

For each RND number between 0 and 10, the integer (whole number) part is a single digit. So, let's tell the PET to keep the integer part and get rid of the fractional part.

Here's how. We will use the INT function. In case you are not already familiar with the INT function, here are some examples.

INT(6.30152513) = 6
INT(7.15665136) = 7
INT(5.07949568) = 5
INT(.328904955) = 0
INT(1.95072511) = 1

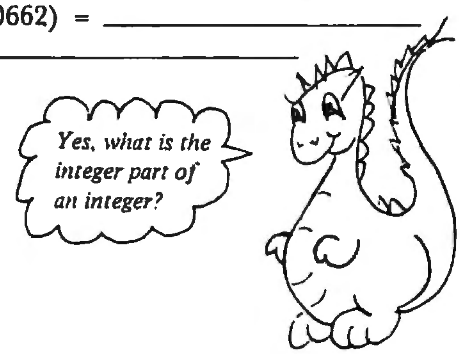
For positive numbers, the INT function gives the integer part of the number and throws away the fractional part.

Exercise 5. Complete the following.

(a) INT(2.49452329) = ________

(b) INT(.0878570662) = ________

(c) INT(7) = ________



What happens if you ask the PET to compute the INT of a negative number? Try it and find out.

Exercise 6. Show how to rewrite line 220 to get *integers* in each range shown.

(a) 0 or 1 220 ________

(b) 0, 1, 2, 3, 4, or 5 220 ________

(c) 0 to 99, inclusive 220 ________

Hmmm . . . suppose we want to simulate (imitate) rolling dice. We need integer RND numbers from 1 to 6, inclusive (1 or 2 or 3 or 4 or 5 or 6).

220 PRINT INT(6*RND(1)) + 1,

Let's see now, how does that work?

INT(6*RND(1)) is an integer, 0 to 5.
INT(6*RND(1)) + 1 is an integer, 1 to 6.

In our program on page 9, change line 220 as shown above. Also, change line 120, as follows.

120 INPUT "HOW MANY DIE ROLLS" ; N

Go ahead — try it. Here's what happened when we did it.

```
HOW MANY DIE ROLLS? 2Ø

4         2         2         6
1         1         2         1
3         1         5         3
2         1         6         2
1         6         4         5

READY.
■
```

Exercise 7. Show how to rewrite line 220 to get integers in each range shown.

(a) 1 or 2 220 ________

(b) 1,2,3,4,5,6,7,or 8 220 ________

(c) 1 to 100, inclusive 220 ________

The next two are tricky!

(d) 2 or 3 220 ________

(e) 2, 3 or 4 220 ________

Now try these. What numbers might be printed by each PRINT statement

(f) 220 PRINT INT (4*RND(1)) + 5, ________

(g) 220 PRINT 2*(3*RND(1)) + 1, ________

Coin Flipper

Let's ask the PET to flip a coin for us. Well, actually let's ask the PET to simulate (imitate) flipping a coin. Here is one program to simulate flipping a coin.

```
1ØØ REM****COIN FLIPPER #1

2ØØ REM****FIND OUT HOW MANY FLIPS
21Ø PRINT "[CLR]" ;
22Ø INPUT "HOW MANY COIN FLIPS" ; N
23Ø PRINT

4ØØ REM****FLIP COIN N TIMES, PRINT EACH EVENT
41Ø FOR K = 1 TO N
42Ø   COIN = INT(2*RND(1))◄C will be either 0 or 1
43Ø   IF COIN = Ø THEN PRINT "TAILS",
44Ø   IF COIN = 1 THEN PRINT "HEADS",
45Ø NEXT K
46Ø PRINT
999 END
```

Let's try it.

```
HOW MANY COIN FLIPS? 1Ø

HEADS     HEADS     TAILS     HEADS
HEADS     TAILS     HEADS     TAILS
TAILS     HEADS

READY
■
```

We got six HEADS and four TAILS. RUN the program several times, using various numbers of flips. Count the HEADS and TAILS each time.

When we flip a coin, we expect that HEADS and TAILS are equally probable. That is, we are as likely to get HEADS as TAILS. We also expect that, if we flip a coin many times, the number of HEADS and the number of TAILS will be about the same.

Let's modify our program so that the PET counts the HEADS and TAILS. In the following program, we have added lines 300 through 320, changed lines 430 and 440, and added lines 500 and 510.

```
1ØØ REM****COIN FLIPPER #2
2ØØ REM****FIND OUT HOW MANY FLIPS
21Ø PRINT "[CLR]" ;
22Ø INPUT "HOW MANY COIN FLIPS" ; N
23Ø PRINT
3ØØ REM****T = TAILS COUNTER H = HEADS
    COUNTER
31Ø T = Ø
32Ø H = Ø
4ØØ REM****FLIP COIN N TIMES, COUNT TAILS
    AND HEADS
41Ø FOR K = 1 TO N
42Ø   COIN = INT(2*RND(1))
43Ø   IF COIN = Ø THEN PRINT 'TAILS',
      : T = T + 1
44Ø   IF COIN = 1 THEN PRINT 'HEADS",
      : H = H + 1
45Ø NEXT K
5ØØ REM****PRINT RESULTS OF N FLIPS
51Ø PRINT "I GOT" H "HEADS AND" T "TAILS."
999 END
```

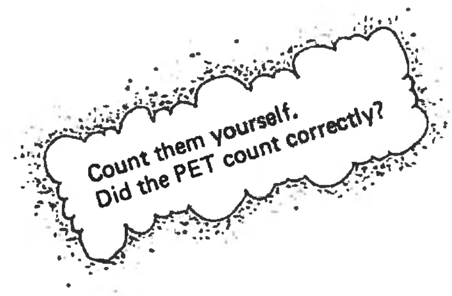
Now a RUN will show the actual "flips" on the screen, followed by the number of HEADS and the number of TAILS.

```
HOW MANY COIN FLIPS? 24

TAILS     HEADS     TAILS     TAILS
TAILS     HEADS     HEADS     TAILS
HEADS     HEADS     HEADS     TAILS
HEADS     TAILS     TAILS     HEADS
TAILS     HEADS     HEADS     TAILS
HEADS     HEADS     TAILS     HEADS

I GOT 13 HEADS AND 11 TAILS

READY.
■
```



How does the program work? In line 420, COIN will be either 0 or 1. If COIN is 0, then line 430 will cause TAILS to be printed and the value of T to be increased by 1.

430 IF COIN = 0 THEN PRINT "TAILS",:T = T + 1

If COIN is 0, all of this is done. If COIN is *not* 0, none of this is done.

If COIN is 1, then line 440 will cause HEADS to be printed and the value of H will be increased by 1.

440 IF COIN = 1 THEN PRINT "HEADS", : H = H + 1

If COIN is 1, all of this is done. If COIN is *not* 1, none of this is done.

This program is OK for small samples. However, if you ask the PET for a larger sample (for example, 1000 flips) then alas, only part of the sample will be on the screen along with the number of HEADS and the number of TAILS.

So, instead of printing HEADS or TAILS on the screen, let's tell the PET to "flip" a COIN N times and count (but don't print) the number of HEADS and the number of TAILS.

Exercise 8. Complete the following program to flip a coin N times and count the HEADS and TAILS.

```
1ØØ REM****COIN FLIPPER #3

2ØØ REM****FIND OUT HOW MANY FLIPS
21Ø PRINT "[CLR]" ;
22Ø PRINT
23Ø INPUT "HOW MANY COIN FLIPS" ; N

3ØØ REM****T = TAILS COUNTER, H = HEADS COUNTER
31Ø T = Ø
32Ø H = Ø
```

```
4ØØ REM****CLIP COIN N TIMES, COUNT TAILS &
    HEADS
41Ø FOR K = 1 TO N
42Ø   COIN = INT(2*RND(1))
43Ø   IF COIN = Ø THEN ______________
44Ø   IF COIN = 1 THEN ______________
45Ø NEXT K

5ØØ REM****PRINT RESULTS OF N FLIPS
51Ø PRINT "I GOT" H "HEADS AND" T "TAILS."
52Ø GOTO 22Ø
```

↖ Go back for another bunch of flips.

A RUN might look like this.

```
HOW MANY COIN FLIPS? 1ØØ
I GOT 53 HEADS AND 47 TAILS.

HOW MANY COIN FLIPS? 1ØØ
I GOT 45 HEADS AND 55 TAILS.

HOW MANY COIN FLIPS? 1ØØØ
I GOT 5Ø6 HEADS AND 494 TAILS.

HOW MANY COIN FLIPS?     and so on . . .
```

Remember. With this program, the PET is actually simulating the coin flips, but is not printing the result of each flip. Instead, it counts the number of HEADS and the number of TAILS and, after doing the required number of flips, prints the results.

Exercise 9. Write a program to simulate flipping *two* coins. For a single toss, there are four possible outcomes.

HH HT TH TT

We show HT and TH as different outcomes, because —
suppose we toss a nickel and a dime. The possible outcomes are like this:

NICKEL	**H**	**H**	**T**	**T**
DIME	**H**	**T**	**H**	**T**
	HH	**HT**	**TH**	**TT**

Here is a RUN of our program to flip two coins at a time.

```
HOW MANY COIN FLIPS? 2Ø

TH        TH        HH        TT
HH        TH        HT        HT
TH        TH        TT        TH
TT        TT        TT        HH
HH        HT        TH        HH

READY
■
```

Exercise 10. Instead of printing the results (HH or HT or TH or TT), count them. Write a program to flip two coins N times, then print the number of times they came up HH, HT, TH and TT. Below is a sample RUN, showing how we would like to see the results.

```
HOW MANY COIN FLIPS? 1ØØØ

OUTCOME    NUMBER OF TIMES
  HH         243
  HT         25Ø
  TH         259
  TT         248

READY
■
```

©

Keep Those Cards and Letters Coming

COMPUTE! Needs You!

Address articles, programming notes and comments to:
The Editor
COMPUTE!
P.O. Box 5406
Greensboro, NC 27403

Stat Lab

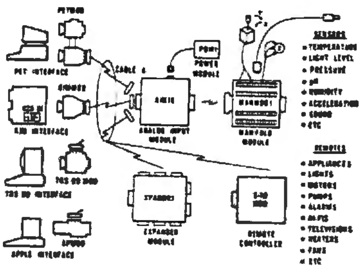
A. Wachtel

The 2^k Experimental design is a method to determine the effect of a number of parameters which influence the outcome of a process, as well as their interactions. The main effects are considered independent from each other and orthogonal. For example, we wish to determine the effects of temperature, pressure, and agitation on the yield of a chemical reaction, or we wish to determine the number of sales per month as a function of product quality, packaging, and the amount of advertising. Combining each of the three parameters in all possible ways, we get 2^3 or 8 figures for yield or sales which are entered as DATA. The program employs Yate's algorithm which is simply a convenient mathematical method to arrive at the results which are read from the table which is produced. Suppose A = temperature (A = high, - = low), B = pressure (B = high, - = low), and C = agitation (C = fast, - = slow or absent), the EFFECT = denote the effects of each of these conditions on yeild. Since $EFFECT^2/2^K$ = mean square, this is essentially a k - way ANOVA. An estimate of the error usually obtains from the sum of mean squares of the interactions (normally low, i.e. noise). If then, we wish to determine the confidence level for some main effect, we divide its mean square by that of the error to arrive at an F value. Replication of the experiment, i.e. obtaining two inputs for each condition is much better, because then we can obtain an independent estimate of the error from the differences between the replicates.

```
 0 GOTO 410
 10 REM 2^K EXPERIMENTAL DESIGN   A. WAC
HTEL  PITTSBURGH, PA 15235.
 20 DIM X(32),Y(32),Z(32)
 30 N=0
 40 READ Y:IF Y=9999 THEN 80
 50 N=N+1:I=N
 60 Y(I)=Y:X(I)=Y(I)
 70 GOTO 40
 80 DEF FNA(X)=INT(X*1000+0.5)/1000
 90 K=INT(LOG(N)/LOG(2)+0.5)
 100 FOR J=1 TO K
 110 FOR I=1 TO N/2
 120 Z(I)=X(2*I)+X(2*I-1)
 130 NEXT I
 140 FOR I=N/2+1 TO N
 150 Z(I)=X(2*(I-N/2))-X(2*(I-N/2)-1)
 160 NEXT I
 170 FOR I= 1 TO N
 180 X(I)=Z(I)
 190 PRINT""
 200 NEXT I:NEXT J
 210 PRINT" N";TAB(6)"Y";TAB(10)"VARIABL
ES";TAB(27)"ESTIMATES"
 220 PRINT" —";TAB(5)"———";TAB(10)"——
————";TAB(27)"—————————"
 230 PRINT
 240 FOR N=0 TO 2^K-1
 250 J=N
 260 IF J=0 THEN A$="  MEAN=":D=2^K
 270 IF J<>0 THEN A$="EFFECT=":D=2^K/2
 280 FOR I=K-1 TO 0 STEP -1
 290 K(K-I)=INT(N/2^I):N=N-K(K-I)*2^I
 300 NEXT I
 310 PRINT J+1;TAB(4)Y(J+1);TAB(10);
 320 FOR I=K TO 1 STEP -1
 330 IF K(I)=0 THEN B$=" -"
 340 IF K(I)=1 THEN B$=" "+CHR$(K-I+65)
 350 PRINTB$;
 360 NEXT I
 370 PRINT TAB(21)A$;TAB(30)FNA(X(J+1)/D
)
 380 N=J
 390 NEXT N
 400 GOTO 540
 410 PRINT""
 420 PRINT"THIS PROGRAM FINDS THE MAIN A
ND INTER-"
 430 PRINT"ACTION EFFECTS OF K VARIABLES
 A,B,..."
 440 PRINT"IN ALL COMBINATIONS BY YATE'S
 ALGORITHM"
 450 PRINT"                  USE LINE 1
AND"
 460 PRINT"ANY LINES UP TO 19 TO ENTER N
 DATA,"
 470 PRINT"FOLLOWED BY 9999. N IS ALWAYS
 2^K."
 480 PRINT"(16 DATA (K=4) WILL FIT ON TH
E SCREEN)"
 490 PRINT
 500 PRINT"THE DATA CORRESPOND TO THE OB
SERVATIONS"
 510 PRINT"OBTAINED WITH THE VARIABLES H
IGH (OR"
 520 PRINT"PRESENT)='A,B,..'OR LOW (OR A
BSENT)='-'."
 530 PRINT"TO REGAIN INSTRUCTIONS, TYPE
RUN 410."
 540 END
READY.
```

©

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

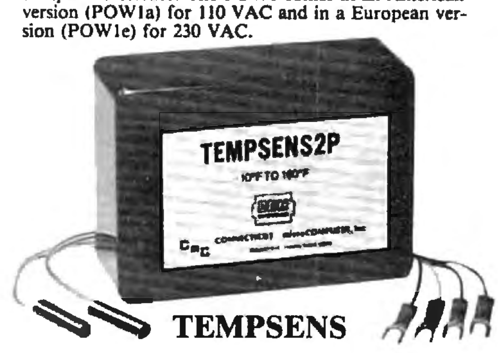
The AIM 16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM 16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the uMAC SYSTEMS special interfaces.

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

Printer And Communication Interfaces For The CBM/PET

SADI For Serial Two-way Communication, Parallel Printers, and Serial Printers

SADI - The microprocessor based serial and parallel interface for the Commodore PET. SADI allows you to connect your PET to parallel and serial printers, CRT's, modems, acoustic couplers, hard copy terminals and other computers. The serial and parallel ports are independent allowing the PET to communicate with both peripheral devices simultaneously or one at a time. In addition, the RS-232 device can communicate with the parallel device.

Special Features for the PET interface include:
- Conversion to true ASCII both in and out
- Cursor controls and function characters specially printed
- Selectable reversal of upper and lower case
- Addressable - works with other devices

Special Features for the serial interface include:
- Baud rate selectable from 75 to 9600
- Half or full duplex
- 32 character buffer
- X-ON, X-OFF automatically sent
- Selectable carriage return delay

Special Features for the parallel interface include:
- Data strobe - either polarity
- Device ready - either polarity
- Centronics compatible

Complete with power supply, PET IEEE cable, RS-232 connector, parallel port connector and case. Assembled and tested.
SADIa (110VAC) $295
SADIe (230VAC) $325

ADA1600 • For Parallel NEC and Centronics Standard Printers

The ADA1600 is a low cost easy to use interface for the Commodore Computers. It allows the PET and CBM computers to use standard Centronics type printers (including the NEC 5530) for improved quality printing. The ADA1600 has a two foot cable which plugs into the PET IEEE port. Another IEEE card edge connector is provided for connecting disks and other peripherals to the PET. The ADA1600 is addressable and does not tie up the bus. The address is switch selectable. A four foot cable with a standard 36 pin Centronics connector is provided. A switch selects upper/lower case, upper/lower case reversed (needed for some Commodore machines) and upper case only for clearer program listings. Works with WORDPRO, BASIC and other software. No special programming is required. The case measures 3 1/2 x 5 3/4 inches. Comes complete, assembled and tested, with case and cables. Power is obtained from the printer or an external power supply may be used. Retail price for the ADA1600 is $129.

ADA1450 • Serial Printer Adapters

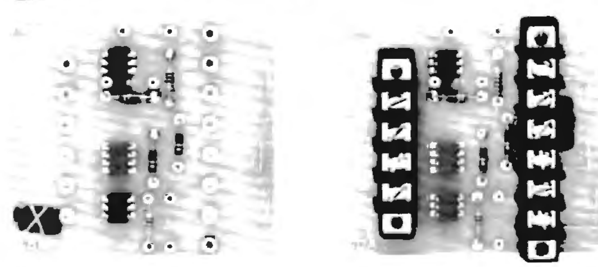
The ADA1450 is a low cost, easy to use serial interface for the Commodore Computers. It allows the PET and CBM computers to use standard serial printers for improved quality printing. The ADA1450 has a two foot cable which plugs into the PET IEEE port. Another IEEE card edge connector is provided for connecting disks and other peripherals to the PET. The ADA1450 is addressable and does not tie up the bus. The address is switch selectable. A six foot RS-232 cable is provided with a DB25 connector. Pin 3 is data out. Pins 5,6 and 8 act as ready lines to the printer. Pins 4 and 20 act as ready lines from the printer. These lines can be switched for non-standard printers. Baud rate is selectable to 9600 baud. A switch selects upper/lower case, upper/lower case reversed (needed for some Commodore machines) and upper case only for clearer program listings. Works with WORDPRO, BASIC and other software. No special programming is required. The case measures 3 1/2 x 5 3/4 inches. Comes complete, assembled and tested, with case, cables, power supply and software on cassette for graphing functions, formatting data etc. The ADA1450 has a female DB25 connector at the end of the RS-232 cable for most standard printers. The ADA1450N has a male DB25 at the end of the RS-232 cable for the DIABLO serial printers. Retail price for the ADA1450 or 1450N is $139.

ADA730 Parallel • For the Centronics 730 and 737 Printers

The ADA730 is a low cost easy to use interface for the Commodore Computers. It allows the PET and CBM computers to use Centronics type 730 and 737 printers. The ADA730 has a two foot cable which plugs into the PET IEEE port. Another IEEE card edge connector is provided for connecting disks and other peripherals to the PET. The ADA730 is addressable and does not tie up the bus. The address is switch selectable. A cable with a 36 pin card edge connector is provided. A switch selects upper/lower coase, upper/lower case reversed (needed for some Commodore machines) and upper case only for clearer program listings. Works with WORDPRO, BASIC and other software. No special programming is required. The case measures 3 1/2 x 5 3/4 inches. Comes complete, assembled and tested, with case and cables. Power is obtained from the printer or an external power supply may be used. Retail price for the ADA is $129.

Word Processor Program •

PET Word Processor. On tape - $39.50, On disk - 49.50
For 8K Pets 29.50
For 16K and 32K Pets 39.50
Compose and print letters, flyers, ads, manuscripts, etc. Uses disk or tape. 30 page manual included.



ADA400 RS-232 To Current Loop Adapter

RS-232 to current loop adapter (ADA 400). $29.50
Two circuits - 1 each direction. Run an RS-232 device off a computer's teletype port or vice versa. Optoisolated.

ADA400S - Solder Pads 24.50
ADA400B - Barrier Strips with screw terminals 29.50

QUANTITY DESCRIPTION PRICE TOTAL

SUBTOTAL
Handling and shipping — add per order $3.00
Foreign orders add 10% for AIR postage
Conn. residents add 7% sales tax
TOTAL ENCLOSED
NAME ____
COMPANY ____
ADDRESS ____
CITY ____
STATE ____ ZIP ____
VISA O M/C O Expiration date ____
Card number ____
Mention this magazine and deduct 2% from the TOTAL

In the US order from: Connecticut MicroComputer, Inc.
34 Del Mar Drive Brookfield, CT 06804 (203) 775-4595
In Canada order from: Batteries Included, LTD
71 McCaul St. F6 Toronto, Canada M5T2X1 (416)596-1405

Connecticut microComputer, Inc.
34 Del Mar Drive, Brookfield, CT 06804
203 775-4595 TWX: 710 456-0052

A BCD to Floating-Point Binary Routine

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

Introduction

The principal purpose of this article is to provide the reader with a program that converts a BCD number (ASCII representation) with a decimal point and/or an exponent to a floating-point binary number. The floating-point binary number has a mantissa of 32 bits, an exponent byte consisting of a sign bit and seven magnitude bits, and a sign flag (one byte) for the mantissa. Positive and negative numbers whose magnitudes vary from $1.70141183*10^{38}$ to $1.46936795*10^{-39}$ and zero can be handled by this routine. In subsequent articles I *hope* to provide an output routine and a four-function arithmetic routine. The routine described here could be used in conjunction with the Am9511 Arithmetic Processing Unit [1] to perform a large variety of arithmetic functions.

Floating-Point Notation

Integer arithmetic is relatively simple to do with the 6502. Consult the Bibliography for a number of sources of information on multiple-byte, signed number addition, subtraction, multiplication and division. Scanlon's book, in particular, has some valuable assembly language routines of this sort. However, additional problems arise when the decimal number has a fractional part, such as the "14159" in the number 3.14159. Also, integer arithmetic is not suitable for handling large numbers like $2.3*10^{15}$. The solution is to convert decimal numbers to floating-point binary numbers. A binary floating-point number consists of a **mantissa** with an **implied binary point** just to the left of the most-significant non-zero bit and an **exponent** (or characteristic) that contains the information about where the binary point must be moved to represent the number correctly. Readers who are familiar with **scientific notation** will understand this quickly. Scanlon's book has a good section on floating-point notation. We will merely illustrate what a decimal number becomes in floating point binary by referring you to Table 1. The dashed line over a sequence of digits means that they repeat. For examples, $1/3 = .\overline{33}$ and $1/11 = .09\overline{090} = .\overline{090}$ while a binary example is $1/1010 = .0001100\overline{1100} = .000\overline{1100}$.

Table 1. Decimal number to floating-point binary conversions.

NUMBER	BINARY NUMBER	FLOATING POINT NOTATION	MANTISSA	EXPONENT
0	0	0×2^0	0	0
1	1	$.1 \times 2^1$	1	1
2	10	$.1 \times 2^2$	1	10
4	100	$.1 \times 2^3$	1	11
1.5	1.1	$.11 \times 2^1$	11	1
0.75	.11	$.11 \times 2^0$	11	0
0.1	$0.0001100\overline{1100}$	$.\overline{1100} \times 2^{-3}$	$\overline{1100}$	-11
31	11111	$.11111 \times 2^5$	11111	101
32	100000	100000	1	110

A close examination of Table 1 yields some important conclusions. Unless a number is an integer power of two (2^n where n is an integer), the mantissa required to correctly represent the number will require more bits as the numbers increase. Thus, the number 1 can be correctly represented with a one-bit mantissa, but the number 31 requires a five-bit mantissa. A n-bit mantissa can correctly represent a number as large as $2^n - 1$, but no larger. There is another problem associated with numbers like 0.1_{ten} that become *repeating* numbers in binary. It should be clear that no mantissa with a *finite* number of bits can represent 0.1 *exactly*. The fact that computers use a finite number of bits to represent numbers like 0.1 can be illustrated by using BASIC to add 0.1 to a sum and print the answer repeatedly. Starting with a sum of zero, we obtained an answer of 3.6 after 36 times through the loop, but the next answer is 3.69999999 which is clearly incorrect. The error incurred by using a finite number of bits, to represent a number that requires more than that number of bits to correctly represent it, is called *roundoff error*.

How many bits should be used for the mantissa? Clearly it should be an integer number of bytes for ease in programming. Some computers have software packages that use a 24 bit mantissa. The largest number that can be represented by 24 bits is $2^{24} - 1 = 16777215$. This represents about seven decimal digits, giving about six digit accuracy after several calculations. With my salary there is no trouble with six digit accuracy, but many financial calculations require accuracy to the nearest cent, and six digits are frequently not enough. If we choose 32 bits for our mantissa size we get a little more than nine digits (4.3×10^9). This is the mantissa size used in several versions of Microsoft BASIC, and it is the size chosen here. The propagation of round-off errors through the calculations normally gives about eight digit accuracy. It is generally true that the roundoff errors accumulate as the number of calculations to find a specific result increases, but this is a subject beyond the scope of this article.

How big should the **exponent** be? If we choose to represent the binary exponent with one byte then we will have seven bits to represent the exponent (one sign bit and seven magnitude bits). The largest

exponent is then +127. If all the bits in the mantissa are ones, then the largest number that can be represented is $(1/2 + 1/4 + 1/8 + 1/16 + ... + 1/2^{32})*2^{127}$, which is approximately $1.70141183*10^{38}$. The smallest exponent is -128. The smallest *positive* number that the mantissa can be is 1/2, thus the smallest positive number that can be represented is 2^{-129} which is approximately $1.46936795*10^{-39}$. Of course, if we chose to use two bytes for the exponent then much larger and smaller exponents could be accommodated, but for most calculations by earth people, a range of 10^{-39} to 10^{38} will do quite nicely. Remember that if you try to enter a number whose absolute value is outside of the range just given (except for zero) you will obtain erroneous results. No overflow or underflow messages are given when entering numbers with this routine.

One more note before turning to the program. The mantissa is said to be *normalized* when it is shifted so that the most-significant bit is one, and the binary point is assumed to be to the left of the most-significant bit. The only exception to this is the number zero which is represented by zeros in both the mantissa and the exponent. Although you are free to assume the binary point is some other place in the mantissa, it is conventional to keep it to the left of the mantissa, as illustrated in Table 1.

The Program To Float A Number

The program in Listing 1, written in the form of a subroutine, together with the other subroutines given in the listings, will accept numbers represented by ASCII from an input device and convert the numbers into their floating point representation. A typical entry might be +12.3456789E+24 or -.123456789E-30. The plus sign is optional since the computer simply disregards it. Up to 12 significant digits may be entered, although the least-significant three will soon be disregarded, leaving approximately 9 decimal digits (32 binary digits). At the completion of the routine, the floating-point representation will be found in locations $0001, $0002, $0003, $0004 (mantissa), $0005 (exponent) and location $0007 contains the sign of the mantissa. The sign byte is $FF if the number is negative, otherwise it is $00. Note that the accumulator (locations $0001-$0004) has not been complemented in the case of a minus number. Forming the twos complement may be done, when required, by the arithmetic routines. If a format compatible with the Am9511 Arithmetic Processing Unit is required, simply drop the least-significant byte of the mantissa ($0004), put the sign (set the bit for a minus, clear it for a plus) in bit seven of the exponent ($0005) and shift the sign of the exponent from bit seven to bit six, making sure to keep the rest of the exponent intact. Table 2 gives a summary of the important memory locations.

Table 2. Memory assignments for the BCD to floating-point binary routine.

$0000 = OVFLO; overflow byte for the accumulator when it is shifted left or multiplied by ten.
$0001 = MSB; most-significant byte of the accumulator.
$0002 = NMSB; next-most-significant byte of the accumulator.
$0003 = NLSB; next-least-significant byte of the accumulator.
$0004 = LSB; least-significant byte of the accumulator.
$0005 = BEXP; contains the binary exponent, bit seven is the sign bit.
$0006 = CHAR; used to store the character input from the keyboard.
$0007 = MFLAG; set to $FF when a minus sign is entered.
$0008 = DPFLAG; decimal point flag, set when decimal point is entered.
$000A = ESIGN; set to $FF when a minus sign is entered for the exponent.
$000B = TEMP; temporary storage location.
$000C = EVAL; value of the decimal exponent entered after the "E."
$0017 = DEXP; current value of the decimal exponent.

After clearing all of the memory locations that will be used by routine, the program in Listing 1 jumps to a subroutine at $0F9B. Most users will not want to call this subroutine, since it merely serves to clear the AIM 65 display. Subroutine INPUT, called next, must be supplied by the user. It must get a BCD digit represented in ASCII code from some input device, store it in CHAR at $0006, *and* return to the calling program with the ASCII character in the 6502's accumulator. The necessary subroutines for the AIM 65 are given in Listing 4. They are given in the "K" disassembly format with no comments since they have previously been described by De Jong[2]. Our subroutines input the number on the keyboard and echo the number on the printer and the display.

The algorithm for the conversion routine was obtained from an article by Hashizume[3]. If you are interested in more details regarding floating-point arithmetic routines, please consult his fine article. A flow chart of the routine in Listing 1 is given in Figure 1. The flow chart and the program comments should be sufficient explanation. Basically it works by converting the number, as it is being entered, to binary and multiplying by ten, in binary of course. Later, if and when the exponent is entered, the number is either multiplied or divided by ten, in binary, to get a normalized mantissa and an exponent representing a power of two rather than a power of ten. Each time a multiplication or division by ten occurs the mantissa is renormalized *and* rounded upward if the most-significant discarded bit is one. Each normalization adjusts the binary exponent. When the decimal exponent finally reaches zero no more multiplications or divisions are necessary since $10^0 = 1$. To maintain 32-bit precision, an extra byte, called OVFLO, is used in the accumulator for all *10 and /10 operations.

REFERENCES

1. De Jong, Marvin L., "Interfacing the Am9511 Arithmetic Processing Unit," **COMPUTE II.** (in press).
2. De Jong, Marvin L., "An AIM 65 Notepad," **MICRO**, No. 16, Sept. 1979, p. 11.
3. Hashizume, Burt, "Floating Point Arithmetic," **BYTE**, V 2, No. 11, Nov. 1977, p. 76.

BIBLIOGRAPHY

1. **Programming and Interfacing the 6502, With Experiments,** Marvin L. De Jong, Howard W. Sams & Co., Indianapolis, 1980.
2. **6502 Assembly Language Programming,** Lance A. Leventhal, Osborne/McGraw-Hill, Berkeley, 1979.
3. **6502 Software Design,** Leo J. Scanlon, Howard W. Sams & Co., Indianapolis, 1980.

Listing 1. ASCII to Floating-Point Binary Conversion Program

```
$0E00 D8        START   CLD          Decimal mode not required
0E01 A2 20              LDX $20      Clear all the memory loca-
0E03 A9 00              LDA $00      tions used for storage by
0E05 95 00      CLEAR   STA MEM,X    this routine by loading
                                     them with zeros.
0E07 CA                 DEX
0E08 10 FB              BPL CLEAR
0E0A 20 9B 0F           JSR CLDISP   Clears AIM 65 display.
0E0D 20 30 0F           JSR INPUT    Get ASCII representation of
0E10 C9 2B              CMP $2B      BCD digit. Is it a + sign?
0E12 F0 06              BEQ PLUS     Yes, get another character.
0E14 C9 2D              CMP $2D      Is it a minus sign?
0E16 D0 05              BNE NTMNS
0E18 C6 07              DEC MFLAG    Yes, set minus flag to $FF.
0E1A 20 30 0F   PLUS    JSR INPUT    Get the next character.
0E1D C9 2E      NTMNS   CMP $2E      Is character a decimal
                                     point?
0E1F D0 08              BNE DIGIT    No. Perhaps it is a digit.
                                     Yes, check flag.
0E21 A5 08              LDA DPFLAG   Was the decimal point flag
                                     set?
0E23 D0 2C              BNE NORMIZ   Time to normalize the
                                     mantissa.
0E25 E6 08              INC DPFLAG   Set decimal point flag,
0E27 D0 F1              BNE PLUS     and get the next character.
0E29 C9 30      DIGIT   CMP $30      Is the character a digit?
0E2B 90 24              BCC NORMIZ   No, then normalize the
                                     mantissa.
0E2D C9 3A              CMP $3A      Digits have ASCII repre-
0E2F B0 20              BCS NORMIZ   sentations between $30
                                     and $39.
0E31 20 00 0D           JSR TENX     It was a digit, so multiply
0E34 A5 06              LDA CHAR     the accumulator by ten and
0E36 38                 SEC          add the new digit. First
0E37 E9 30              SBC $30      strip the ASCII prefix by
                                     subtracting $30.
0E39 18                 CLC          Add the new digit to the
0E3A 65 04              ADC LSB      least- significant byte
                                     of the accumulator.
0E3C 85 04              STA LSB      Next, any "carry" will be
0E3E A2 03              LDX $03      added to the other bytes of
                                     the accumulator.
$0E40 A9 00     ADDIG   LDA $00
0E42 75 00              ADC ACC,X    Add carry here.
0E44 95 00              STA ACC,X    And save result.
0E46 CA                 DEX
0E47 10 F7              BPL ADDIG    The new digit has been
                                     added.
0E49 A5 08              LDA DPFLAG   Check the decimal point
                                     flag.
0E4B F0 CD              BEQ PLUS     If not set, get another
                                     character.
0E4D C6 17              DEC DEXP     If set, decrement the
0E4F 30 C9              BMI PLUS     exponent, then get another
                                     character.
0E51 20 30 0D   NORMIZ  JSR NORM     Normalize the mantissa.
0E54 84 0B              STY TEMP     Save Y. It contained the
0E56 A9 20              LDA $20      number of "left shifts" in
                                     NORM.
0E58 38                 SEC          The binary exponent is 32 -
0E59 E5 0B              SBC TEMP     number of left shifts that
0E5B 85 05              STA BEXP     NORM took to make the
                                     most-significant bit one.
0E5D A5 01              LDA MSB      If the MSB of the accumu-
0E5F F0 5A              BEQ FINISH   lator is zero, then the
0E61 A5 06              LDA CHAR     number is zero, and its all
0E63 C9 45              CMP $45      over. Otherwise, check if
                                     the last character was an
                                     "E".
0E65 D0 52              BNE TENPRW   If not, move to TENPRW.
0E67 20 30 0F           JSR INPUT    If so, get another character.
0E6A C9 2B              CMP $2B      Is it a plus?
0E6C F0 06              BEQ PAST     Yes, then get another
                                     character.
0E6E C9 2D              CMP $2D      Perhaps it was a minus?
0E70 D0 05              BNE NUMB     No, then maybe it was a
                                     number.
0E72 C6 0A              DEC ESIGN    Set exponent sign flag.
0E74 20 30 0F   PAST    JSR INPUT    Get another character.
0E77 C9 30      NUMB    CMP $30      Is it a digit?
0E79 90 3E              BCC TENPRW   No, more to TENPRW.
0E7B C9 3A              CMP $3A
0E7D B0 3A              BCS TENPRW
0E7F 38                 SEC          It was a digit, so strip
                                     ASCII prefix.
$0E80 E9 30             SBC $30      ASCII prefix is $30.
0E82 85 0B              STA TEMP     Keep the first digit here.
0E84 20 30 0F           JSR INPUT    Get another character.
0E87 C9 30              CMP $30      Is it a digit?
0E89 90 13              BCC HERE     No. Then finish handling
0E8B C9 3A              CMP $3A      the exponent.
0E8D B0 0F              BCS HERE
0E8F 38                 SEC          Yes. Decimal exponent is
0E90 E9 30              SBC $30      new digit plus 10 times the
                                     old digit.
0E92 85 0C              STA EVAL     Strip ASCII prefix
                                     from new digit.
0E94 A5 0B              LDA TEMP     Get the old character and
0E96 0A                 ASL A        multiply it by ten. First
                                     times two.
0E97 0A                 ASL A        Times two again makes
                                     times four.
0E98 18                 CLC
0E99 65 0B              ADC TEMP     Added to itself makes times
                                     five.
0E9B 0A                 ASL A        Times two again makes
                                     times ten.
0E9C 85 0B              STA TEMP     Store it.
0E9E 18         HERE    CLC          Add the new digit,
0E9F A5 0B              LDA TEMP
0EA1 65 0C              ADC EVAL     to the exponent.
0EA3 85 0C              STA EVAL     Here is the exponent,
0EA5 A5 0A              LDA ESIGN    except for its sign. Was
                                     it a negative?
0EA7 F0 09              BEQ POSTV    No.
0EA9 A5 0C              LDA EVAL     Yes, then form its twos
0EAB 49 FF              EOR $FF      complement by complemen-
0EAD 38                 SEC          tation followed by adding
                                     one.
0EAE 69 00              ADC $00
0EB0 85 0C              STA EVAL     Result into exponent value
                                     location.
0EB2 18         POSTV   CLC          Prepare to add exponents.
0EB3 A5 0C              LDA EVAL     Get "E" exponent.
0EB5 65 17              ADC DEXP     Add exponent from input
                                     and norm.
0EB7 85 17              STA DEXP     All exponent work finished.
$0EB9 A5 17     TENPRW  LDA DEXP     Get decimal exponent.
0EBB F0 71              BEQ FINISH   If it is zero, routine is
                                     done
0EBD 10 61              BPL MLTPLY   Ir it is plus, go multiply by
                                     ten.
0EBF A2 03      ONCMOR  LDX $03      It's minus. Divide by ten.
0EC1 06 04      BACK    ASL LSB      First shift the accumulator
```

```
0EC3 26 03              ROL NLSB      three bits left.
0EC5 26 02              ROL NMSB
0EC7 26 01              ROL MSB
0EC9 26 00              ROL OVFLO
0ECB C6 05              DEC BEXP      Decrease the binary
0ECD CA                 DEX           exponent for each left shift.
0ECE D0 F1              BNE BACK
0ED0 A0 20              LDY $20       Number of trial divisions
0ED2 06 04     AGAIN    ASL LSB       of $0A into the accumu-
0ED4 26 03              ROL NLSB      lator giving a $20 = 32
                                      bit quotient.
0ED6 26 02              ROL NMSB
0ED8 26 01              ROL MSB
0EDA 26 00              ROL OVFLO
0EDC 88                 DEY
0EDD F0 0E              BEQ OUT       Get out when number of
0EDF A5 00              LDA OVFLO     trial divisions reaches
                                      $20 = 32.
0EE1 38                 SEC           Subtract 10 = $0A from
0EE2 E9 0A              SBC $0A       partial divident in OVFLO.
0EE4 30 EC              BMI AGAIN     If result is minus, zero into
                                      quotient
0EE6 85 00              STA OVFLO     Otherwise store result in
0EE8 E6 04              INC LSB       OVFLO, and set bit to one
                                      in quotient.
0EEA 18                 CLC
0EEB 90 E5              BCC AGAIN     Try it again.
0EED A5 00     OUT      LDA OVFLO     Check once more to see if
0EEF C9 0A              CMP $0A       quotient should be rounded
                                      upwards.
0EF1 90 15              BCC AHEAD     No.
0EF3 A2 04              LDX $04       Yes. Add one to quotient.
$0EF5 B5 00    REPET    LDA ACC,X     Get each byte of the accu-
0EF7 69 00              ADC $00       mulator and add the carry
0EF9 95 00              STA ACC,X     from the previous addition.
0EFB CA                 DEX
0EFC D0 F7              BNE REPET
0EFE 90 08              BCC AHEAD     What if carry from accumu-
0F00 A5 01              LDA MSB       lator occurred? Get most-
0F02 09 80              ORA $80       significant byte and put a 1
                                      in bit seven.
0F04 85 01              STA MSB       Result into high byte,
0F06 E6 05              INC BEXP      and increment the binary
                                      exponent.
0F08 A5 01     AHEAD    LDA MSB       Because of three-bit shift at
0F0A 30 0A              BMI ARND      start of division, a one-bit
0F0C 06 04              ASL LSB       shift (at most) may be re-
0F0E 26 03              ROL NLSB      quired to normalize the
                                      mantissa now.
0F10 26 02              ROL NMSB
0F12 26 01              ROL MSB
0F14 C6 05              DEC BEXP      If so, also decrement binary
                                      exponent.
0F16 A9 00     ARND     LDA $00       Clear overflow byte.
0F18 85 00              STA OVFLO
0F1A E6 17              INC DEXP      For each divide-by-10,
0F1C D0 A1              BNE ONCMOR    increment the decimal ex-
0F1E F0 0E              BEQ FINISH    ponent until it is zero.
                                      Then its all over.
0F20 A9 00     MLTPLY   LDA $00       Clear overflow byte.
0F22 85 00              STA OVFLO
0F24 20 00 0D  STLPLS   JSR TENX      Jump to multiply-by-ten
                                      subroutine.
0F27 20 30 0D           JSR NORM      Then normalize the
                                      mantissa.
0F2A C6 17              DEC DEXP      For each multiply-by-10,
0F2C D0 F6              BNE STLPLS    decrement the decimal ex-
0F2E 60        FINISH   RTS           ponent until it's zero. All
                                      finished now.
```

Listing 2. Multiply by Ten Subroutine.

```
$0D00 18    TENX   CLC          Shift accumulator left.
0D01 A2 04         LDX $04      Accumulator contains
0D03 B5 00  BR1    LDA ACC,X    four bytes so X is set to
                                four.
0D05 2A            ROL A        Shift a byte left.
0D06 95 10         STA ACCB,X   Store it in accumula-
                                tor B.
0D08 CA            DEX
0D09 10 F8         BPL BR1      Back to get another
                                byte.
0D0B A2 04         LDX $04      Now shift accumulator B
0D0D 18            CLC          left once again to get
                                "times four."
0D0E 36 10  BR2    ROL ACCB,X   Shift one byte left.
0D10 CA            DEX
0D11 10 FB         BPL BR2      Back to get another byte.
0D13 A2 04         LDX $04      Add accumulator to
0D15 18            CLC          accumulator B to get
                                A + 4*A = 5*A.
0D16 B5 00  BR3    LDA ACC,X
0D18 75 10         ADC ACCB,X
0D1A 95 00         STA ACC,X    Result into accumulator.
0D1C CA            DEX
0D1D 10 F7         BPL BR3
0D1F A2 04         LDX $04      Finally, shift accumula-
0D21 18            CLC          tor left one bit to get
                                2*5*A = 10*A.
0D22 36 00  BR4    ROL ACC,X
0D24 CA            DEX
0D25 10 FB         BPL BR4      Get another byte.
0D27 60            RTS
```

Listing 3. Normalize the Mantissa Subroutine.

```
$0D 30 18   NORM  CLC
0D 31 A5 00 BR6   LDA OVFLO Any bits set in the over-
0D33 F0 0F        BEQ BR5   flow byte? Yes, then
                            rotate right.
0D35 46 00        LSR OVFLO No, then rotate left.
0D37 66 01        ROR MSB
0D39 66 02        ROR NMSB
0D3B 66 03        ROR NLSB
0D3D 66 04        ROR LSB   For each shift right,
0D3F E6 05        INC BEXP  increment binary
                            exponent.
0D41 B8           CLV       Force a jump back.
0D42 50 Ed        BVC BR6
0D44 90 0D  BR5   BCC BR7   Did the last rotate cause
0D46 A2 04        LDX $04   a carry? Yes, then round
0D48 B5 00  BR8   LDA ACC,X the mantissa upward.
0D4A 69 00        ADC $00   Carry is set so one is
                            added
0D4C 95 00        STA ACC,X
0D4E CA           DEX
0D4F 10 F7        BPL BR8
0D51 30 DE        BMI BR6   Check overflow byte
                            once more.
0D53 A0 00  BR7   LDY $00   Y will count number of
                            left shifts.
0D55 A5 01  BR10  LDA MSB   Does most-significant
0D57 30 0D        BMI BR11  byte have a one in bit
                            seven? Yes, get out.
0D59 18           CLC       No. Then shift the
0D5A A2 04        LDX $04   accumulator left one bit.
0D5C 36 00  BR9   ROL ACC,X
0D5E CA           DEX
0D5F D0 FB        BNE BR9
0D61 C8           INY       Keep track of left shifts.
0D62 C0 20        CPY $20   Not more than $20 = 32
                            bits.
0D64 90 EF        BCC BR10
0D66 60     BR11  RTS       That's it.
```


Listing 4. AIM 65 Input/Output Subroutines.

```
$0F30 20 JSR E93C
0F33 20 JSR F000
0F36 85 STA 06
0F38 20 JSR 0F72
0F3B 20 JSR 0F60
0F3E A5 LDA 06
0F40 60 RTS

$0F85 A2 LDX #12
0F87 BD LDA A438,X
0F8A E8 INX
0F8B 9D STA A438,X
0F8E CA DEX
0F8F CA DEX
0F90 10 BPL 0F87
0F92 A9 LDA #20
0F94 8D STA A438
0F97 20 JST 0F60
0F9A 60 RTS

$0F60 A2 LDX #13
0F62 8A TXA
0F63 48 PHA
0F64 BD LDA A438,X
0F67 09 ORA #80
0F69 20 JSR EF7B
0F6C 68 PLA
0F6D AA TAX
0F6E CA DEX
0F6F 10 BPL 0F62
0F71 60 RTS

$0F9B A2 LDX #13
0F9D A9 LDA #20
0F9F 9D STA A438,X
0FA2 CA DEX
0FA3 10 BPL 0F9F
0FA5 60 RTS

$0F72 8D STA A44C
0F75 A2 LDX #01
0F77 BD LDA A438,X
0F7A CA DEX
0F7B 9D STA A438,X
0F7E E8 INX
0F7F E8 INX
0F80 E0 CPX #15
0F82 90 BCC 0F77
0F84 60 RTS
```

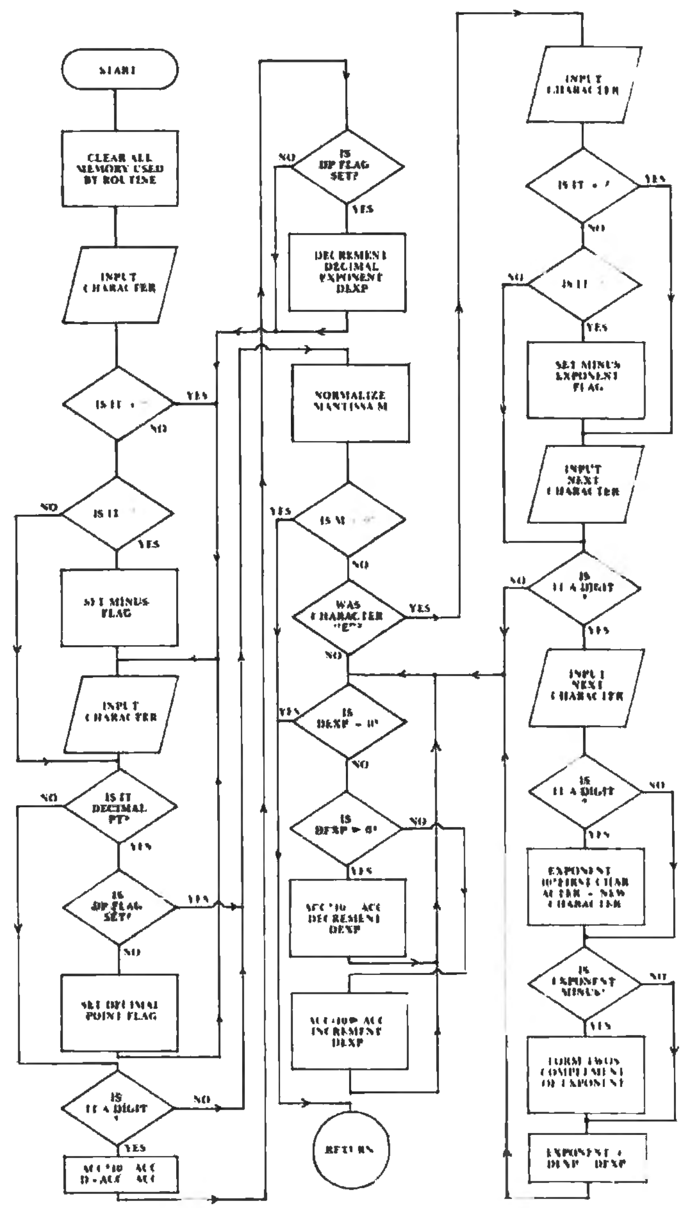


Figure 1. A Flow Chart for the BCD to Floating-Point Binary Routine.

©


ΩMEGA
SALES
CO.
"WHOLESALE COMPUTER PRICES"
DIRECT TO THE PUBLIC
12 Meeting St., Cumberland, RI 02864
PRODUCT SPECIAL
of the MONTH!!
Intertec Superbrain
32K Ram - $2449
64K Ram - $2649
NEC spinwriter
5510-5530 - $2449
Apple II - 16K $ 949
48K $1099
Epson MX-80 - $599
Atari 800 - $769
Televideo
912 B or C — $699
920 C — $769
Diablo 630
$1995
$2195 (with tractor feed)
Products are
NOW
IN
STOCK
AT
ΩMEGA
Sales
Co.
Centronics
(limited quantities-
call for availability)
779-II $ 749
704-9 $1500
730-3 $ 599
737-1 $ 699
ΩMEGA OFFERS THE BEST DELIVERY AND PRICE ON:
APPLE • ATARI • TRS-80 Model II • INTERTEC •
DIABLO • EPSON • HEWLETT—PACKARD • SOROC •
COMMODORE • NEC • QUME • CENTRONICS
CALL TOLL FREE FOR ΩMEGA'S PRICE!
1-800-556-7586
ΩMEGA sells only quality merchandise to our customers.
ΩMEGA will try to match any current advertised price with similar purchase conditions.
Before you buy anywhere else — be sure to call ΩMEGA Sales Co.
1-401-722-1027
ΩMEGA "A member in good standing of the better business bureau."
ΩMEGA ships via UPS, truck, or air. COD's.
Visa, Mastercharge accepted, with no service charge.
VISA
master charge

Editor's Note: Jim Lowell comments in his cover letter: "The program is aimed at parents, elementary level math teachers, and anyone who wants to brush up on basic math skills. I developed it to aid one of my children who has a learning disability. It succeeded in holding his attention over the recommended one-half hour work periods and in eventually improving his skills."

Basic Math For Fun And Profit

J.R. Lowell,
Whitehouse Station, NJ

"Basic Math For Fun And Profit" is an 8K elementary arithmetic program written for the 16K PET with new ROMs. It makes broad use of the PET's non-destructive cursor, excellent graphics, and programmable sound (with a suitable amplifier).

Though there are many basic math programs available today, each has its own particular shortcomings. One variety treats only a single math mode (addition or subtraction or multiplication or division) in each program. A second may offer all modes in a single program, but randomly choose the level of difficulty for the problems. Others force large numbers of problems in one mode before allowing a new mode to be chosen.

Program Feature Overview

"Basic Math For Fun And Profit" not only overcomes these problems, it also provides a whole host of unique features which make it both easy and fun to use:

Menu selection for the four math modes;

level-of-difficulty selection in each mode;

mode and level change opportunity every 10 problems;

right-to-left entry of answer (just like on paper);

Two chances, with prompts, per problem;

sound effects for prompts and rewards;

randomly generated reward statements;

graphics reward for 10 out of 10 correct answers;

first-name personalization throughout program.

Let's look at the program features in more detail.

Math Mode And Skill Level Selection

After entering his or her name, the program gives the user a menu choice of the four math modes. As soon as a choice is entered (using a "GET" statement), the computer asks for the desired skill level: one to four digits in the problems.

Problem Sets And Prompts (addition, subtraction, multiplication)

The computer now generates — one at a time — 10 randomly configured problems in the chosen mode. The program uses two random problem generators: lines 269-293 for all modes except division and lines 294-316 for division.

Once an answer is entered and RETURN pushed, the computer either rewards or prompts the user. The reward is a pulsing laser-type sound effect plus one of four randomly selected "atta-a-person" statements. The prompts, like the rewards, have two parts that are mutually reinforcing. If the user's answer is too low a low tone is generated followed by a statement to the same effect. A high tone and statement follows too large an answer. In the case of a wrong answer, the program gives a second chance. A correct answer now receives the same reward as an initial right answer; a second error gets the appropriate "too high/low" sound and statement followed by the correct answer. The computer then generates a new problem.

Problem Sets And Prompts (Division)

As above, the computer generates 10 problems at the desired skill level. The prompt system, however, is a bit different because each division problem has two parts: the quotient and the remainder. When the quotient is too high, too low, or correct, the same sounds and statements are generated as in the other modes. If the quotient is correct, but the remainder is wrong, however, a new set of high/low sounds and statements is given. As above, a second chance is provided before the correct answer is given.

Answer Format And Correction

The sub-routine in lines 907-931 is included to make the program as compatible with paper and pencil math as possible. It allows the user to enter his or her answers from right to left as is taught in most math classes. As long as the RETURN key isn't pressed, the answer can be corrected. This routine makes full use of the PET's GET and STRING capabilities.

In division problems the quotient and remainder answers are entered separately to allow for individual corrections.

Problem Set Scores And The Ultimate Reward

After each problem set, the computer gives the number correct and asks if the user wants another set of problems. If the answer is "yes", the program again presents the menu. If the answer is "no", the program thanks the user and ENDs.

If the user has achieved a perfect score — 10 out of 10 — the PET goes "wild": the screen goes blank; a siren sounds; the screen then announces in 3-D letters, "WOW 10." As above, the user can then choose whether or not to continue with a new problem set.

Minus And Pluses

First the minus. If you decide to type up this program, you have about five solid hours of work ahead of you (including debugging any slips in digital dexterity you might make). On the other hand, however, I believe, if you have a need for a really good, complete basic math program, the work will be more than worth it. The major pluses are:

Kids love the program. Compared with the drudgery of paper work alone, the "whistles and bells" provided by this program make math fun.

Kids like the personalization of both the instructions and rewards.

Even mistakes aren't too painful to the user since both sound effects and written prompts are provided to help.

In a one-half hour session, the user can work in several modes at different levels (avoiding the boredom of unchanging masses of single mode problems).

And last, but not least, if you belong to a computer club, you'll be a hero for providing the membership with one of the best basic math programs available for the PET.

Happy typing, and for your kids, happy learning.

```
1 PRINT"ĥ
2 REM BASIC MATH FOR FUN AND PROFIT - ¬
      ¬JIM LOWELL 3/23/80.  PLEASE WRITE ¬
      ¬P.O BOX
3 REM 364, WHITEHOUSE STATION, N. J. ¬
      ¬08889 WITH QUESTIONS OR COMMENTS.
4 PRINT:PRINT:PRINT"+-/*+-/*+-/*+-/*+-/*
      ¬+-/*+-/*+-/*+-/*+-
5 PRINT:PRINT:PRINT SPC(5)"BASIC MATH ¬
      ¬FOR FUN AND PROFIT"
6 PRINT:PRINT:PRINT"+-/*+-/*+-/*+-/*+-/*
      ¬+-/*+-/*+-*/+-*/+-
7 PRINT:PRINT:PRINT"                  ¬
      ¬BY JIM LOWELL
8 FOR I = 1 TO 5000:NEXT: PRINT "ĥ":
      ¬GOTO 50
9 PRINT"ĥ
10 PRINT"
11 PRINT"   SS        SS  S  SS      SS
12 PRINT"  N N%     N N%N M'M M    'M M
13 PRINT" N N %    N N N   M M M   ! M M
14 PRINT" N N  %   N N N     M M M  !  M ¬
      ¬M
15 PRINT" &&    %  &&  &&&&&&&  &&  '   &
      ¬&
16 PRINT" &&    % $&&  &&''  &&  &&$ '   &
      ¬&
17 PRINT" &&    LN &&  &&''  &&  && M:   &
      ¬&
18 PRINT" &&   NN N&&  &&''  &&  &&M MM  &
      ¬&
19 PRINT" && NN N  &&  &&''  &&  && M MM &
      ¬&
20 PRINT" &&N &&  &&  N&&N#M&&M  &&  && M&
      ¬&
21 PRINT" &&&&&&&&&&N  &&&&&&&  M&&&&&&&&&
      ¬&
22 PRINT"              S SSSSS
23 PRINT"             N %%    M
24 PRINT"             && &&&&&&
25 PRINT"             && && %&&
26 PRINT"             && && %&&
27 PRINT"             && && %&&
28 PRINT"             && && %&&
29 PRINT"             && &&$%&&
30 PRINT"             && && M&&
31 PRINT"             && &&&&&&
32 PRINT""
34 PRINT"        GOOD FOR YOU, ";A$
37 FOR N= 1 TO 70:RR= INT(50*RND(1)+50):
      ¬POKE59466,0:POKE 59464,RR:
      ¬POKE 59467,16
38 POKE 59466,15:FOR NN=1 TO 3:NEXT:
      ¬POKE 59467,0:NEXT
40 PRINT"ĥ":S=0
41 GOTO 487
50 PRINT:PRINT:PRINT:PRINT"PLEASE TYPE ¬
      ¬IN YOUR FIRST NAME"
60 PRINT:INPUT"AND PRESS THE 'RETURN' ¬
      ¬KEY";A$
65 PRINT"ĥ
70 PRINT:PRINT:PRINT"THANK YOU, ";A$
75 FOR X=1 TO 2000:NEXT:PRINT"ĥ
80 PRINT:PRINT
90 PRINT"OK ";A$;", NOW I WILL CREATE ¬
      ¬SPECIAL"
100 PRINT:PRINT"MATH PROBLEMS JUST FOR ¬
      ¬YOU.  I WILL"
105 PRINT:PRINT"HELP YOU LEARN HOW TO ¬
      ¬ADD, SUBTRACT,"
110 PRINT:PRINT"MULTIPLY AND DIVIDE. IF ¬
      ¬WE WORK FOR 1/2"
130 PRINT:PRINT"HOUR AT A TIME, IT WILL ¬
      ¬BE MORE LIKE"
133 PRINT:PRINT"FUN THAN WORK.":PRINT
135 PRINT"PRESS ANY KEY TO START"
136 GET C$:IF C$="" THEN 136
139 PRINT"ĥ
140 PRINT:PRINT
141 PRINT"MAKE A CHOICE NOW PLEASE."
142 PRINT:PRINT"TYPE THE NUMBER OF THE ¬
      ¬KIND"
143 PRINT:PRINT"OF PROBLEMS YOU WANT.":
      ¬PRINT
146 PRINT:PRINT"     1 = ADDITION":PRINT
148 PRINT:PRINT"     2 = SUBTRACTION":
      ¬PRINT
150 PRINT:PRINT"     3 = DIVISION":PRINT
152 PRINT:PRINT"     4 = MULTIPLICATION"
      ¬:PRINT
165 GET G$:IF G$="" GOTO 165
166 IF G$<"1" OR G$>"4" THEN GOTO 165
167 PRINT"ĥ
168 G=VAL(G$)
170 ON G GOTO 180,190,200,210
180 PRINT"FINE ";A$;", NOW I WILL GIVE ¬
      ¬YOU 10"
181 PRINT:PRINT"rADDITIONf PROBLEMS.":
      ¬FOR X=1 TO 4000:NEXT:PRINT"ĥ
182 PRINT:PRINT:PRINT"IF YOU MISS A ¬
      ¬PROBLEM, I WILL GIVE"
183 PRINT:PRINT"YOU A HINT TO HELP YOU. ¬
      ¬ THEN YOU'LL"
184 PRINT:PRINT"HAVE ONE MORE CHANCE ¬
      ¬BEFORE I GIVE"
185 PRINT:PRINT"YOU THE RIGHT ANSWER.":
      ¬FOR T=1 TO 10000:NEXT
```

```
186 GOSUB 220
187 IF G<>3 THEN 270:REM GO TO PROBLEM ¬
      ¬GENERATOR
188 GOTO 294:REM PROBLEM GENERATOR FOR ¬
      ¬DIVISION ONLY
190 PRINT"OK ";A$;", ɼSUBTRACTIONf̂ IT ¬
      ¬IS":FOR X=1TO4000:NEXT:PRINT"ĥ":
      ¬GOTO 182
200 PRINT"ALRIGHT ";A$;", THIS TIME ¬
      ¬WE'LL TRY"
205 PRINT:PRINT"ɼDIVISION f̂PROBLEMS.":
      ¬FOR X=1TO4000:NEXT:PRINT"ĥ":
      ¬GOTO 182
210 PRINT:PRINT"OK ";A$;", LET' HAVE A ¬
      ¬GO AT SOME"
211 PRINT:PRINT"ɼMULTIPLICATIONf̂.":
      ¬FOR X=1TO4000:NEXT:PRINT"ĥ":
      ¬GOTO 182
213 REM CHOOSE NUMBER OF DIGITS TO BE ¬
      ¬GENERATED FOR EACH PROBLEM
220 PRINT"ĥ
222 PRINT"HOW MANY NUMBERS WOULD YOU ¬
      ¬LIKE"
230 PRINT:PRINT"IN YOUR PROBLEMS, 1,2,
      ¬3 OR 4 ?"
232 GET B$:IF B$="" THEN GOTO 232
235 IF B$<"1" OR B$>"4" THEN GOTO 232
237 B=VAL(B$):PRINT"ĥ"
240 PRINT:PRINT:PRINT"ALRIGHT ";A$;",
      ¬ I'LL CREATE ";B;"- NUMBER"
245 PRINT:PRINT"PROBLEMS FOR YOU. YOU ¬
      ¬WILL PROBABLY WANT"
246 PRINT:PRINT"TO DO THE MORE COMPLEX ¬
      ¬PROBLEMS ON PAPER"
247 PRINT:PRINT"BEFORE TYPING YOUR ¬
      ¬ANSWER ON THE SCREEN."
250 PRINT:PRINT"PRESS ANY KEY WHEN ¬
      ¬YOU'RE READY."
254 GET B$:IF B$ ="" THEN 254
260 PRINT"ĥ
261 RETURN: REM GO BACK TO 187
269 REM RANDOM NUMBER PROBLEM GENERATOR
270 PRINT:C=0
271 C=C+1:IF C>10 GOTO 482
272 PRINT:PRINT"ɼPROBLEM #";C
273 LET X=INT((10^B-1)*RND(1)+1)
274 IF X< 10^(B-1) THEN 273
280 LET Y=INT((10^B-1)*RND(1)+1)
285 IF Y< 10^(B-1) THEN 280
287 O=0
291 IF G=2 THEN GOTO 600: REM SUBTRACTIO
      ¬N SUB-ROUTINE
292 IF G=4 THEN GOTO 500: REM MULTIPLICA
      ¬TION SUB-ROUTINE
293 GOTO 336: REM ADDITION SUB-ROUTINE
294 PRINT:PRINT"ĥ":H=0
295 H=H+1:IF H>10 THEN GOTO 482:
      ¬REM GENERATES PROBLEMS FOR ¬
      ¬DIVISION ONLY.
296 PRINT"ɼPROBLEM #";H
298 LET F=INT((10^B-1)*RND(1)+1)
299 IF F<10^(B-1) THEN GOTO 298
305 LET Y=INT(10^5*RND(1)+1)
306 IF F>Y THEN 305
307 PRINT
316 O=0:GOTO 318
317 O=2:REM MARKER FOR 1ST WRONG ANSWER
318 PRINT:PRINT"USE ɼDELETEf̂ KEY FOR ¬
      ¬CORRECTIONS."
320 PRINT:PRINT:PRINT"             ɼ";Y;"/";
      ¬F
321 PRINT:PRINT:PRINT"TYPE YOUR ¬
      ¬WHOLE-NUMBER ANSWER,THEN"
322 PRINT:PRINT"PRESS 'RETURN', AND ¬
      ¬TYPE THE REMAINDER."
323 PRINT:PRINT"IF THERE IS NO REMAINDER
      ¬, TYPE A 0.
324 PRINT:PRINT:INPUT"YOUR ɼWHOLE-NUMBER
      ¬f̂ ANSWER =";Z
325 PRINT:INPUT"ENTER ɼREMAINDERf̂ AND ¬
      ¬PRESS'RETURN'.";R:PRINT"ĥ
326 M=INT(Y/F):U=Y-(M*F):REM DETERMINE ¬
      ¬REMAINDER
328 IF M=Z AND R=U THEN 1000
329 IF M=Z AND R<>U THEN 3999
330 IF M>Z THEN 2000
331 IF M<Z THEN 3000
333 O=2
334 IF G=4 THEN 500
335 IFG=2 THEN 605
336 PRINT:PRINT"      ";X
340 PRINT"     +";Y
345 PRINT"      -----  TYPE ANSWER,
      ¬ PRESS RETURN
346 PRINT:PRINT"             USE THE ¬
      ¬ɼSPACE-BARf̂ TO"
347 PRINT:PRINT"             DELETE ANY ¬
      ¬ERRORS"
350 M=X+Y
355 PRINT"↑↑↑↑";SPC(B+5):GOTO907
357 PRINT"ĥ
358 IF W=M THEN 1000
359 IF W<M THEN 2000
365 IF W>M THEN 3000
370 E=INT(4*RND(1)+1)
380 ON E GOTO 390,400,410,420:REM PICK ¬
      ¬AN "ATTAPERSON" STATEMENT.
390 PRINT"GOOD SHOW ";A$;", I'M PROUD ¬
      ¬OF YOU!":REM PRINT AND INCREMENT ¬
      ¬SCORE.
395 S=S+1:O=0:IF G<>3 GOTO 271
396 GOTO 295
400 PRINT"THAT'S IT ";A$;", WAY TO GO!!"
405 S=S+1:O=0:IF G<>3 GOTO 271
406 GOTO 295
410 PRINT"RIGHT YOU ARE ";A$;".  KEEP ¬
      ¬IT UP."
415 S=S+1:O=0:IF G<>3 GOTO 271
416 GOTO 295
420 PRINT"YOU GOT IT RIGHT.  HOORAY ¬
      ¬";A$;"!!!"
425 S=S+1:O=0:IF G<>3 GOTO 271
426 GOTO 295
430 PRINT"SORRY ";A$;", THAT'S TOO ¬
      ¬ɼLOW."
434 IF O >1 AND G<>3 THEN GOTO 470
435 IF O >1 AND G=3 THEN GOTO 650
436 IF O <2 AND G<>3 THEN GOTO 333
437 GOTO 317:REM GIVE PROB. AGAIN.
450 PRINT"NICE TRY ";A$;", BUT THAT'S ¬
      ¬TOO ɼHIGH."
459 IF O>1 AND G<>3 THEN GOTO 470
460 IF O>1 AND G=3 THEN GOTO 650
461 IF O<2 AND G<>3 THEN GOTO 333
462 GOTO 317
465 REM GIVE CORRECT ANSWER.
470 PRINT:PRINT"THAT'S TWO CHANCES ¬
      ¬";A$;"."
471 PRINT:PRINT"THE CORRECT ANSWER IS ¬
      ¬";M:FORX=1TO5000:NEXT:PRINT"ĥ
472 PRINT"                    ---------":
      ¬PRINT:PRINT
```

```
475 O=Ø
480 GOTO 271
482 FOR I=1TO1500:NEXT:PRINT"ĥ
483 IF S=1Ø THEN PRINT"ĥ":GOTO 5ØØØ
485 PRINT:PRINT:PRINT"YOU GOT ";S;" ¬
     ¬RIGHT FOR THIS SET ";A$:PRINT:
     ¬PRINT
486 S=Ø
487 PRINT"WOULD YOU LIKE TO TRY ANOTHER ¬
     ¬SET OF"
489 PRINT:PRINT"PROBLEMS? ";:PRINT"TYPE ¬
     ¬Y OR N"
49Ø GET E$:IF E$="" THEN GOTO 49Ø:
     ¬PRINT"ĥ"
491 IF E$="Y" THEN GOTO 14Ø
492 PRINT"ĥ
493 PRINT:PRINT:PRINT:PRINT"OK, ";A$;". ¬
     ¬WE'LL CALL IT QUITS FOR NOW."
495 PRINT:PRINT"THANKS FOR USING ME ¬
     ¬TODAY. SEE YOU ɽSOON."
496 PRINT:PRINT:FOR X=1TO 8Ø:PRINT"^ɽ^ȓ"
     ¬;:NEXT
497 END
5ØØ PRINT:PRINT:PRINT"        ";X
5Ø5 PRINT"    X  ";Y
51Ø PRINT"    ---------   TYPE ANSWER,
     ¬ PRESS RETURN"
513 PRINT"                USE THE ¬
     ¬ɽSPACE-BAR ȓTO"
515 PRINT:PRINT"                DELETE ¬
     ¬ANY ERRORS"
517 M=X*Y
52Ø PRINT"↑↑↑↑";SPC(B+7):GOTO9Ø7
6ØØ IF Y>X THEN 28Ø
6Ø5 PRINT:PRINT"       ";X
61Ø PRINT"    -";Y
62Ø PRINT"     -----  TYPE ANSWER,
     ¬ PRESS RETURN
621 PRINT:PRINT"              USE THE ¬
     ¬ɽSPACE-BARȓ TO"
622 PRINT:PRINT"              DELETE ANY ¬
     ¬ERRORS"
625 M=X-Y
626 PRINT"↑↑↑↑";SPC(B+5):GOTO 9Ø7
63Ø PRINT"GOOD TRY ";A$;".  YOUR ANSWER ¬
     ¬"
631 PRINT:PRINT"OFɽ ";Z;"ȓ IS RIGHT,
     ¬ BUT YOUR"
632 PRINT:PRINT"REMAINDER IS WRONG.":
     ¬FOR X=1TO 55ØØ:NEXT:PRINT"ĥ"
634 IF O>1 THEN 65Ø
636 IF O<2 THEN 317
65Ø PRINT:PRINT"THE CORRECT ANSWER IS ¬
     ¬ɽ";M
651 PRINT:PRINT"AND THE REMAINDER IS ¬
     ¬ɽ";U:FOR X=1TO 65ØØ:NEXT:PRINT"ĥ"
652 GOTO 295
9Ø7 T=Ø
9Ø8 FOR I=1TO 9:Z$(I)=STR$(Ø):NEXT
91Ø FOR I=1TO 9
913 T=T+1
92Ø GETZ$(T):IF Z$(T)=""THEN 92Ø
921 IF Z$(T)=CHR$(32)THEN Z$(T)=STR$(Ø):
     ¬GOTO931
922 IFZ$(T)=CHR$(13)THEN Z$(T)=STR$(Ø):
     ¬GOTO357
923 PRINTZ$(T)"←←";
924 Y$=Z$(9)+Z$(8)+Z$(7)+Z$(6)+Z$(5)+Z$(
     ¬4)+Z$(3)+Z$(2)+Z$(1):W=VAL(Y$)
925 NEXT
931 PRINT"→"CHR$(32)"←";:T=T-1:I=I-1:
     ¬GOTO92Ø
999 REM RIGHT ANSWER PHASER SOUND
1ØØØ POKE 59466,Ø:POKE 59467,16:
     ¬POKE 59466,15:FOR N= 1 TO 3
1Ø1Ø FOR NN= 3Ø TO 255 STEP 6:POKE ¬
     ¬59464,NN:NEXT:NEXT
1Ø2Ø POKE 59467,Ø:GOTO 37Ø
1999 REM LOW ANSWER SOUND
2ØØØ POKE 59466,Ø: POKE 59464,255:
     ¬POKE 59467,16:POKE 59466,1
2Ø1Ø FOR N= 1 TO 12ØØ:NEXT
2Ø2Ø POKE 59467,Ø: GOTO 43Ø
2999 REM HIGH ANSWER SOUND
3ØØØ POKE 59466,Ø: POKE 59464,1ØØ:
     ¬ POKE 59467,16: POKE 59466,2ØØ
3Ø1Ø FOR N= 1 TO 12ØØ:NEXT
3Ø2Ø POKE 59467,Ø:GOTO 45Ø
3281 F Z=M AND R=U THEN 1ØØØ:IF M=Z AND ¬
     ¬R<>U THEN 4ØØØ:IF Z>M THEN 3ØØØ
3999 REM RIGHT ANSWER, WRONG REMAINDER ¬
     ¬SOUND
4ØØØ C=Ø
4ØØ5 IF C=5 THEN 63Ø
4Ø1Ø POKE 59464,15Ø: POKE 59467,16:
     ¬ POKE 59466,15:FORN=1TO75:NEXT:
     ¬ POKE 59467,Ø
4Ø2Ø FOR X=1 TO 5ØØ:NEXT X
4Ø25 C=C+1
4Ø3Ø GOTO 4ØØ5
4999 REM SOUND FOR 1Ø OUT OF 1Ø RIGHT
5ØØØ PRINT"ĥ
5ØØ5 POKE 59466,Ø: POKE 59467,16:
     ¬ POKE 59466,51: FOR N= 1 TO 5
5Ø1Ø FOR NN= 225 TO 12Ø STEP-2: POKE ¬
     ¬59464,NN:NEXT:FOR NN= 12Ø TO 255 ¬
     ¬STEP 2
5Ø2Ø POKE 59464,NN:NEXT:NEXT:POKE 59467,
     ¬Ø
5Ø3Ø GOTO 9
```

©

PET Spelling Lessons Your Students Can Prepare

Tory Esbensen
Minneapolis, MN

This article presents and explains the format for a spelling program that requires only the addition of some data lines in order to become fully operational. The needed data lines are so easy to create that even elementary school students (grades four and up) should be able to do the job.

My own experience as a professional educator indicates that drill and practice spelling tapes for microcomputers are among those programs most frequently requested by classroom teachers. In some instances, the need is for programs that run "on all fours" with a particular set of spelling workbooks. In other cases, teachers would like to have programs that focus on certain groups of words identified as Spelling Demons. Sometimes, there is a desire to shape word lists that will meet the needs of individual students.

The program listed in this article is called **GUESS THAT WORD.** It is offered to the readers of COMPUTE as one way of developing a flexible response to the demand on the part of teachers for microcomputer spelling exercises that can be tailor-made to fit individual learning objectives.

Figure 1 is the program listing of GUESS THAT WORD. Lines 7000-7999 are for entering spelling words as data. Multiple spelling lists can be entered. Each list should be preceded by a number identifying the list. An arrow pointing up concludes each list. Lines 7000-7010 are the data lines for the first spelling list. Note that all data entries are separated by commas.

Typing data line entries is the only thing that needs to be done in order to complete the GUESS THAT WORD program. Once students are provided with the word lists to be entered, typing them as data line entries should be a relatively simple task. Following this, the data lines should be checked to spot any typographical errors, and the entire program should be run to identify any operational errors. These are the final steps in the process. When this has been accomplished, the program is complete.

Briefly, here is how GUESS THAT WORD works when the program is run:

1. As requested by the computer, the student types in the number of the desired word list.

2. The computer randomly selects a word from this list and, near the top of the screen, prints a row of gray boxes equivalent in length to the length of the chosen word.

3. The student now has three choices. He/she can (a) try to guess the entire word, (b) guess a single letter, or (c) ask the computer to reveal a letter of the word.

4. If the student tries to guess the word, 100 points are won if the guess is right, and 5 points are lost if the guess is wrong.

5. If the student tries to guess a letter, the cost of the guess (regardless of its accuracy) is 1 point. If the student guesses correctly, all such letters in the word are revealed. If the student's guess is wrong, no letters are revealed.

6. If the student asks the computer to show a letter, only one letter is revealed even though more than one such letter may be in the word. The cost of this option is always 2 points.

7. When the student finally guesses the word, the computer summarizes the results on an ongoing basis. This includes the average score per word, plus a list of the specific words presented by the computer.

The program listing in Figure 1 shows 5 lists of words sometimes identified as Spelling Demons. These lists can be changed simply by changing the data line entries.

Readers who wish to copy this program listing are invited to do so. Readers who do not want to bother with this may purchase the program tape itself for $7.95 from MICRO-ED, Inc., P.O. Box 24156, Minneapolis, Minnesota 55424.

The author would be glad to respond to questions and comments from interested readers.

```
8 POKE59468,12
9 REM WRITTEN BY T. ESBENSEN FOR
10 PRINT"ĥ↓↓↓↓↓↓→→→→→→→→→→rMICRO-EDr̂,
     ¬ INC."
11 PRINT"↓→→→→→→→→→→P.O. BOX 24156"
12 PRINT"↓→→→→→→→→→→MINNEAPOLIS, ¬
     ¬MINNESOTA 55424"
15 FORZ=1TO2000:NEXTZ
70 SP$="                                       ¬
     ¬         "
80 DIM W$(51),M%(51),WD$(51),MM(25)
90 PRINT"ĥ":TI$="000000"
95 PRINT"↓↓"
101 PRINT"r→→→→→→→→→*******************
102 PRINT"r→→→→→→→→→→*                 *
103 PRINT"r→→→→→→→→→→* GUESS THAT WORD! *
104 PRINT"r→→→→→→→→→→*                 *
105 PRINT"r→→→→→→→→→*******************
110 PRINT"↓"
165 PRINT"↓WHICH WORD LIST DO YOU WANT";
170 INPUT"→→?←←←";LI$
175 PRINT"ĥ"
180 RESTORE
190 READD$:IFD$="^^"THENPRINT"ĥ↓NO SUCH ¬
     ¬LIST.  TRY AGAIN.":GOTO165
192 PRINT"h↓↓↓↓↓↓→→→→→→→→→→→→→→→rSEARCHI
     ¬NG"
```

```
195 IFD$<>LI$THEN190
300 W=W+1:READW$(W):IFW$(W)<>"^"THEN300
400 N=INT(RND(TI)*(W-1))+1
410 IFM%(N)=-1THEN400
420 M%(N)=-1
430 PRINT"ĥ↓↓↓>>>>>>>>>>>";
440 FORZ=1TOLEN(W$(N))
450 PRINT"&̲";
460 NEXTZ
500 PRINT"h̲↓↓↓↓↓↓PRESS THE NUMBER OF ¬
     ¬YOUR CHOICE:"
510 PRINT"↓r̲1r̂>READY TO GUESS THE WORD"
520 PRINT"↓r̲2r̂>READY TO GUESS A LETTER"
530 PRINT"↓r̲3r̂>COMPUTER SHOULD SHOW A ¬
     ¬LETTER"
550 GETG$:IFG$<>"1"ANDG$<>"2"ANDG$<>"3"T
     ¬HEN550
560 PRINT"h̲↓↓↓↓↓"
570 FORZ=1TO8:PRINTSP$:NEXTZ
575 GETZ$:IFZ$<>""THEN575
580 G=VAL(G$):ONGGOTO1000,2000,3000,4000
1000 PRINT"h̲↓↓↓↓↓↓WHAT IS THE WORD?"
1010 PRINT"↓WIN r̲100r̂ POINTS IF YOU GET ¬
     ¬IT."
1020 PRINT"↓LOSE r̲5r̂ POINTS IF YOU MISS ¬
     ¬IT."
1030 PRINT"↓"
1050 INPUT">>2̲<<<";R$
1055 IFR$="2̲"THEN1000
1060 IFR$=W$(N)THENPRINT"↓r̲RIGHT":
     ¬PT=PT+100:FORZ=1TO1000:NEXTZ:
     ¬GOTO9000
1070 IFR$<>W$(N)THENPRINT"↓r̲WRONG"
1080 PT=PT-5:FORZ=1TO1000:NEXTZ
1560 PRINT"h̲↓↓↓↓↓"
1570 FORZ=1TO10:PRINTSP$:NEXTZ
1600 GOTO500
2000 PRINT"h̲↓↓↓↓↓↓WHAT LETTER DO YOU ¬
     ¬GUESS?"
2010 PRINT"↓THE COST IS r̲1r̂ POINT PER ¬
     ¬GUESS."
2030 PRINT"↓"
2050 INPUT">>2̲<<<";R$
2052 IFR$="2̲"THEN2000
2055 IFLEN(R$)>1THENPRINT"h̲↓↓↓↓↓":
     ¬FORZ=1TO10:PRINTSP$:NEXTZ:GOTO2000
2057 PRINT"h̲↓↓↓>>>>>>>>>>>";
2060 FORZ=1TOLEN(W$(N))
2070 IFR$=MID$(W$(N),Z,1)THENPRINTR$;:
     ¬MM(Z)=-1
2080 IFR$<>MID$(W$(N),Z,1)THENPRINT">";
2090 NEXTZ
2095 PT=PT-1
2560 PRINT"h̲↓↓↓↓↓"
2570 FORZ=1TO10:PRINTSP$:NEXTZ
2600 GOTO500
3000 PRINT"h̲↓↓↓↓↓↓THE COMPUTER WILL ¬
     ¬SHOW A LETTER."
3010 PRINT"↓IT WILL COST YOU r̲2r̂ ¬
     ¬POINTS."
3020 FORZ=1TO1000:NEXTZ
3030 NN=INT(RND(TI)*LEN(W$(N)))+1
3040 IFMM(NN)=-1THEN3030
3050 MM(NN)=-1
3060 PRINT"h̲↓↓↓>>>>>>>>>>>";
3070 FORZ=1TONN
3080 PRINT">";
3090 NEXTZ
3100 PRINTMID$(W$(N),NN,1)
3110 PT=PT-2
3560 PRINT"h̲↓↓↓↓↓"
3570 FORZ=1TO10:PRINTSP$:NEXTZ
3600 GOTO500
7000 DATA 1,WHETHER,WRITING,THROUGH,
     ¬ACHE,DOCTOR,KNOW,LAID,EARLY,
     ¬MAKING,BELIEVE
7010 DATA OFTEN,FRIEND,PIECE,GUESS,
     ¬RAISE,CHOOSE,HOARSE,SEPARATE,
     ¬INSTEAD,JUST,^
7020 DATA 2,WOMEN,WEDNESDAY,TONIGHT,
     ¬VERY,STRAIGHT,AGAIN,DOES,LOOSE,
     ¬EASY,BEEN
7030 DATA FEBRUARY,NONE,ONCE,GRAMMAR,
     ¬HALF,READ,COLOR,SAID,HOUR,KNEW,^
7040 DATA 3,ALWAYS,DONE,LOSE,AMONG,
     ¬ANSWER,MANY,MEANT,MINUTE,ENOUGH,
     ¬EVERY,FORTY
7050 DATA BEGINNING,CHANCE,HAVING,SAYS,
     ¬SOME,TOO,WHICH,THEIR,WEAR,^
7060 DATA 4,ANY,EXISTENCE,MUCH,BLUE,
     ¬BREAK,BUILT,BUSINESS,BUSY,BUY,
     ¬READY,HEAR
7070 DATA HERE,COMING,SEEMS,COUGH,COULD,
     ¬SHOES,COUNTRY,SIMILAR,DEAR,^
7080 DATA 5,SINCE,SUGAR,TROUBLE,WHOLE,
     ¬SURE,TRULY,TEAR,TUESDAY,WHILE,TWO,
     ¬WOULD
7090 DATA DUMB,THERE,WRITE,THEY,THOUGH,
     ¬WROTE,TIRED,WEEK,WHERE,^
8000 DATA ^^
9000 PRINT"ĥ"
9002 FORZ=1TOLEN(W$(N)):MM(Z)=0:NEXTZ
9005 P=P+1
9010 PRINT"↓r̲LIST:r̂>";LI$;">>>>>>>>>r̲WORD
     ¬ PROBLEM:r̂";P
9015 SC=INT((PT/P)+.5)
9020 PRINT"↓r̲AVERAGE SCORE PER WORD:
     ¬r̂";SC
9050 PRINT"↓WORDS PRESENTED BY COMPUTER"
9060 PRINT"############################"
9070 WD$(P)=W$(N)
9080 FORZ=1TOP
9090 PRINT"r̲";WD$(Z);">";
9100 C=C+1:IFC=3THENC=0:PRINT
9110 NEXTZ
9115 PRINT
9117 C=0
9118 IFP=W-1THENEND
9120 PRINT"↓PRESS S↓←#↑ TO STOP OR ¬
     ¬C↓←#↑ TO CONTINUE."
9130 GETG$:IFG$<>"S"ANDG$<>"C"THEN9130
9140 IFG$="C"THEN400
9999 END
```

©

Atari® 800™ 16K Personal Computer

List $1080

only $779

ATARI®

Atari® Peripherals:

400 8K	$ 389
410 Recorder	59
815 Disk	1139
810 Disk	519
822 Printer	349
825 Printer	749
830 Modem	149
850 Interface Module	169

Atari® Accessories

CX852 8K RAM	$ 89
CX853 RAM	144
CX70 Light Pen	59
CX30 Paddle	17
CX40 Joystick	17
CX8100 Blank Diskettes (5/box)	21

Atari® Software

Entertainment:	
CXL4004 Basketball	$29
CX4105 Blackjack	12
CXL4009 Chess	29
CXL4011 Star Raiders™	44
CX4111 Space Invaders	17
CXL4006 Super Breakout™	29
CXL4010 3-D Tic-Tac-Toe	29
CXL4005 Video Easel™	29
Personal Interest & Developement	
CXL4104 Mailing List	$16
CX4107 Biorhythm	12
CXL4007 Music Composer	44
CX4110 Typing	19
CX4101 An Invitation to Programming	16
Information & Communication:	
CXL4015 TeleLink™	19
Programming Languages:	
CXL4003 Assembler Editor	45
CXL4002 BASIC Computing Language	45

Call For Prices On:

Commodore Computers
Professional Software
Personal Software

Education: (Talk & Teach Courseware)	
CXL4001 Education System Master	$20
CX6001 U.S. History	22
CX6002 U.S. Government	22
CX6003 Supervisory Skills	22
CX6004 World History (Western)	22
CX6005 Basic Sociology	22
CX6006 Counseling Proceedures	22
CX6007 Principles of Accounting	22
CX6008 Physics	22
CX6009 Great Classics	22
CX6010 Business Communications	22
CX6011 Basic Psychology	22
CX6012 Effective Writing	22
CX6014 Principles of Economics	22
CX6015 Spelling	22
CX6016 Basic Electricity	22
CX6017 Basic Algebra	22
Professional Applications:	
CX8102 Calculator	$22
CX4109 Graph It	16
CX4103 Statistics	16
Investment Analysis:	
CX8106 Bond Analysis	$ 19
CX8107 Stock Analysis	19
CX8101 Stock Charting	19

To Order:

Phone Orders invited. Or send cashiers check or money order. Equipment shipped UPS collect. Pennsylvania residents add 6% sales tax. Equipment is subject to price change and availability without notice.

Computer Mail Order
501 E. Third St.
Williamsport, PA 17701
(717) 323-7921

McTerm

MADISON COMPUTER

A Telecommunication Package

With the McTerm package your Commodore computer can now become a very intelligent terminal. It allows you to easily participate in the electronic mail revolution. McTerm can communicate simultaneously with another PET or mainframe. You can transfer entire disk files to other disk drives; send or receive program, WordPro, sequential, or relative files. This package works on Commodore computers with 3.0 or 4.0 Basic, 1.0 or 2.0 DOS, 40 or 80 column computers, and 2040 or 8050 disk drives (requires the use of TTL Level RS-232 modems).

Complete with Program, ROM, Cable and Manual.
Price: $195.00

Features include:

- -Baud selection from 75 to 1200.
- -Full and half duplex, also local echo.
- -Supports odd, even and mark parity.
- -CRC, error checking for PET to PET files.

1825 MONROE STREET, MADISON, WI 53711 (608) 255-5552
-Dealer Inquiries Are Welcome-

THE apple® GAZETTE

List Apple Integer Basic Programs One Page At A Time

Keith Falkner,
Toronto, Canada

The obvious way to examine an unfamiliar program is to type "LIST". In APPLE's INTEGER BASIC, this often gives little or no useful information, because the whole program is listed at great speed, and the moving display can scarcely be read. If the listing could be stopped, this would be no problem; however, only the RESET key stops the listing. Pressing the RESET key is brutal and inelegant, and can cause loss of the program being listed.

This small program in Assembly Language provides a convenient way to list INTEGER BASIC programs without those two problems. It lists one screen-full of the BASIC program, then waits for any key to be pressed. If any key but CTRL-C is pressed, the next screen-full of the program is listed, and so on until the whole BASIC program has been displayed. At any time, CTRL-C can be entered, and the listing ceases, with one screen-full of the BASIC program still visible. This makes it simple to browse an INTEGER BASIC program either quickly or slowly, and stop after any screen-load ("page").

This program does not interfere with BASIC, and as listed here, it occupies a part of memory where it will not likely be damaged. Locations 700-762 ($2BC-$2FA) are approximately the final quarter of the 256-byte keyboard input buffer, and are used only if more than 188 characters are entered as a line of BASIC or in reponse to an INPUT instruction. Either of these is very unusual, and in practice, the program is not over-written.

Users with little experience in machine language can easily enter this program with the Mini-Assembler which is part of APPLE's Monitor, as follows:

```
> CALL - 151 (enter the Machine-language monitor)
*F666G        (enter the Mini-Assembler)
!2BC:LDACA (no need to type spaces or $)
! STAE2       (a space is needed after the !)
! LDACB       (and so on ...)
```

For the "branch" instructions, BCC, BNE, BCS, and BPL, the actual address branched-to is needed. For example:

```
! CMP4D       (the instruction on line 0027)
! BNE2E0      (it branches to SHOWME at $2E0)
```

APPLE suggests using the RESET key to exit the Mini-Assembler, but there is a gentler way:

```
!$FF69G       (type it as shown, with no spaces)
```

The program can be saved on disk via:

```
*BSAVE LISTAPAGE,A700,L63
```

It can be saved onto tape via:

```
*2BC.2FAW (there will be only 1 "beep")
```

At any time, this program can be loaded into memory without disturbing any BASIC program already present. To load it from disk, type:

```
>BLOAD LISTAPAGE
```

To load it from tape, a more complicated sequence is needed:

```
>CALL -151 (to Monitor again)
*2BC.2FAR   (press PLAY before pressing RETURN)
*E003G      (or CTRL-C)
```

In either case, the program is safely hiding in locations 700-762 inclusive, and it can be used in these ways:

To list a BASIC program from the beginning, just type "CALL 700" to see the first page. Press any key except CTRL-C to see more, or press CTRL-C to stop listing after any page.

The program has a second entry-point which is also useful. Type "CALL 708" to resume listing a program after the line most recently listed. For example, to list some lines starting with line 2000, type "LIST 1999", whether or not such a line exists, then type "CALL 708", and successive pages starting with line 2000 will be listed. "CALL 708" can also be used to resume a listing which had been begun by "CALL 700" and stopped by CTRL-C.

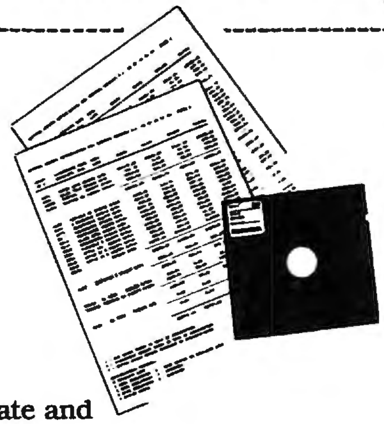
Experienced users of machine-language will have noticed that this program is relocatable. In other words, it does not contain any reference to its own absolute address. That in turn means that it can occupy any locations in memory that are not in use for other purposes, and function there without needing any changes. Other locations which can be used to contain this program include, from most convenient to least:

Page 3, locations 768-830 (or nearby) is easiest because neither the APPLE monitor nor BASIC makes use of this space, hence of course, it is the popular place for noise-making routines and various

FOR: **apple*** **FROM:** Innerface BUSINESS SYSTEMS

Capital Asset Managment System™

SIMPLE,
CONCISE,
FUNCTIONAL,
ACCURATE,
EFFICIENT,
USABLE
SOFTWARE from I.B.S.



Having developed accurate and understandable software for both government and business, I.B.S. has at last Gone Public, with C.A.M.S. —

CAPITAL ASSETS MANAGEMENT SYSTEM, a simple, easy-to-use system designed for the APPLE Computer* and you.

- DEPRECIATES assets according to one of the eight approved schedules
- INVESTMENT CREDIT and RECAPTURE are computed automatically
- PROJECTIONS aid in the selection of method and term for new assets
- PERSONALIZED REPORTS in either 80 or 132 column formats
- AUTOMATIC SUBTOTALING selected by the user
- SAFTEY ROUTINES check input for most errors
- AUTOMATIC DISK FILING with user control

CAMS:

With just a few clear and simple keystrokes, you can view any of two hundred assets depreciated FROM any time TO any time, using any of the eight schedules.

By SEEING the results, you can make accurate decisions NOW, when it counts.

CAMS will automatically search your records for assets which qualify for either Investment Credit or Investment Credit Tax Recapture.

CAMS prompts you with clear and simple messages on the screen for all entries. Special safety routines check all practical inputs for errors. Because CAMS maintains all files automatically, it serves not only as a tool for projections, but as a RECORD FILING system as well.

And since no one should be limited by software, INNERFACE has made CAMS for user back-up copies as well. Because of this, you can record an UNLIMITED number of assets on individual disks for privacy and safe-keeping.

CAMS records 23 pieces of information on each asset: GL accounts; memos; dates; costs; method & life; full description; user ID/dept.; other deductions.
CAMS reports on depreciation: Straight-line; 125%, 150%, 200% w/wo automatic switch to Straight-line, Sum-of-year-digits AND Investment Tax Credit & Recapture amounts for each year.
CAMS requires a 48K Apple with disk and (at least) 80-column printer.
Cams is fully supported by professional documentation and by I.B.S.

* APPLE is a trademark of Apple Computer Inc.

Innerface BUSINESS SYSTEMS

CAMS is 99.50 $2.00 SHIPPING CA RES ADD 5.97 TAX

box 834, pacific grove CA 93950

other uses which would conflict with this.

2048-2110. In issue CLR and LOMEM:2110 to prevent BASIC variables from over-writing the routine.

16322-16384. Issue NEW and HIMEM:16322 to prevent a BASIC program from over-writing the routine. Those addresses interfere with High-Resolution graphics, and will be different in a machine with more or less than 16K of memory.

Other locations, such as the gap above the variables and below the program might be tried if none of the above appeals. Experiment at will in this fashion, and remember, "You can't hurt the computer by pressing keys".

This program provides a helpful alternative to the "LIST" command, filling an irritating gap in APPLE's flexible and rapid INTEGER BASIC.

```
LOC  CODE    LINE

0000                 .OPT NOSYM
0000         ;  LIST INTEGER BASIC PROGRAM
0000         ;  ONE SCREEN-FULL AT A TIME:
0000         ;
0000                 *=700
02BC         ;
02BC         ;  **ENTER HERE TO LIST FROM START
02BC         ;
02BC A5CA            LDA $CA
02BE 85E2            STA $E2        ;INIT POINTERS TO
02C0 A5CB            LDA $CB        ;START OF PROGRAM
02C2 85E3            STA $E3
02C4         ;
02C4         ;  **ENTER HERE TO RESUME LISTING
02C4         ;
02C4 A54C            LDA $4C
02C6 85E6            STA $E6        ;LIST UNTIL
02C8 A54D            LDA $4D        ;HIMEM: HIT
02CA 85E7            STA $E7
02CC 2C10C0  RESUME  BIT $C010      ;RESET KEYBOARD
02CF 2058FC          JSR $FC58      ;CLEAR THE SCREEN
02D2         ;
02D2         ;  SEE IF THERE IS MORE TO LIST.
02D2 A5E3    ANYMOR  LDA $E3
02D4 C54D            CMP $4D        ;ALL DONE?
02D6 9008            BCC SHOWME     ;NO.
02D8 D01B            BNE EXIT       ;YES.
02DA A5E2            LDA $E2        ;MAYBE ...
02DC C54C            CMP $4C        ;FOR SURE?
02DE B015            BCS EXIT       ;YES.
02E0 206DE0  SHOWME  JSR $E06D      ;LIST ONE LINE
02E3         ;
02E3         ;  SEE IF ROOM TO LIST ANOTHER LINE.
02E3 A525            LDA $25        ;CURRENT LINE ON SCREEN
02E5 18              CLC
02E6 6904            ADC #4         ;LEAVE ROOM FOR 4 LINES
02E8 C523            CMP $23        ;ROOM FOR ANOTHER LINE?
02EA 90E6            BCC ANYMOR     ;Y: GO TRY TO LIST MORE
02EC         ;
02EC         ;  SCREEN IS FULL. WAIT FOR A KEY.
02EC AD00C0  WAITKY  LDA $C000      ;SEE WHICH KEY PRESSED.
02EF 10FB            BPL WAITKY     ;NONE. KEEP ON WAITING.
02F1 C983            CMP #$83       ;WAS IT CTRL-C?
02F3 D0D7            BNE RESUME     ;N: DO ANOTHER PAGE.
02F5 2C10C0  EXIT    BIT $C010      ;RESET KEYBOARD.
02F8 4C03E0          JMP $E003      ;BACK TO BASIC.
02FB                 .END
```

©

The 25¢ Apple II Real Time Clock

Erann Gat
Oak Ridge, Tennessee

It is interesting to count the number of features of the Apple II which traditionally require boards full of parts to implement, but are done with only one or two inexpensive chips. For instance, the analog to digital conversion for the game paddles would normally cost at least $25, but is done on the Apple with a single inexpensive timer chip. The refresh for the dynamic memory requires no extra parts at all as this is done by the video circuitry.

This philosophy of doing things the easy way makes one wonder at the prices that are being charged for some of the peripheral boards for the Apple, particularly real time clocks. A search for an easier (and hopefully cheaper) way yielded a clock with good accuracy and any feature found on the more expensive boards, including many extra fringe benefits, with a total cost of between 3 to 25 cents depending on how sophisticated you want it to be.

All About Interrupts

Interrupts are something almost every computer hobbyist has heard of, but most of the information about them is rather cryptic. This section will attempt (note that verb) to clarify how interrupts work because they form the basis of the 25 cent clock.

Here is how an interrupt works: on the 6502 microprocessor there are two pins called IRQ and NMI. IRQ stands for Interrupt ReQuest and NMI stands for Non Maskable Interrupt. When either one of these pins is grounded, the processor finishes the machine language instruction it is currently working on, saves the program counter and processor status register onto the stack, (if you don't know what that means it isn't important) and jumps to a program somewhere in memory called an interrupt handling routine or interrupt handler. It then executes the interrupt handler until it encounters a RTI (ReTurn from Interrupt) instruction. It then restores the status register and program counter to their original values and continues executing the main program at the point where the interrupt occurred.

The main program is not affected by an interrupt except that some time is lost during the interrupt and the main program slows down. How much it slows depends on the length of the interrupt handler.

Now suppose that the interrupt handler was a routine that incremented a memory location and returned. This would then be an interrupt counter; i.e. every time an interrupt occurs, the counter is incremented. Now suppose that a pulse was applied to the interrupt line exactly once each second. Voila! A real time clock that tells time in seconds. This is the idea behind the 25 cent clock.

More About Interrupts

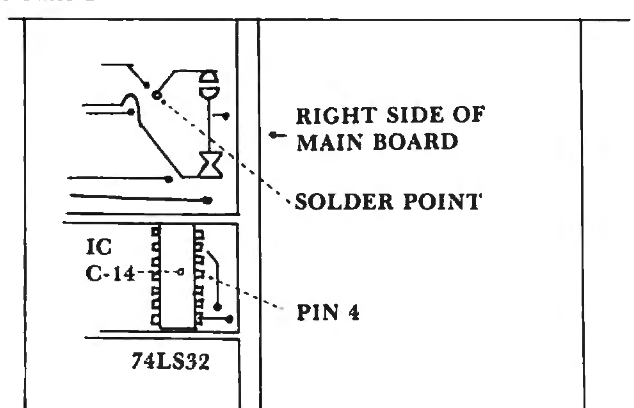
Up until now the 25 cent clock has been discussed in generalities and theories. This section discusses the actual implementation.

First some more facts about interrupts on the 6502: There are two main differences between the IRQ and NMI interrupts. In the 6502 status register there is a flag called interrupt enable. This flag can in effect turn off the IRQ line. If the enable flag is not set, the 6502 will deny Interrupt ReQuests. It will ignore them as if they were not there. On the other hand, NMI cannot be turned off. When a Non Maskable Interrupt occurs, the processor will always act on it and jump to the interrupt handler.

The second difference is that NMI and IRQ have their interrupt handlers at different places in memory. IRQ has another difference in that its interrupt handler is the same routine which handles the BRK instruction. BRK in effect generates a IRQ signal. There is a way to tell IRQ's from BRK's (in fact the Apple monitor does this for you) but this takes up quite a bit of time as well as creating other complications. NMI therefore is more suitable than IRQ for the clock. However, there is no law that says IRQ can't be used.

Next, a signal of known frequency must be found. A time base generator can be used, but at several dollars a piece it would be difficult to stay within the 25¢ budget. An ideal signal can be found in the video circuitry. This signal is the 60 Hz (meaning 60 times each second) pulse which generates the vertical retrace. This signal can be tapped at two locations shown in figure 1. The physical details are discussed in the next section.

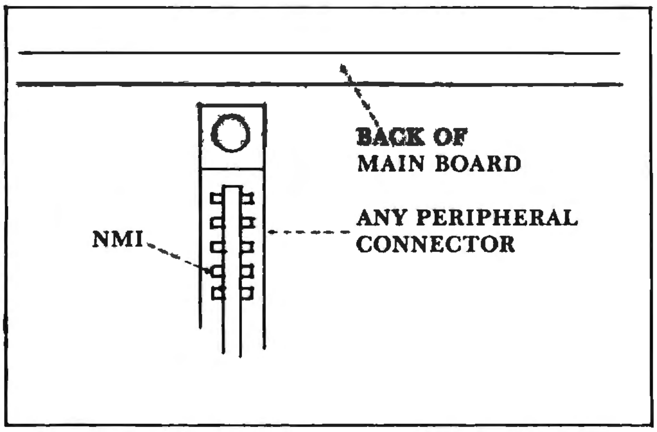
FIGURE 1



The Three Cent Clock

Implementing the clock in its simplest form involves simply connecting the NMI line to a signal source. On the Apple, the NMI line can be accessed from any of the peripheral slots on the rear of the board. The location of the NMI line is shown in figure 2. The connection can be made using a prototype board or by simply inserting a wire between the metal contact and the plastic housing of the connector.

FIGURE 2



The 60 Hz signal can be accessed in the two locations shown in figure 1. The first place is a small solder filled hole in the board. A wire may be soldered in the hole, or a wire wrap pin may be attached and connected to the NMI line via an alligator clip to make the clock removable. **NOTE; This may void your warranty. Check with your dealer!**

The other connection point does not involve soldering. To make the connection, carefully remove the IC at location C-14. The row and column numbers are marked on the board itself. Then insert a piece of very thin (wire wrap) wire into pin 4 of the socket. (See figure 1.) Now carefully reinsert the IC making sure it is oriented correctly and all the pins are securely seated in the socket.

Before this connection is made an interrupt driver must be entered into memory. If this is not done, the system will crash and RESET will have no effect until the connection is broken.

To get the three cent clock off to a flying start, enter the short program in listing 1. This can be done in the monitor or the mini-assembler. When the program is in memory, connect the interrupt line and watch the upper left hand corner of the screen. If everything was done correctly, the first character on the screen should start changing rapidly. What is happening is that sixty times a second the video circuitry generates a signal which is now being used to generate an interrupt. When an interrupt occurs, the processor starts executing the interrupt handler which is located at 3FB hexadecimal on an Apple. Usually the interrupt handler starts with a jump instruction since there are only five bytes of usable memory at

LISTING 1

```
*3FBL
03FB-    EE 00 04     INC    $0400
03FE-    40           RTI
03FF-    00           BRK
```

3FB, but since this program is so short it can be entered directly at 3FB. The interrupt handler that is now in memory simply increments a memory location and returns to the main program. This is a real time clock. It tells time in sixtieths of a second. Granted, it isn't very useful as it is now, but that will be fixed in a moment.

Now incrementing a memory location on the screen isn't very exciting, but try hitting a few keys. Surprise! They still work. In fact, everything works. Try dumping out some memory or printing something in basic. Everything will work normally and the first character on the screen will go right on counting. WARNING: the disk will **NOT** work. Neither will the tape. This is because the interrupts slow down the main program enough to upset the precise timing required by the disk and tape routines. Having the interrupt connected will also make the bell tone sound peculiar.

To make the clock more useful, enter the three programs in listing 2. The first program is simply a jump instruction to the second program which is a clock routine to drive an hour-minute-second clock. The third program is a basic routine which sets the clock and outputs the time of day. The programs are thoroughly documented so they won't be discussed here.

LISTING 2
PROGRAM #1

```
*3FBL
03FB-    4C 00 03     JMP    $0300
03FE-    00           BRK
03FF-    00           BRK
```

Making It Better or When Is An NMI Really An IRQ?

It should be clear by now that the power of the clock lies in the interrupt driver program, but there are some hardware enhancements that can be made. These extra features will roll the price up to a respectable 25 cents (more or less).

The first add-on is a sophicitcated piece of hardware called a switch. This is used to make easier the task of turning the interrupts on and off. The switch is installed so that it breaks the connection from the 60 Hz signal. Personal experience has shown that flipping a switch makes a more dignified display than pulling a wire in and out.

The second modification is a bit more complicated. (Seriously.) This modification allows the computer to control the interrupts via one of the annunciator outputs on the game I/O connector. The

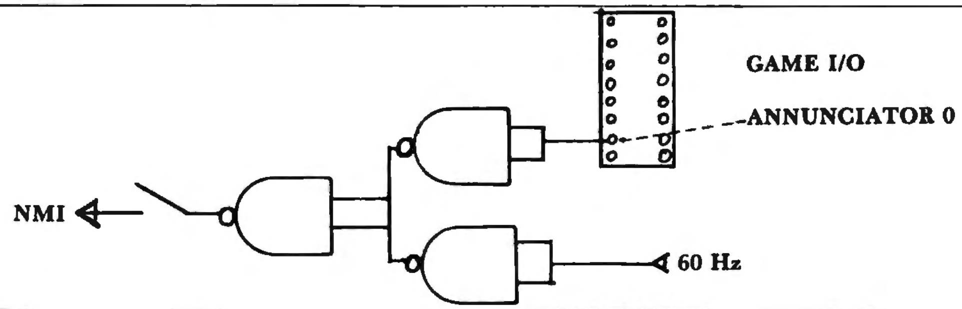
LISTING 2
PROGRAM #2

```
*300LL              CLOCK
0300-   85 05       STA   $05       SAVE A AND X
0302-   86 06       STX   $06
0304-   A9 3C       LDA   #$3C      A=60 DECIMAL   X=0
0306-   A2 00       LDX   #$00
0308-   E6 04       INC   $04       COUNT 1/60 SECOND
030A-   C5 04       CMP   $04        FULL SECOND YET?
030C-   D0 22       BNE   $0330       IF NO THEN RESTORE REGISTERS & RETURN
030E-   86 04       STX   $04       RESET 1/60 SECONDS
0310-   E6 03       INC   $03       COUNT 1 SECOND
0312-   C5 03       CMP   $03        1 MINUTE YET?
0314-   D0 1A       BNE   $0330
0316-   86 03       STX   $03
0318-   E6 02       INC   $02       MINUTES
031A-   C5 02       CMP   $02
031C-   D0 12       BNE   $0330
031E-   86 02       STX   $02
0320-   A9 0D       LDA   #$0D      SET A=# HOURS IN 1 DAY PLUS 1
0322-   E6 01       INC   $01       HOURS
0324-   C5 01       CMP   $01       FULL DAY?
0326-   D0 08       BNE   $0330
0328-   E8          INX             IF YES SET HOURS TO 1
0329-   86 01       STX   $01
032B-   A5 05       LDA   $05       RESTORE REGISTERS
032D-   A6 06       LDX   $06
032F-   40          RTI
0330-   A5 05       LDA   $05       RESTORE THEM HERE TOO
0332-   A6 06       LDX   $06
0334-   40          RTI
0335-   00          BRK
0336-   00          BRK
0337-   00          BRK
0338-   00          BRK
0339-   00          BRK
```

LISTING 2
PROGRAM #3

```
>LIST CLOCK DRIVER
  5 PRINT CHR$(4);"BLOAD CLOCK"
  7 POKE 1020,0: POKE 1021,3: REM  SET INTERRUPT
    VECTOR
 10 INPUT "INPUT TIME -->",H,M,S
 15 REM  SET CLOCK
 20 POKE 1,H
 30 POKE 2,M
 40 POKE 3,S
 43 POKE 4,0
 45 A= PEEK (-16296): REM  TURN CLOCK ON
 47 INPUT "12 OR 24 HOUR CLOCK",A: POKE 801,A+1
 48 REM  SEE LISTING  FOR EXPLAINATION OF LINE 47
 50 CALL -936: REM  CLEAR SCREEN
 60 VTAB 10: PRINT "
 61 REM  ERASE OLD TIME
 70 VTAB 10: TAB 10
 75 REM  DISPLAY CURRENT TIME
 80 PRINT PEEK (1);":";
 81 REM  HOURS
 90 IF PEEK (2)<10 THEN PRINT "0";: PRINT PEEK (2),
 91 REM  MINUTES
100 PRINT PEEK (3), PEEK (4): GOTO 60
110 REM  SECONDS AND 1/60 SECONDS
```

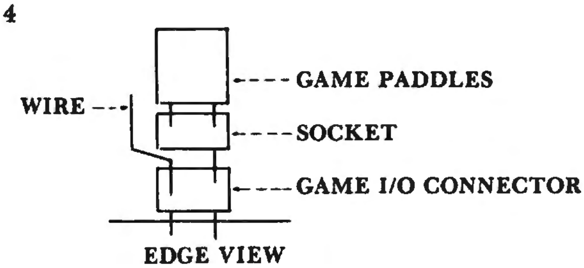
FIGURE 3



only extra part required is a 7400 or 74LS00 nand gate. It is wired according to figure 3 using a prototype board, an off-board wire wrap socket, or the breadboard area on the Apple board.Even the revision 1 boards have room for two IC's in the right hand corner under the keyboard. NOTE: To wire the modification in this way requires removal of the Apple board and will probably void your warranty. Check with your local dealer.

The connection to the game I/O connector is made using a piece of stiff wire such as the lead of a small resistor. This wire is inserted into the connector and bent as shown in figure 4. A 16 pin IC socket with one pin clipped to accommodate the wire is inserted over that and the game paddles are plugged into that socket. Many connections can be made to the game connector in this manner without having to clip pins off of the game paddles.

FIGURE 4



The Disadvantages

Unfortunately, every silver lining comes equipped with a cloud and the 25 cent clock is no exception. The main problem is that the disk and tape will not work, as well as other programs which involve precise timing. The interrupts must be disabled, either manually or under program control, while such programs are running.

Another hitch is in the computer control circuit itself. When an Apple is turned on, the annunciator outputs are high (logic 1) so this has been made to disable the interrupts. An autostart rom however, turns all the annunciators to logic 0. Before this happens all the annunciators are still at logic 1 for a few milliseconds so inverting the signal from the annunciator will still leave the interrupts enabled for enough time to cause an interrupt and a system crash. Therefore, the interrupts must be disabled manually upon power up with an autostart rom.

Another problem is that the bell tone sounds raspy. This isn't serious, but it can get on your nerves after a while. It doesn't make a good way to check if interrupts are enabled.

The final problem is that the clock seems to lose about ten seconds each hour. This can be remedied by adding ten seconds to the seconds counter each hour.

Fringe Benefits

The 25 cent clock is remarkably user proof. The NMI line doesn't require debouncing, and resetting the comupter doesn't interfere with its operation either (unless the reset key is held down for a long time).

The two main dangers of system crashes are working on the interrupt handler while interrupts are enabled, and not saving registers. THIS IS IMPORTANT!!! You must save each register you intend to modify. If you do not you will get very mysterious results. You can save registers in memory or you can push them onto the stack. There is also a routine to save and restore all registers in the monitor.

Once these restrictions have been met, the 25 cent clock opens a vast new horizon of features that would cost tens of dollars if bought from vendors. The price you pay is speed. The longer the interrupt routine, the slower the computer runs. This is not a severe handicap. The clock routine does not slow the computer down enough to be perceived, even when the interrupts are switched on and off for comparison. In order to slow the computer down by even one percent it requires a one hundred instruction routine.

Some things that can be done include:

Control Of Computer Speed Using Game Paddles: have the interrupt driver pause according to the position of a game paddle to give control of listing speed, how fast a program runs, etc.

Keyboard Buffering: have the interrupt routine sample the keyboard and store any keypresses in a buffer to give storage of multiple keypresses while something else is going on.

Mixing Display Modes: sixty times a second switch to another display mode to mix text and graphics, or mix two graphics modes for extra colors.

The possibilities are endless. You can even run two programs at once using the interrupt. The twenty-five cent Apple II real time clock is a lot more than just a clock, it's a cheap way of doing a lot of expensive things, right in line with Apple tradition.

GLOSSARY

INCREMENT- to add 1 to a counter

INTERRUPT HANDLER- a machine language program which is executed whenever an interrupt occurs

INTERRUPT VECTOR- the address of the interrupt handler routine

IRQ- Interrupt ReQuest; an interrupt line which can be disabled under program control

NMI- Non Makable Interrupt; interrupt line which cannot be disabled

REAL TIME CLOCK- a device which provides a computer with information about the time without disrupting the computer's normal functions

©

THE ATARI® GAZETTE

Ticker Tape Atari Messages

Eric Martell and Chris Murdock
The Education Connection
Boulder, Colorado

The large text modes [GR. 1, GR.2] are very convenient. With text like this available, the Atari can become a useful and eye catching message presentation device. The following program makes use of some simple string manipulations, to move text across the screen in a manner reminescent of ticker tape or a marquee sign. The actual text movement is done by line 50 in the following manner:

The first 19 characters of the message string [A$] are printed at position 1.5 [the vertical center of the screen]. A temporary string [C$] is set equal to the second through the 20th characters in A$. Then A$ is added [concatenated] to C$. Since C$ and A$ are dimensioned to be the same length, this has the effect of attaching the first character in A$ to the end of C$. A$ is then set equal to C$ and printed once again.

The variable K is set up to check for any key being pressed. This action will terminate the program in line 55. A delay loop is inserted in line 55 to increase readability, since the string manipulation is so fast that the letters become blurred unless slowed down.

The rest of the program contains enough remarks to be self explanatory.

The Ticker Tape Program

```
0 REM MOVING MESSAGE PROGRAM FOR THE
ATARI
1 ? "esc-shift-clear":REM CLEAR SCREEN BEFORE
GOING ON
9 REM DIMENSION STRINGS
10 DIM X$[1000],B$[1],W$[20],P$[20],Y$[20],Z$[20]
15 W$ = "                    ":REM 20 SPACES
19 REM CLEAR STRINGS AND SET B$ = BLANK FOR
    CLEARING THE REMAINDER OF X$
20 X$ = "":B = " "
24 REM INPUT YOUR TEXT HERE
25 ?:? "ENTER YOUR MESSAGE";:INPUT X$
29 REM CLEAR THE REST OF X$ IF SHORTER THAN
    SCREEN WIDTH [19]
30 IF LEN[X$]<20 THEN FOR C = 1 TO 20-LEN[X$]:
  X$[LEN[X$] + 1] = B$:NEXT C:X$[LEN[X$] + 1] = B$
35 DIM A$[LEN[X$]],C$[LEN[X$]]:A$ = X$
39 REM GOTO GRAPHICS MODE 2 + 16 AND PRINT
    STRINGS
40 GRAPHICS 18
45 REM MOVE BORDERS OF STARS
46 POS. 1,3:? #6;W$[1,19]:P$ = W$[2]:P$[LEN[P$] + 1] =
    W$:W$ = P$
47 POS. 1,7:? #6;Y$[1,19]:Z$ = Y$[2]:Z$[LEN[Z$] + 1] =
    Y$:Y$ = Z$
49 REM MOVE MESSAGE STRING AND CHECK
    LOCATION 764 TO SEE IF A KEY WAS STRUCK
50 POS. 1,5:? #6;A$[1,19]:C$ = A$[2]:C$[LEN[C$] + 1] =
    A$:A$ = C$:K = PEEK[764]
54 REM PAUSE TO INCREASE READABILITY, SET
    COLOR RANDOMLY, AND RESET ATTRACT FLAG
55 FOR TI = 1 TO 50:NEXT TI:POKE 77,0:SETCOLOR
    INT[RND[0]*4],INT[RND[0]*15],8:IF K = 255
    THEN 46
```

Additional Goodies

For those people who would like to discourage exit from their programs by means of the Break key or the System Reset key, here are three memory locations which can be poked to accomplish this task.

The **Break key** interrupt routine seems to begin and end in ROM, but is vulnerable when it passes through RAM. If you POKE 16,64 and POKE 53774,64 [this resets the Break key enable bit], you will find that the Break key will no longer respond until the locations are poked with 192, the program changes graphics modes, or the System Reset is pressed.

The **System Reset key** is not vectored through RAM until after it does a number of irreversable initializations and so is more or less impervious to attempts to disable it. However, the reset routine does look at a flag in location 580. If you POKE 580,1, or any non-zero integer, you can fool the computer into thinking that a System Reset impulse is a **cold start**. The major effect of this trick is to erase everything in RAM. Needless to say, having to reload a program once or twice is an effective deterrent to use of the System Reset key. ©

Atari Colors And Sounds With Paddles

Arthur Schreibman

The Atari computer has excellent graphics and sound capabilities. With 16 colors and eight levels of brightness we can generate 128 different colors. There are 256 notes available, each with 8 distortion values, totaling 2,048 sounds. Each color or sound can be accessed by a unique combination of numbers used in the SETCOLOR or SOUND statements. If you want to use a specific color or sound in your program, the problem is to find the correct values to use in the Basic statements.

The programs below enable you to see every color and hear almost every sound while also displaying the accompanying values used to generate them. These programs are also instructive in the use of the Atari paddles.

```
10 REM ATARI COLORS WITH PADDLES
20 GRAPHICS 3
30 POKE 752,1
40 COLOR 1
50 A = PADDLE (0)
60 B = PADDLE (1)
70 SETCOLOR 4, INT (A/15), 2* INT (B/30)
80 PRINT "COLOR = "; INT (A/15),
    "BRIGHTNESS = "; 2* INT (B/30); "  "
90 PRINT " ↑↑ "
100 GOTO 50
```

One paddle will change the screen color while the other changes the brightness. The numerical values used in the SETCOLOR statement are shown in the text window.

In the above program, line 30 surpresses the cursor. The two divisions in line 70 break the 228 positions of the paddle into 16 and 8 different positions, thereby using the full range of the paddles to display all 16 colors and 8 levels of brightness. The blank at the end of line 80 holds the space when the value changes from 2 digits to 1. Line 90 uses control characters to print line 80 in the text window only once. They are entered into the program by pressing the ESC key and then the CTRL key and ↑ key simultaneously. The last line sends the program back to line 50 where it waits for a change in the value of the paddle.

```
10 REM ATARI SOUNDS WITH PADDLES
20 N = INT (1.12 * PADDLE (0) )
30 D = 2 * INT (PADDLE (1)/30)
40 PRINT "NOTE = ";N;" DISTORTION = ";D
50 SOUND 0, N, D, 8
60 IF INT (1.12 * PADDLE (0) ) < > N THEN 20
70 IF 2 * INT (PADDLE (1)/30) < > D THEN 20
80 GOTO 60
```

In the above program, one paddle changes the notes while the other changes the distortion. The numerical values used in the SOUND statement are shown on the screen. The SYSTEM RESET key turns the sound off.

Since there are only 228 paddle positions and 256 notes, we cannot access every note with this method. The 1.12 factor in line 20 allows us to hear the full range of notes while skipping some notes along the way. Line 30 generates even numbers from 0 to 14 for the distortion value. Lines 60 and 70 wait for changes in the paddle values.

These two simple programs can be quite useful in the writing of other programs, and more fun than using trial and error to pick colors and sounds. ©

Atari As Terminal

A Short Communications Program

Henrique Veludo
N.Y.C., N.Y.

Here is a short, unsophisticated (it has no provisions for a printer, etc.) program to convert the ATARI into a terminal for communication over the telephone with a remote computer system such as the MICRONET data bank, using the ATARI modem and 850 Interface Module. After it is entered and RUN, it can be exited with the BREAK key (this will close all devices and reset parameters).

Lines 30-40 open the keyboard and RS232 devices.
Line 40 starts the Concurrent I/O Mode.
Line 50 gets characters from the keyboard and sends them.
Line 60 checks for an empty buffer.
Line 70 gets characters from the buffer and prints them.
Line 80 checks if a key has been pressed, and if so, directs program to send the character.

```
10 ? " }": POKE 82,0
20 OPEN #1,4,0,"K:
30 OPEN #2,13,0,"R:
40 XIO 40,#2,0,0,"R:
50 GET#1,A:PUT#2,A:POKE 764,255
60 STATUS#2,R:IF PEEK(747) = 0 THEN 80
70 GET#2,B:? CHR$(B);
80IF PEEK(764)< >255 THEN 50
90 GOTO 60
```

©

Character Generation on the Atari

Charles Brannon
Greensboro, N.C.

This article is a tutorial on a little-known feature of the Atari microcomputers -- the ability to re-define the character set. The **character set** is the group of 255 alphanumeric characters that can appear on the screen. It comprises the upper and lower case alphabet, the numbers, special symbols, and punctuation. Also included in the Atari character set are 29 "control graphics" characters. When the CTRL key is held down and a letter of the alphabet is typed, the corresponding graphics symbol is displayed. These symbols are much like those found on the PET. Unlike the PET, however, the Atari can re-define any of these characters. This allows custom graphics, user-defined special symbols (like pi, theta, or foreign language alphabets), and logos.

There is no built-in command to perform the changes; it has to be done the hard way with PEEK and POKE. These are commands to look at and modify memory, respectively. First of all, you must understand how the Atari stores and displays these characters. It is beneficial if you know how to work with binary numbers, but it is not a prerequisite.

Start out by designing your characters. Fill in the blocks on an 8x8 grid; each block will represent a pixel (picture element, dot). Observe the "A" in figure one. Notice the heavy vertical lines. A television screen will display horizontal lines brighter than vertical lines, so it is necessary to have two vertical lines in order for it to be clearly visible. Therefore, the "pi" in figure two may be hard to see unless enlarged in grapics mode 1 or 2.

Figure one

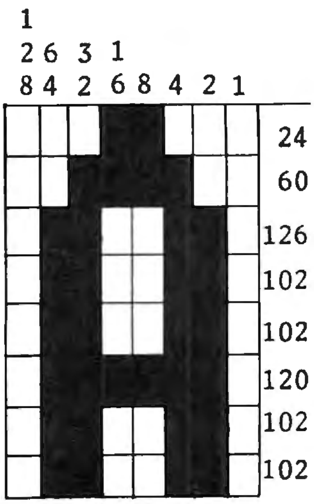
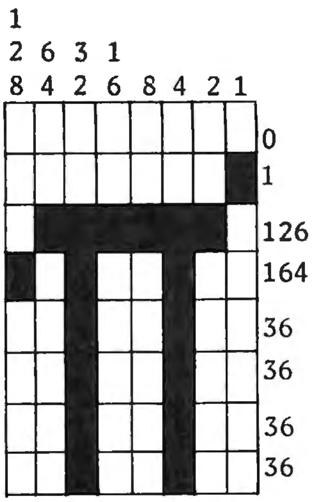


Figure two



After you have designed your characters, you have to convert them to the numbers that a computer loves. Each row in your grid represents a binary **byte.** A filled in block represents a 1 and a blank one means 0. Hence, the top row of the "A" is 00011000 or 24 decimal. Now write the bytes for each row. If you do not work with binary numbers, you can convert each line in the following manner:

1. Notice the numbers above each column. They are the powers of base two.
2. If a block is filled in, take the number above it and add it to a "Sum". Sum up all the blocks in the row. (e.g. the fourth line of the "pi" would be 128 + 32 + 4 = 164)
3. Do this for all eight rows.

Next, assemble the numbers into DATA statements. The numbers for "pi" would then look like this:

```
1000 DATA 0,1,126,164,36,36,36,36
```

Finally, you have your numbers. Now all you have to do is replace the numbers of the character you want to re-define with your numbers. Unfortunately, this table is stored in ROM, so it can not be altered. The solution is to copy this table into RAM memory, which can be changed, and then tell the computer where you have moved the characters to.

The first part of the program would then look like this:

```
10 ROM = 57344: REM START OF ROM CHARACTER
    TABLE
20 RAM = 8192 : REM HIGH UP IN MEMORY
30 FOR I = 0 TO 1023
40 POKE RAM + I, PEEK(ROM + I)
50 NEXT I
```

The transfer takes about 15 seconds, a seemingly LONG time. It need not be executed more than once, unless you go into a GAPHICS mode greater than 3.

The next line:

```
60 POKE 756,32 :REM 32*256 = 8192
```

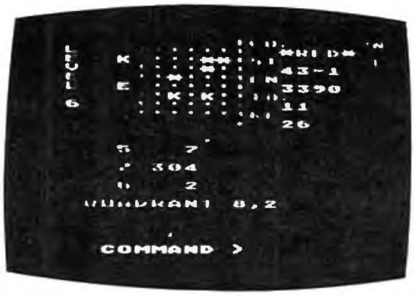
Now that the table is in RAM, we can now find the place in it for the new numbers. Look up the character you want to replace in table 9.6 -- Internal Character Set, on page 55 of the **Atari BASIC Reference Manual.** Write down this number as well. Notice that it is *not* the ATASCII value of the character. Include this number preceeding your eight bytes in the data statements. For our "pi":

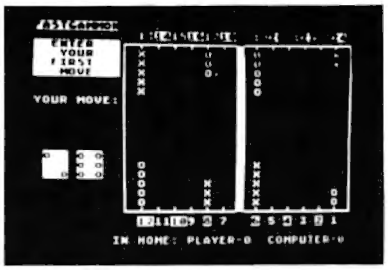
```
1000 DATA 32, 1,126,164,36,36,36,36
```

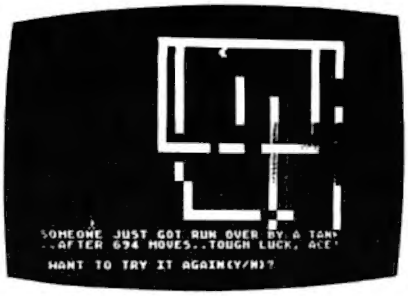
A few more lines, and the program is finished:

```
65 READ NCHR :REM NUMBER OF CHARACTER TO
    BE RE-DEFINED
70 FOR I = 1 TO NCHR
80 READ  RPLC: REM CHARACTER TO BE REPLACED
90 FOR J = 0 TO 7
100 READ A
110 POKE RAM + 8*RPLC + J, A
120 NEXT J
130 NEXT I
140 REM FOLLOWING LINE IS OPTIONAL
150 FOR I = 0 TO 255: PRINT CHR$(27);CHR$(I); :
    :NEXT I :REM DISPLAYS CHARACTERS
160 END
999 DATA 1 :REM NUMBER OF CHARACTERS TO BE
    RE-DEFINED
```

SOFTWARE FOR THE ATARI 800* AND THE ATARI 400*







TARI TREK™
By Fabio Ehrengruber

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

Cassette - $11.95 Diskette - $14.95

FASTGAMMON™
By Bob Christiansen

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

On cassette only - $19.95

TANK TRAP
By Don Ursem

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

Cassette - $11.95 Diskette - $14.95

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

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

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

On diskette only - $79.95

* * * * * * * * * *

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

On cassette only - $24.95

* * * * * * * * * *

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

Cassette - $11.95 Diskette - $14.95

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

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

*Indicates trademarks of Atari Inc.

A few program notes:

1. You can use multiple statements per line and squeeze the program into less memory; delete REMs if you like.
2. This program should be appropriately renumbered for use as a subroutine
3. VERY IMPORTANT:
   This is not the only way to accomplish the changes in the character set. Also see "Card Games in Graphics Modes 1 and 2" and the program on page 69 in COMPUTE!, November/December, 1980.
   E X P E R I M E N T !
4. Entering any GRAPHICS mode will reset the pointer to the table (line 60). Any time you enter a new text mode, re-execute POKE 756, 32.

The complete program, and a utility program that lets you look at characters wrap up this article. Study them, puzzle them out, and get down to business!

Happy POKEing!

Program to re-define characters

```
10 ROM=57344: REM START OF ROM CHARACTER TABLE
20 RAM=8192  :REM HIGH UP IN MEMORY
30 FOR I=0 TO 1023
40 POKE RAM+I, PEEK(ROM+I)
50 NEXT I
60 POKE 756,32 :REM 32*256=8192
65 READ NCHR :REM # OF CHARACTERS
70 FOR I=1 TO NCHR
80 READ RPLC :REM CHARACTER TO BE REPLACED
90 FOR J=0 TO 7
100 READ A
110 POKE RAM+8*RPLC+J, A
120 NEXT J
130 NEXT I
140 FOR I=0 TO 255:PRINT CHR$(27);CHR$(I);
   :NEXT I :REM DISPLAYS CHARACTERS
150 END
160 REM DATA FOR CHARACTERS FOLLOWS
READY.
```

Program to view characters in ROM. Note that the characters appear in GRAPHICS mode 4 !

```
10 GRAPHICS 4
20 SCR=PEEK(560)+256*PEEK(561)+4
30 SCR=PEEK(SCR)+256*PEEK(SCR+1)
40 PRINT "CHARACTER #? (0-127)";
50 INPUT CHR
60 IF CHR<0 OR CHR>127 THEN 40
70 PRINT #6;CHR$(125);
80 FOR I=0 TO 7
90 POKE SCR+4+10*I, PEEK(57344+CHR*8+I)
100 NEXT I
110 GOTO 40
READY.
```

©

The Atari Hall Of Fame: Iridis, Founding Member

Craig Patchett
Greenwich, CT

Having followed the evolution of TRS-80 software quality from poor to not-so-bad, I expected to have to go through the same evolution when I upgraded to an Atari 800. The people at **The Code Works** have proven me wrong. They publish an ongoing "Atari Tutorial" called **IRIDIS** which, in this reviewers opinion, is quickly bound for Atari stardom.

I tried numerous times to write this review to cover everything about IRIDIS that I thought deserved to be covered. Each time I thought of more things that I should have included. Eventually, out of desperation, I ended up making this outline so that I wouldn't forget anything. Then I thought, "aha(!), they (as in you) don't want to wade through unecessary verbosity (what you're reading now), so I'll just give them my outline." So, without any further unecessary ado (what you're reading now), here's that outline:

Iridis

Details: 2-4 programs each 'issue" (so far).

16K needed for each program for cassette, 24K for disk.

Each issue consists of one cassette/disk and 1 user's guide.

A User's What?

The **User's Guide** is a booklet (32-56 pages) containing:

Listings of each program.

Complete explanations of each listing, including an explanation of *every* line and *every* variable, and averaging 3½ pages long in the first issue, *ten* in the second. And ten very understandable pages at that!

Hacker's Delight: explanations of various Atari mysteries, such as display lists and 23 very interesting memory locations in the midst of the Atari memory jungle.

Novice Notes: for those of you who thought "so what?" or "huh?" to the description of **Hacker's Delight, Novicer Notes** explains, in very simple terms, such things as bit patterns (3½ pages) and

string manipulation (2½ pages).
Oddments: "Facts, Fancies, and Rumors."
The Oracle: questions to the editor.

So what are these programs, anyway?

IRIDIS 1:
CLOCK: a clock with hands, and ticking, and chimes, and everything!
ZAP: a one player "chase" game.
LOGO: an interesting demo program.
POLYGONS: an even more interesting demo (remember *Spirograph*? Well...)
IRIDIS 2:
FONTEDIT: design your own Atari character sets with this feature packed character set (or "font") editor. Work with an 8X enlargement of a character, and see it in it's regular size at the same time. Fonts can be saved to tape or disk, and can be used in your own programs using an included BASIC subroutine.
KNOTWORK: an interactive demo program involving "celtic interlace" and using a custom designed font. This one tends to defy a simple description, so I'll leave it as a (pleasant) surprise.

Anything else?

For those people sick of sending away for programs and then having to wait for weeks before finally receiving them, you might be pleased to know that I mail ordered both **IRIDIS'** and received them both in about a week! Keep up the good work **Code Works.**

What's your point? (As if I haven't already guessed)

Buy these programs. **IRIDIS 2** should be as much a part of your programming library as your BASIC cartridge is. **IRIDIS 1** also contains some valuable programming techniques that can be adapted easily to your own programs, and the programs are fun to use, besides. Although I realize it is impossible to completely convey my own admiration of **IRIDIS** in the length of this review, take it from an old hand; **IRIDIS** is, and promises to be in the future, one of the major works ever to be published for the Atari 800. ©

Review

Atari Music Composer

Jerry White

Atari owners with an ear for music will love the Atari Music Composer. It is as much fun as it is educational. There's something fascinating about hearing music and seeing it displayed in music form at the same time. After a little experimentation, you will find the creative possibilities endless.

At first you may be awed by the twenty page manual. Relax! To get started, you need only read thru pages 3 thru 13. Part 1 is a general description that explains all your options and commands in detail. There is a great deal of data on these six pages. Don't try to memorize it, just read thru it. Part 2 is a sample session where you actually create the song Row, Row, Row Your Boat. It is very well written and easy to follow the step by step instructions. Once you finish this part, go back and reread part 1. Now it will be easier to digest since you are reading it for the second time and have used many of the commands. By now about an hour has passed and you are ready to enter a song from your human memory or copy one from sheet music. Go to pages 19 & 20 in your manual. Here you will find a Quick Guide of all the commands. Use it as reference.

Allow me to give you some hints that will be quite helpful. Remember that a phrase is a section of music. There are four voices as in the Atari Basic Sound command. However, in the Music Composer, they are numbered 1 thru 4 instead of 0 thru 3. These voices are preset so that each has a Play command. Voice 1 is set to Play Phrase 1, Voice 2 is set to Play Phrase 2, and so on. Let's assume you have just created a one voice song consisting of two phrases. Assume you have Arranged Voice 1 to Display, Play Phrase 1, and Play Phrase 2. Now you want to Save your song on tape or diskette. Don't save it yet. Since Voice 2 was preset to Play Phrase 2, you will have Voice 1 playing one section of your song while Voice 2 is playing the other section. That will probably sound terrible since you did not create these two phrases as harmony. The thing to do is to change each of the preset Voice 2, 3, and 4 commands to Play Phrase 9. Since you have no phrase 9, those Voices will remain silent.

Sooner or later you will add harmony voices. When you do, all voices will have to be syncronized. You may want Voice 1 to Play Phrase 1 while Voice 2 plays phrase 3 and Voice 3 plays phrase 5 and Voice 4 plays phrase 7. That may sound difficult to you but your Atari computer will understand it. A problem may arise when you try to coordinate the four voices. For this reason, you should use the Check Measures option. This will tell the computer to check the length of each measure as it is entered. You will be notified if a measure is too long or too short. Since the measures are counted and numbered, it shouldn't be difficult to track down a problem as long as your measures are correct length.

When you save your music files on tape or diskette, I suggest you use the extension .MUS for music files and .HAR for music files including harmony. This will make it easy to tell music files from programs.

For those who do some programming, Part 3 of the manual explains the music file structure and supplies three Basic program listings. Alas, here the manual is flawed. The first program is the only one documented. It dumps music files onto the screen. It works if you leave out line 80. I believe that was meant to be a REM statement. As written, the other two programs were meant to be used only as guidelines to the experienced programmer.

To those of you who decide to key in the harmony program, you will need a disk system and over 32K. It will run on a 32K system if you change line 5 to NN = 180. NN is used to dimension many arrays. It is the number of notes the program can handle. You will have to make NN only as large as the music file it must read. Therefore it can run on less than 32K if it is to create harmony to a short song.

Enjoy the Atari Music Composer. Good luck and good music. ©

COMPUTE needs you! Let us know what interesting applications you're coming up with for your Atari.

Star Fleet to All Cruisers...

Who's the current champ of Star Raiders? Send in your best score, comments and playing strategies to COMPUTE, P.O. Box 5406, Greensboro, NC 27403.
Attn: "Atari Gazette"

Put A Printer On The Atari Ports

C. Kingston
White Plains, N.Y.

In order to use a printer other than the two 40 column models that plug directly into the serial port on the Atari, the Atari owner must either buy an Interface Module or find some alternate method of communicating with the printer. A suspicion that a method of using the joystick ports for general I/O purposes might be found was partially confirmed when a commercial cable and program became available to drive a printer through joystick ports 3 and 4. Although getting the commercial cable would be the easy (albiet expensive) way to proceed, I felt that more could be learned about the Atari by designing and building one. After a lot of digging or information on the Atari, the pieces fell together and resulted in a cable and program for the Atari that would run a printer operating out of the joystick ports. This article provides the necessary information so that the reader can construct a similar cable for a printer, or use the joystick ports for general I/O.

First, a little information about the Atari joystick ports. These use the two eight bit I/O ports of a 6520 PIA chip. Joystick ports 1 and 2 share one of the 6520's eight bit ports (Port A), and joystick ports 3 and 4 share the other 6520 port (Port B - this is the one we will use for the printer). Pins 1-4 of the 9-pin D connector of joystick port 3 are connected to bits 0-3 of PIA port B; pins 1-4 of joystick port 4 are connected to bits 4-7 of the same PIA port. Diagram 1 shows this arrangement.

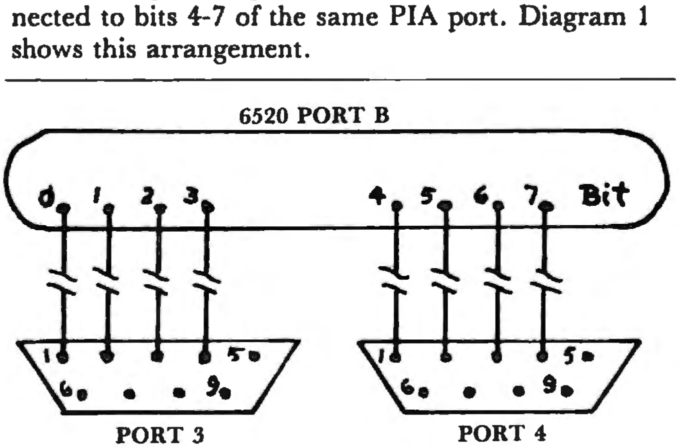


DIAGRAM 1

Port B is addressed by Atari locations $D301 and $D303. Port A is addressed by locations $D300 and $D302. Unfortunately, the conrol lines associated with these ports are apparently not available to the user. With this limitation, the joystick ports can easily be used for general I/O purposes. A 6520 port uses the two registers to control the specific function of the port bits. For port B, location $D303 is the control register, which we will call PCR. Location $D301 is the data or data direction register, which we will call DDR. Note that DDR has two functions. When it is functioning as a data direction register, it allows us to select which bits of the data are to be input and which are to be output. A 0 in the data direction register selects the input mode, while a 1 selects the output mode. When it is functioning as a data register, it inputs or outputs the appropriate data bits when connected to a peripheral device.

We select DDR as a direction register by setting bit 2 of PCR to 0; we select DDR as a data register by setting bit 2 of PCR to 1. So the sequence for setting Port B up as an output port is as follows:

1. Put $30 in PCR (Make DDR a direction register)
2. Put $FF in DDR (Make all bits output)
3. Put $34 in PCR (Make DDR a data register)

Note that $30, rather than $00, is used as the base byte or PCR. This is to maintain the normal operating mode of the Atari, which presumably uses the control lines for purposes other than the ports (the bits other than bit 2 are used for other control purposes). If you wanted to make the port an input port, which it is for the joysticks, put $00 in DDR in step 2. Specific bits can be made either input or output by making the associated direction bit a 0 or 1 respectively in DDR in step 2. Note that the bits are pulled to +5 volts when set for input. Leventhal's book (6502 Assembly Language Programming) has instructions and several examples on using the 6520 chip (in Chapter 11), and the reader is referred there, or to specification sheets, for further information on the operation of the 6520 PIA. Pin 6 of each joystick port is connected to the joystick trigger. The trigger or port 4 is read at location $D013. Only the least significant bit is used, so the value is either 1 (trigger not pressed-line pulled high) or 0 (trigger pressed-line grounded). We will use this for handshaking.

The plan of action begins to become clear -- or does it? We simply connect Port A to the printer and connect the trigger pin to the outgoing Busy line on the printer. Then we'll connect the Strobe pulse to, uh. There's the rub; we don't have an extra output line available in joystick ports 3 or 4. We could bring another joystick port into action, but this would be wasteful. Well, what about bit 7, which is only used for parity or special purposes. If we can get along without it, then we can use it for the strobe, and indeed, this is what we'll do. It must be kept in mind that special operations of the printer that may use bit 7 cannot be invoked if we do this.

Now a direct connection between the Atari and the printer would seem to be acceptable. This may be the case if twisted pair cable is used and good grounding practice is followed. I have been using a direct connection off of the KIM-1 application port to drive a printer for some time (the PIA is not a 6520 though). But it appears not to be acceptable if only two or three ground connections are used, which keeps the cable reasonably simple, and the printer uses pull-up resistors for the input lines. My guess is that the 6520 cannot sink enough current to, drop the lines to a respectable level for a 0, thus leaving them near the transition voltage. Any induced hum or noise can then cause a fluctuation between 0 and 1 on the lines. And induced hum or noise can then cause a fluctuation between 0 and 1 on the lines. And indeed, a direct hookup produced a machine gun like output of the same letter as the strobe line was apparently bounced up and down by 60 cycle hum. One answer to this is to use buffer chips or transistors to adequately drop the lines for a 0 output. Because of their availability, inexpensive PNP transistors (2N3906 or 2N5139, etc.) were chosen. Diagram 2 illustrates the complete cable.

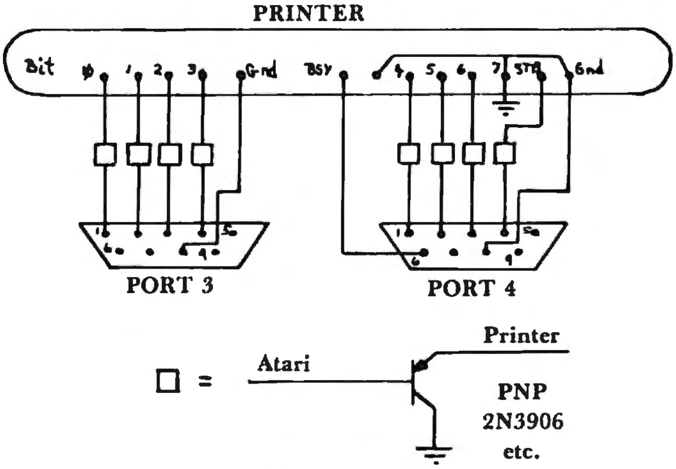


DIAGRAM 2

Note that the connections to the transistors from the Atari ports must be as short as possible. The connections on the one I built are about one inch long, and the transistor board sets on the table under the ports. Also note that bit 7 on the printer must be tied to ground. The entire cable should not cost more than $25, probably much less with careful mail order shopping.

A simple software driver applicable to any 6502 based computer could be used to drive the printer. However, if we want to take advantage of the Atari's flexibile I/O system, the program must be written specifically for this purpose. The program in this article was written so that it will hook into the operating system (OS) and operate in place of the normal OS printer subroutines. It is located at $067A-$06FE, which is an area that will presumably be left alone by Atari software so that it will remain available for users' programs.

The Atari controls I/O by means of a set of routines in the Central I/O Utility (CIO). Almost all I/O calls go through the CIO, which is why the Atari has such flexibility in its handling of I/O. A section of the OS ROM is dedicated to the routines which perform the I/O operations. These routines are called through I/O Control Blocks (IOCBs), which in turn transfer operation to the required routine segment (Handler) by using a vector table (Handler Vector Table). (There are eight IOCBs, and thus the Atari can have eight active I/O devices at any one time.) The key to the use of the handlers is the Device Table, which is transferred from ROM to RAM on system initialization. This table contains an identifying letter for each device along with the address of its handler vector table. We can therefore change the address in the Device Table to point to our own handler vector table, which we can set up in RAM. The program in this article sets up a printer handler vector table at locations $680-$68E, which points to the handlers starting at $690. Note that the vectors point to the handler routine address minus one. The vector table address in the device table for the printer (located at $31B-$31C) is changed to point to our handler vector table. The IOCB set up for the printer therefore directs the program to one of our handlers rather than to the Atari OS handlers for printer operation.

One problem is encountered in such an arrangement: the device table is re-initialized whenever the system reset button is pushed. Unless we can put our handler vector table address back into the device table at that time, we would have to do a separate re-initialization step. Fortunately the OS system reset sequence uses certain page zero vector locations for initialization purposes. One is for cassette operation initialization (CASINI at $0002-$0003), and one is for disk operation (DOSINI at $000C-$000D). Depending upon which we are using (disk or cassette) we can set this vector to point to a short routine that re-establishes our handler vector table address. By also transferring the original content of the page zero initialization vector, we can then send the program off to do whatever it was originally supposed to do so that everything will operate properly. The following brief description of the operation of the parallel printer handler shows how these facts are incorporated into the program.

The six bytes at $067A-$067F control the driver's hookup to the OS after a system reset. The bytes labeled LO and HI are used to store the initialization entry location (for the program - or cartridge - that will use the printer driver), which is read from $000C-$000D (DOSINI) during initialization. The segment in $0680-$068E is the handler vector table that points to the appropriate subroutine in the driver (address-1). The byte of $068F is used as a counter for the line length. The subroutine 'OPEN' sets up the 6520 PIA port B as an output port. The subroutine 'WRITE' is the actual printer driver. The byte at $06D2 determines the line length, and is set to the desired number of characters per line plus one. As written, the program is set for a line length of 78; the byte is set to 79 ($4F). It can be set for any line length up to 254.

The printer driver looks for the Atari code for RETURN, which is $9B, and converts it to the ASCII code of $0D. This is the only ATASCII (Atari ASCII) code that is decoded by the driver. The ATASCII and ASCII codes for letters, numbers, and most punctuation and symbols are the same, and other conversions do not seem necessary. The driver assumes that the strobe is high to low. If your printer strobes from low to high, change the following:

```
06B9  A0 00   0033   LDY  #$00
06BB  29 FF   0034   AND  #$7F
06C0  09 80   0036   ORA  #$80
```

If your printer automatically outputs a line feed after a carriage return, change $06D7 from $0A to $00.

The segment at BINIT is the initialization subroutine. This sets $000C-$000D (DOSINI) to point to the handler setup subroutine, and puts the original content of DOSINI in LO-HI. If you are not using DOS, then change $06E3 and $06EF from $0C to $02, and $06E8 and $06F3 from $0D to $03. This sets the program up for cassette operation and initialization. If you are using the driver with BASIC, you can initialize it by using the USR instruction pointing to BINIT (1761 decimal). This supplies the necessary PLA command for the USR instruction. If you initialize it from a machine language program, do a JSR to INIT ($06E2). If you are using a disk to load the printer routines, wait until the disk drive shuts off before initializing the driver. For some reason that I have not tracked down, initializing the driver while the disk drive is running seems to inhibit it from turning off. There is no problem here once the printer driver initialization is complete.

The segment HANFX is the one that re-establishes contact with the OS. This is run during initialization, and is called after a system reset. The only way to remove the driver from operation is to turn the computer off or change DOSINI ($000C-D), or CASINI ($0002-3) for cassette operation, back to the values in LO-HI.

Once initialized, the driver will operate with all BASIC commands that drive the regular printer routines. It will also work with all machine language programs that use the I/O control blocks to drive the printer routines. You may have to clear the printer and return the carriage to the left by outputting a RETURN (using the command 'LPRINT' in BASIC) after initialization. This will depend upon the particular printer that you are using. If you write a machine language program that outputs to a printer, it will interface to either the Atari OS handlers or the one here if you go through the IOCBs. However, using the IOCBs requires a bit of programming to set up the proper parameters. It is simpler to directly use the driver routines without going through the CIO. This seems to be what is generally done on most other microcomputers. In that case however, the program will not operate a printer connected to the serial port.

To use the parallel handler directly in a machine language program, the handler program must be loaded into $67A-$6FE. Then it must be initialized by a JSR INIT. This locks it into the system. Before using the printer, the port must be initialized by a JSR OPEN. Then each character to be printed is placed in the accumulator (A) followed by a JSR WRITE. At the completion of the printed material, do a JSR CLOSE (this only puts out a CR, and may not be necessary depending upon the printer used and the program). A skeleton program would look like this:

```
START  JSR INIT
       JSR OPEN
MAIN   ...
       ...
       LDA CHAR
       JSR WRITE
       ...
END    JSR CLOSE
       JMP EXIT
```

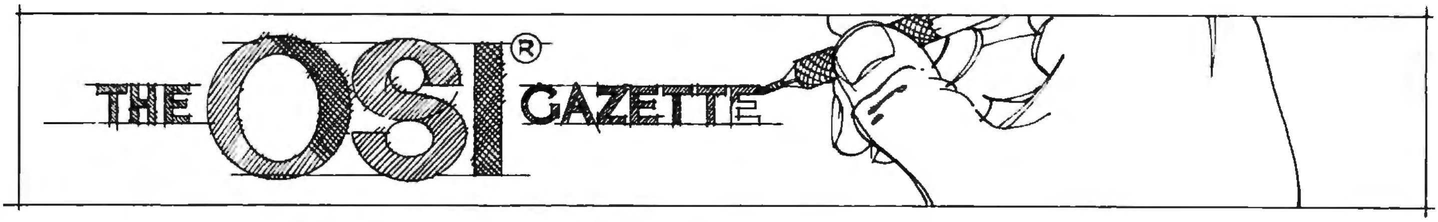
You must be careful in assuming that a machine language program that supports printer output uses the I/O Control Blocks (IOCB's). For instance, the driver was written using the assembler for the Atari by Quality Software. This program does support printer output, but it does not use the IOCB's completely. The actual output that sends the character for printing calls the Atari WRITE handler directly. The calling address must be changed in such a case to point to the WRITE handler in this program.

There is no reason that the joystick ports cannot be used as pseudo RS-232 ports as well, and thus for printers or other peripherals that require serial I/O. I expect to be writing a program for this in the near future in order to connect a digital input pad. One problem in the Atari for this may be the use of interrupt processing subroutines by the OS; these may throw off any timing loops used for serial control. This might force one to inhibit the interrupts, or to use the timers in the Atari for timing control. Who knows, maybe the Interface Module isn't really necessary for flexible I/O with the Atari.

```
                0002        PON
                0003        ORG   $67A
                0004 *    PRINTER DRIVER
                0005 *    C. KINGSTON (1980)
067A 20 F4 06   0006 REENT  JSR   HANFX
067D 4C         0007        HEX   4C
                0008 LO     DS    1
                0009 HI     DS    1
0680 8F 06 A8
     06 DE 06   0010 HANTABHEX    8F06A806DE06
0686 A8 06 DE
     06 DE 06   0011        HEX   A806DE06DE06
068C 4C 78 EE   0012        HEX   4C78EE
                0013 CTR    DS    1
0690 A9 30      0014 OPEN   LDA   #$30
0692 8D 03 D3   0015        STA   $D303
0695 A9 FF      0016        LDA   #$FF
0697 8D 01 D3   0017        STA   $D301
069A A9 34      0018        LDA   #$34
069C 8D 03 D3   0019        STA   $D303
069F A9 80      0020        LDA   #$80
06A1 8D 01 D3   0021        STA   $D301
06A4 A0 01      0022 ALCLO  LDY   #$01
06A6 60         0023        RTS
06A7 A9 0D      0024 CLOSE  LDA   #$0D
06A9 C9 9B      0025 WRITE  CMP   #$9B
06AB D0 02      0026        BNE   PRT
06AD A9 0D      0027        LDA   #$0D
06AF A2 04      0028 PRT    LDX   #$04
06B1 AC 13 D0   0029 BSY    LDY   $D013
06B4 D0 F9      0030        BNE   PRT
06B6 CA         0031        DEX
06B7 D0 F8      0032        BNE   BSY
06B9 A0 80      0033        LDY   #$80
06BB 09 80      0034        ORA   #$80
06BD 8D 01 D3   0035        STA   $D301
06C0 29 7F      0036        AND   #$7F
06C2 8D 01 D3   0037        STA   $D301
06C5 8C 01 D3   0038        STY   $D301
06C8 C9 0D      0039        CMP   #$0D
06CA D0 0E      0040        BNE   TEST
06CC A2 80      0041 DELAY  LDX   #$80
06CE CA         0042 DEL    DEX
06CF D0 FD      0043        BNE   DEL
06D1 A9 4F      0044        LDA   #$4F
06D3 8D 8F 06   0045        STA   CTR
06D6 A9 0A      0046        LDA   #$0A
06D8 D0 D5      0047        BNE   PRT
06DA CE 8F 06   0048 TEST   DEC   CTR
06DD F0 C8      0049        BEQ   CLOSE
06DF D0 C3      0050 BACK   BNE   ALCLO
06E1 68         0051 BINIT  PLA
06E2 A5 0C      0052 INIT   LDA   $0C
06E4 8D 7E 06   0053        STA   LO
06E7 A5 0D      0054        LDA   $0D
06E9 8D 7F 06   0055        STA   HI
06EC A9 7A      0056        LDA   #REENT
06EE 85 0C      0057        STA   $0C
06F0 A9 06      0058        LDA   #>REENT
06F2 85 0D      0059        STA   $0D
06F4 A9 80      0060 HANFX  LDA   #$80
06F6 8D 1B 03   0061        STA   $031B
06F9 A9 06      0062        LDA   #$06
06FB 8D 1C 03   0063        STA   $031C
06FE 60         0064        RTS

SYMBOL TABLE
  REENT 067A   LO    067E   HI    067F
  HANTAB0680   CTR   068F   OPEN  0690
  ALCLO 06A4   CLOSE 06A7   WRITE 06A9
  PRT   06AF   BSY   06B1   DELAY 06CC
  DEL   06CE   TEST  06DA   BACK  06DF
  BINIT 06E1   INIT  06E2   HANFX 06F4
```

©



Double-Density Graphing On The O.S.I. C1P

Gary Boden,
Narragarrsett, RI

When analyzing data or trying to understand an algebraic equation, a quick X,Y plot is often an invaluable first step. The O.S.I. C1P, despite its extensive graphics character set, doesn't particularly lend itself to graphing because of its cramped 24 x 24 video format.

One way to ease the crowding and double the resolution of a plot is to make use of graphics symbols that divide each square into quarters. I've written a program in BASIC that does this quickly and neatly (Fig. 1). As written it can stand by itself or, with slight modifications, function as a subroutine called up by a number-crunching main program. The effective display is increased to 40 x 40 bringing it into the range of usefulness for many scientific and business applications.

The program is designed first to scale the input data array, DA(I), between 0 and 40 (lines 5260 -5332) and put the results into array DY(I). Datasets with a difference between maximum and minimum values of more than 40 are condensed and smaller ones expanded on the graph. Both positive and negative values now will be plotted above the X-axis. Actual high (YH) and low (YL) values are saved and printed by line 5900 to give an idea of absolute as well as relative magnitudes. Next, 40 locations on the video display are computed for the scaled points. This must be done two points at a time because several symbols can be used to represent the pair, depending on whether they are equal, different by ± 1, or neither. Lines 5340 - 5780 code for the selection of the correct symbols. Figure 2 shows a decision tree that depicts how the choice is made. Since the first point of the pair automatically has an odd X-value (1,3,5,...39) and the second an even value (2,4,6,...40), only Y-values need be evaluated as odd or even. Based on the following table of possible X,Y coordinates, the correct quadrants are chosen for each square:

QUADRANT	STATUS	EXAMPLE
LEFT BOTTOM =	X ODD, Y ODD	(1,1)
LEFT TOP =	X ODD, Y EVEN	(1,2)
RIGHT BOTTOM =	X EVEN, Y ODD	(2,1)
RIGHT TOP =	X EVEN, Y EVEN	(2,2)

Line 5800 computes the video display locations rounded to the nearest integer.

The axes are drawn by lines 5210 - 5252. I also include a background grid (lines 5100 - 5130) to help read the plotted curve, but this may be deleted easily if not needed. The purpose of lines 6000 - 6030 is to check for a "return" that when found causes a recycling to the start.

None of this would be any good without a curve to plot. Line 210 is where the user enters his equation (or READ statement for data input) before running the program. DA(I), the data array, remains unaltered in case it is needed elsewhere. Any number of variables supplied by the user and/or program may be used so long as they are assigned before line 210. Figure 3 shows a graph of DA(I) = SIN(I/X)-COS(I/Y) where X = 3, Y = 6.3, and I goes from 1 to 40. Note that the scaled value of the 34th point is zero and that a blank spot is placed on the X-axis under the previous point.

The program occupies about 1700 bytes of RAM, but by dropping all the extras -- remarks, header, instructions, etc. -- it can fit into about 1 K of memory. Running time is around 8 seconds, much of it spent scaling and drawing the background; the curve plots out rapidly.

Four extensions of this routine come to mind which you may want to make to adapt it for your own purposes:

1. adjust axes to show negative plot quadrants
2. overplot more than one curve on the same graph
3. extend the X-axis with a second plot showing points 41 - 80
4. fill in below the curve to make a bar chart

In conclusion, this routine takes a big step toward relieving the C1P's small display problems when graphing. It is compact and quick, leaving plenty of memory to use for other things.

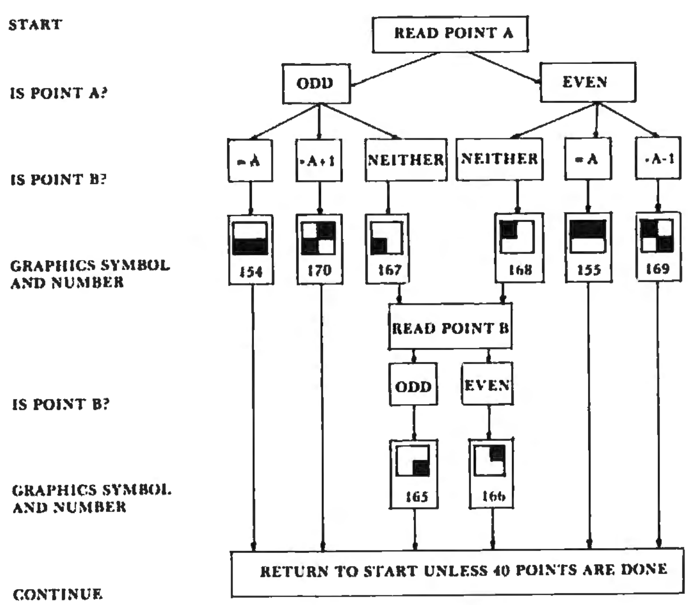


Fig. 2. Decision tree for selecting correct graphics symbols.

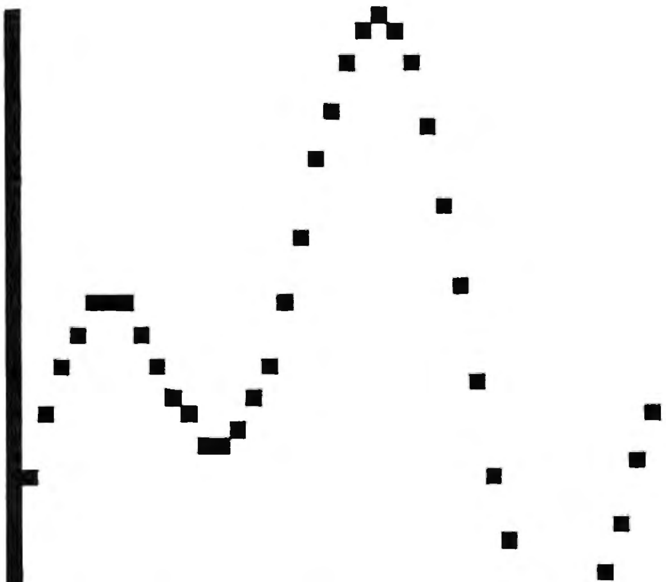


Figure 3.
Double-density Plot On The O.S.I. C1P.

```
10 REM--DOUBLE-DENSITY GRAPHING ON THE C1P
20 REM--BY G. BODEN; 9 DEC 1980
30 REM--ENTER YOUR EQUATION ON LINE 210
40 REM--THE Y-AXIS IS AUTOMATICALLY SCALED
        FROM 0 TO 40
50 REM--AND 40 POINTS ARE PLOTTED ON THE
        Y-AXIS
60 FORI=1TO25:PRINT:NEXT
65 PRINT"************************"
70 PRINT"DOUBLE-DENSITY PLOTTING"
72 PRINT"ON THE O.S.I. CHALLENGER"
74 PRINT"************************":PRINT
   :PRINT
80 PRINT"HIT 'RETURN' AFTER PLOT TO RECYCLE
        TO BEGINNING":PRINT:PRINT

100 DIMDA(50):DIMDX(50):DIMDY(50)
120 INPUT"ENTER X";X
130 INPUT"ENTER Y";Y
200 FORI=1TO40
210 DA(I)=SIN(I/X)-COS(I/Y)
220 NEXT

4900 REM--DRAW THE GRAPH AXES AND REFERENCE
          POINTS
5000 FORI=1TO25:PRINT:NEXT
5100 G2=53446:FORJ=1TO20
5120 FORI=1TO20
5125 POKEG2+I,207:NEXTI
5130 G2=G2+32:NEXTJ
5210 FORG=53446TO54054STEP32
5220 POKEG,157:NEXT
5230 G=54086:FORI=1TO20
5240 POKEG+1,155:NEXT
5250 POKEG,166
5252 T=177:POKEG+5,T:POKEG+10,T:POKEG+15,T
5255 REM--SCALE THE DATA
5260 YH=DA(1):YL=YH
5270 FORI=1TO40
5280 IFDA(I)>YH THENYH=DA(I)
5290 IFDA(I)<YL THENYL=DA(I)
5292 NEXT
5300 YR=YH-YL
5305 NF=0:IFYL<0 THENNF=ABS((YL/YR)*40)
5310 FORI=1TO40
5320 DX(I)=I
5330 DY(I)=INT((DA(I)/YR)*40+NF+.5)
5332 NEXT
5335 REM--PLOT OF CURVE
5340 I=1
5500 IFINT(DX(I)/2)*2>=DX(I) GOTO5670
5510 IFINT(DY(I)/2)*2=DY(I) GOTO5600
5520 IFDY(I)<>DY(I+1) GOTO5550
5530 SY=154:GOSUB5800
5540 GOTO5750
5550 IFDY(I)<>DY(I+1)-1 GOTO5580
5560 SY=170:GOSUB5800
5570 GOTO5750
5580 SY=167:GOSUB5800
5590 GOTO5670
5600 IFDY(I)<>DY(I+1) GOTO5630
5610 SY=155:GOSUB5800
5620 GOTO5750
5630 IFDY(I)<>DY(I+1) GOTO5660
5640 SY=169:GOSUB5800
5650 GOTO5750
5660 SY=168:GOSUB5800
5670 DY(I)=DY(I+1)
5680 IFINT(DY(I)/2)*2=DY(I) GOTO5710
5690 SY=165:GOSUB5800
5700 GOTO5750
5710 SY=166:GOSUB5800
5750 IFINT(DX(I)/2)*2>=DX(I) GOTO5770
5760 I=I+1
5770 I=I+1:IFI>40 GOTO5900
5780 GOTO5500
5800 POKEG+INT((DX(I)/2)+.5)-32*
     INT((DY(I)/2)+.5),SY:RETURN
5900 GOSUB6000
5910 PRINT"HI="YH;"LO="YL
5920 GOSUB6000
5930 GOTO120
5990 REM--LOOK FOR <CR>
6000 POKE530,1:K=57088
6010 POKEK,223:IFPEEK(K)=247THEN6030
6020 GOTO6010
6030 POKE530,0:RETURN
6040 END
```

©

A Small Operating System: OS65D The Kernel

Part 2 of 3

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

Subroutine Descriptions

Table 3 is a short memory map of the kernel. In this section we examine some of the subroutines in the map in more detail because they are either useful or interesting. The operating system input/output section will be discussed in some detail in another article, however, three subroutine addresses are vital for understanding the kernel subroutines. These are listed below.

$2339 Input a character without echo to output.
$2340 Input a character with echo to output.
$2343 Output a character.

Input and output for these subroutines is set by the I/O command.

Most programs are greatly enhanced if they can: (1) give instructions or state questions for users; (2) receive replies or input from users; and (3) convert ASCII hex input to binary and vice versa. The kernel contains subroutines to perform these functions. Below are some of the useful routines in the kernel.

Carriage return, line feed ($2D6A)

This routine sends a carriage return followed by a line feed to the output ($2343). It preserves the X-and Y- registers and uses no Z-page locations.

Output a string of embedded text ($2D73)

Assume we have the code listed below.

```
XX00 20732D JSR $2D73
XX03 484921 HI!
XX06 00
XX07 A200 LDX #$00
```

Suppose this segment of code is embedded in our machine language program and the computer is executing instructions just prior to address $XX00. When $XX00 is encountered, the computer jumps to the kernel subroutine at $2D73. This subroutine treats every byte from $XX03 onward as ASCII text to be sent as output ($2343) until the next $00 is encountered. The code above sends the message 'HI!'. Once output is stopped with a $00 (in this case at address $XX06), control is returned to the main program at the next address (in this case $XX07) where execution continues.

Both the Y-register and the Accumulator are destroyed by this routine, but the X-register remains intact. Z-page locations $E3 and $E4 point to the address (low byte-high byte) before the beginning of the embedded text ($XX02 in the example above). Thus, up to 254 characters may be sent out by this routine. More characters may be sent by repeatedly calling the subroutine.

Line buffer input ($2C98)

The buffer is in $2E1E to $2E2F. The subroutine begins with a carriage return ($0D) and line feed ($0A). Further, a carriage return terminates input and is stored in the buffer. Therefore, the user may input up to 17 additional characters in the buffer. Backarrow ($5F) is the standard erase character used by OSI so that from the polled keyboard (shift-locked) Shift-O erases a character. If you disassemble this subroutine you will see a clever use of the routine $2D73. It is used to output backspaces and spaces in order to erase characters on output. Input is obtained via the subroutine $2340 and subroutine $2D6A is called to send out a carriage return followed by a line feed.

This program destroys all registers. It uses only Z-page locations via $2D73. At $2C9B it resets the line buffer output terminator at $2CED.

Line buffer output ($2CE4)

Each time this routine is called, it returns the next character in the line buffer in the Accumulator. The line buffer pointer ($2CE5) is the operand of an LDY #NN instruction at $2CE4. Locations $E1 and $E2 in Z-page point to the beginning of the line buffer and the Y-register is used to index the buffer. After the seventeenth character the buffer will return a carriage return in the Accumulator. The subroutine leaves only the X-register intact.

ASCII hex to binary nibble ($2D3D,$2D40)

If entered at $2D3D, this routine will read the next buffer character ($2CE4), or you may enter the subroutine at $2D40 with an ASCII hex digit in the Accumulator. It will return with a binary number (0-15) in the first four bits of the Accumulator and 0's in the upper four bits. If entered at $2D40, it uses no Z-page locations and leaves the X- and Y-registers intact, provided there is no error. If something other than an ASCII hex digit is read, subroutine $2CA4 is called to output an error Number 7 (Syntax Error). Further, return will occur to the controlling software system via the link set in the jump at $2A4E.

OSI

SOFTWARE FOR OSI

OSI

A JOURNAL FOR OSI USERS!!

The Aardvark Journal is a bimonthly tutorial for OSI users. It features programs customized for OSI and has run articles like these:

1) Using String Variables.

2) High Speed Basic On An OSI.

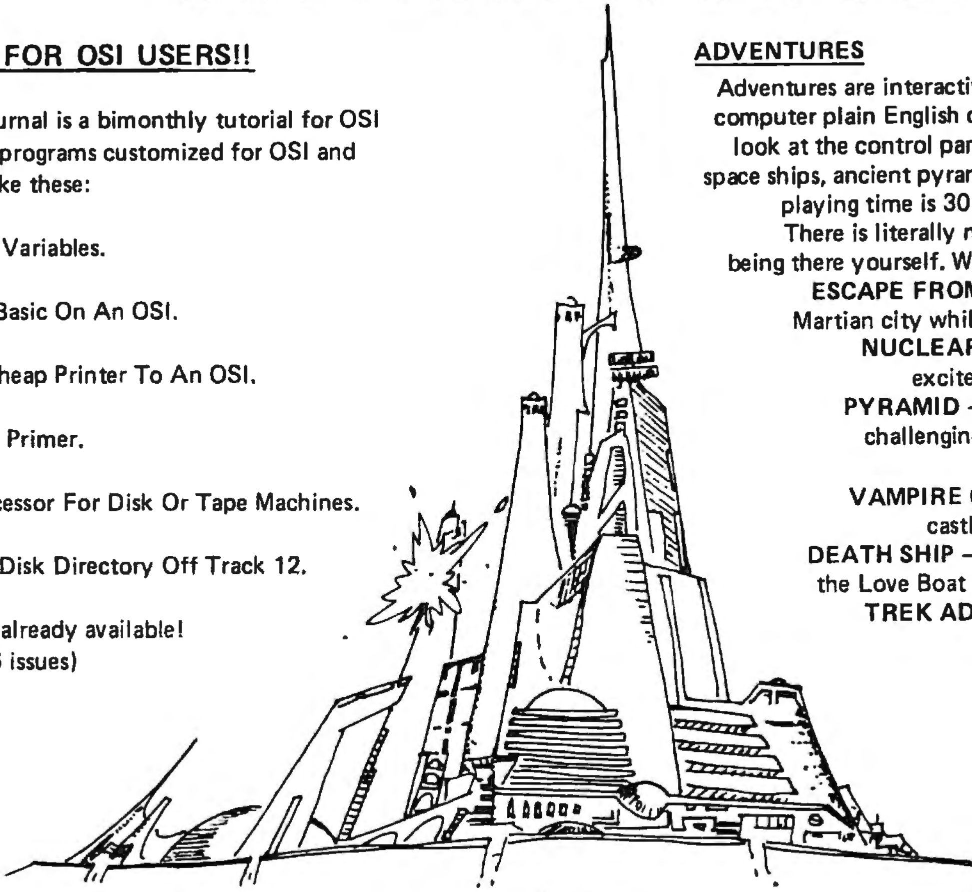
3) Hooking a Cheap Printer To An OSI.

4) An OSI Disk Primer.

5) A Word Processor For Disk Or Tape Machines.

6) Moving The Disk Directory Off Track 12.

Four back issues already available!
$9.00 per year (6 issues)



ADVENTURES

Adventures are interactive fantasies where you give the computer plain English commands (i.e. take the sword, look at the control panel.) as you explore alien cities, space ships, ancient pyramids and sunken subs. Average playing time is 30 to 40 hours in several sessions. There is literally nothing else like them — except being there yourself. We have six adventures available.

ESCAPE FROM MARS — Explore an ancient Martian city while you prepare for your escape.

NUCLEAR SUBMARINE — Fast moving excitement at the bottom of the sea.

PYRAMID — Our most advanced and most challenging adventure. Takes place in our own special ancient pyramid.

VAMPIRE CASTLE — A day in old Drac's castle. But it's getting dark outside.

DEATH SHIP — It's a cruise ship — but it ain't the Love Boat and survival is far from certain.

TREK ADVENTURE — Takes place on a familiar starship. Almost as good as being there.

$14.95 each

NEW SUPPORT ROMS FOR BASIC IN ROM MACHINES

C1S — for the C1P only, this ROM adds full screen edit functions (insert, delete, change characters in a basic line.), Software selectable scroll windows, two instant screen clears (scroll window only and full screen.), software choice of OSI or standard keyboard format, Bell support, 600 Baud cassette support, and a few other features. It plugs in in place of the OSI ROM. NOTE: this ROM also supports video conversions for 24, 32, 48, or 64 characters per line. All that and it sells for a measly $39.95.

C1E/C2E for C1/C2/C4/C8 Basic in ROM machines.
This ROM adds full screen editing, software selectable scroll windows, keyboard correction (software selectable), and contains an extended machine code monitor. It has breakpoint utilities, machine code load and save, block memory move and hex dump utilities. A must for the machine code programmer replaces OSI support ROM. Specify system $59.95

DISK UTILITIES

SUPER COPY — Single Disk Copier
This copy program makes multiple copies, copies track zero, and copies all the tracks that your memory can hold at one time — up to 12 tracks at a pass. It's almost as fast as dual disk copying. — $15.95

MAXIPROSS (WORD PROCESSOR) — 65D polled keyboard only - has global and line edit, right and left margin justification, imbedded margin commands, choice of single, double or triple spacing, file access capabilities and all the features of a major word processor — and it's only $39.95.

P.C. BOARDS

MEMORY BOARDS!! — for the C1P. — and they contain parallel ports!
Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available now!

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. - $14.95.

ARCADE AND VIDEO GAMES

ALIEN INVADERS with machine code moves — for fast action. This is our best invaders yet. The disk version is so fast that we had to add selectable speeds to make it playable.
Tape - $10.95 — Disk - $12.95

TIME TREK (8K) — real time Startrek action. See your torpedoes move across the screen! Real graphics — no more scrolling displays. $9.95

STARFIGHTER — a real time space war where you face cruisers, battleships and fighters using a variety of weapons. Your screen contains working instrumentation and a real time display of the alien ships. $6.95 in black and white - $7.95 in color and sound.

MINOS — A game with amazing 3D graphics. You see a maze from the top, the screen blanks, and then you are in the maze at ground level, finding your way through on foot. Realistic enough to cause claustrophobia. — $12.95

SCREEN EDITORS

These programs all allow the editing of basic lines. All assume that you are using the standard OSI video display and polled keyboard.

C1P CURSOR CONTROL — A program that uses no RAM normally available to the system. (We hid it in unused space on page 2). It provides real backspace, insert, delete and replace functions and an optional instant screen clear.
$11.95

C2/4 CURSOR. This one uses 366 BYTES of RAM to provide a full screen editor. Edit and change lines on any part of the screen. (Basic in ROM systems only.)

FOR DISK SYSTEMS — (65D, polled keyboard and standard video only.)

SUPERDISK. Contains a basic text editor with functions similar to the above programs and also contains a renumberer, variable table maker, search and new BEXEC* programs. The BEXEC* provides a directory, create, delete, and change utilities on one track and is worth having by itself. — $24.95 on 5" disk - $26.95 on 8".

AARDVARK IS NOW AN OSI DEALER!

Now you can buy from people who can support your machine.

—THIS MONTH'S SPECIALS—

Superboard II	$279
C1P Model II	429
C4P	749

. . . and we'll include a free Text Editor Tape with each machine!

Video Modification Plans and P.C. Boards for C1P as low as $4.95

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

OSI

Aardvark Technical Services • 1690 Bolton • Walled Lake, MI 48088
(313) 699-3110 or (313) 624-6316

OSI

A much more useful routine which does the same thing occurs in the ROM machine monitor at $FE93. This latter routine is entered with the hex digit in the Accumulator. It returns with the same data as before except in the case of an error, where $80 is returned in the Accumulator. The ROM subroutine leaves the X- and Y-registers unchanged and uses no Z-page locations.

Full byte binary buffer read ($2D2E)
This routine reads two hex digits from the line buffer and returns with a binary byte in the Accumulator. It calls $2D3D and therefore, has the error procedure of that routine. It uses $E0 as a temporary storage location and affects other registers via subroutine $2CE4.

Full binary address read ($2D23)
By calling $2D2E twice, this subroutine reads four hex digits from the line buffer and stores them as a two byte binary address in Z-page locations $FE and $FF (low byte-high byte).

Nibble to hex digit ($2D9B)
This subroutine converts the first four bits in the Accumulator into an ASCII hex digit and outputs this digit via $2343. It returns with the hex digit in the Accumulator, uses no Z-page addresses, and leaves the X- and Y-registers the same.

One byte binary to two hex digits ($2D92)
By calling $2D9B twice, this routine outputs via $2343 the contents of one full byte binary (in the Accumulator) as an ASCII hex two digit number. It preserves the X- and Y-registers and uses no Z-page locations. The Accumulator is destroyed.

Error output ($2AC4)
If called, this subroutine will reset the 10 flags to the default value, it will disengage the disk head, and it will output "Error # N" where N is a hex digit equal to the first four bits in the Accumulator. Presumably, since an error has occurred, it does not matter which registers have changed.

Stack and Z-page swapper ($2CF7)
This subroutine swaps locations $0000-$01FF (Z-page and the stack) for locations $2F79-$3278 respectively. It returns with the Accumulator and Y-register changed and the X-register equal to 0. When BASIC is resident, OS65D keeps a Z-page and stack separate from BASIC. When the Extended Monitor and Assembler are resident, OS65D and the Extended Monitor keep a Z-page and stack separate from the Assembler.

Shall we swap? ($2D50)
If the contents of $00 are zero the swapper is called, otherwise this subroutine returns with the contents of $00 in the Accumulator and no other changes. BASIC and the Assembler keep nonzero values in $00 while OS65D and the Extended Monitor keep 0 in $00. Thus software can recognize whether or not to swap Z-page and the stack.

Symbol checker ($2D58, $2D5B, $2D5E)
This subroutine reads the buffer to see if the next character is ' = ' ($2D58), ',' ($2D5B), or '/' ($2D5E). If an error occurs the routine behaves as ($2D3D) does, returning to system software control after error Number 7 (Syntax Error). It calls subroutine $2CE4 and uses Z-page location $E0 for temporary storage. This routine uses a standard programming trick of masking 2-byte OPcodes by using a 3-byte BIT instruction.

This concludes a description of the more useful subroutines in the kernel. Most routines are not difficult to decipher. A few have mildly complex flow. The three most involved are: $2A84, The command processor; $2C98, The line buffer input; and $2DA6, The DIRECTORY search. These subroutines are described via flowcharts in Figures 2 to 4. These flowcharts should make it possible to understand disassemblies of the corresponding subroutines.

TABLE 3
MAP - OS65D KERNEL

2A4B
Output an OS65D error # then return to linked software (link is via a jump at 2A4E).

2A51
OS65D Start-up address.

2A7D
Set up the return to software address at 2A4E. Set to 20D7 at 20D1 in BA. Set to 1532 at 152C in ASM. Set to 1756 at 1F31 in in EM. Set to 2A51 at 2A54 in OS65D.

2A84
OS65D Command Processor: called by 2A51. Commands in a table at 2E30 - 2E77.

2AC0
Output ERR# 7: 'SYNTAX ERROR IN COMMAND LINE.'

2AC4
Error message. Enter with error # in accumulator. Resets I/O flags. Disengages disk head.

2ADE
Command AS. Load Tracks 5, 6, and 7, then run the Assembler. Jumps to start at 1300.

2AE6
Command BA. Load Tracks 2, 3, and 4, then run BASIC. Jumps to start at 20E4.

2AEE
Load from the disk the track numbers requested by a command routine starting at 0200 and continuing for 3 tracks.

2B11
Command CA. Call a track and sector from the disk to memory.

2B1A
Engage head, read a sector to memory, then disengage the head.

2B23
Command D9. Disable error #9 in the disk routines. This routine is not called in my version of OS65D. It may be called by changing the address in the COMMAND DIRECTORY.

2B29
Command DI. Give a sector map of a track.

2B2F
Command EM. Load Tracks 5, 6, and 7, then run Extended Monitor. Jumps to start at 1700.

2B37
Command EX. Load an entire disk track to memory for examination.

2B46
Command GO. Start a machine program at specified address.

2B55
Command IN. Initialize a track or the whole disk.

2B68
Text: 'ARE YOU SURE?'

2B83
Command IO. Change the I/O flags.

2BA7
Command LO. Load a named disk file to memory.

2BC6
Command ME. Sets the vectors for memory input and output.

2BDD
Command PU. Puts named file on disk.

2BFD
Command RE. Returns from OS65D to linked software. If software is not in memory, return set to 2AC0 for error #7 out. Settings as follows: ASM to 1303; EM to 1700; BA to 20C4; and M to FEFC (which jumps to FE00).

2C22
Command XQ. Load (starting at 3179) and execute (starting at 317E) a named program

2C28
Command SA. Save memory on a specified sector and track of the disk.

2C43
Command SE. Select a disk drive (A,B,C,D,).

2C60
Get the disk ready for a read or write on a given sector and track.

2C70
Buffer loader. Set the disk start vector to 3179. Engage the disk head.

2C83
Advance head one track. Check for the last track in a file. Report error #D if a read goes beyond the last track of the file.

2C98
Carriage return, line feed, then:

2C9B
Enter and edit a line in the OS65D line buffer at 2E1E - 2E2F.

2CD3
Three empty bytes.

2CD6
Routes input to the Indirect File.

2CE4
Read a line from the OS65D line buffer software, one character at a time.

2CF7
Swapper routine. Switches 0-page and stack for 2F79 - 3178.

2D23
Read 4 ASCII hex digits from the buffer and convert to 2 bytes of binary. Store in FE, FF.

2D2E
Read 2 ASCII hex digits from the buffer and convert to 1 byte binary in accumulator.

2D3D
Read 1 ASCII hex digit from buffer and convert to 1/2 low byte binary in accumulator. Enter at 2D40 with digit in accumulator to skip buffer read.

2D50
Swapper flag check. Initialize for a return to BASIC after an error message. (See BA addresses 20D7 and 20C7).

2D58/2D5B/2D53
Check character to see if it is ' = ', ',', or '/'. Three entry points. Two hidden by BIT instructions.

2D6A
Carriage return and line feed.

2D73
Display embedded text. Display text from the JSR 2D73 instruction until the next null (00).

2D92
1 byte binary in accumulator is converted to 2 ASCII hex digits and displayed in order.

2D9B
Low half byte binary in accumulator is converted to 1 ASCII hex digit and displayed.

2DA6
Directory search. The code from 2DA6-2E1D searches the DISK DIRECTORY to match a file name in the OS65D Buffer with one in the DIRECTORY. When a match is found, the track numbers of the file are saved: last track in 00E5; first track in the accumulator. If a track number (rather than a file name) is given then the track number is read from the line buffer. This routine is used by PU and LO to process the DISK DIRECTORY.

2E1E-2E2F
OS65D Line buffer.
2E30-2E77
OS65D Command directory. 4 bytes per command. First two bytes = First two ASCII letters of Command. Second two bytes = Address of routine - 1.
2E79-2F78
DISK DIRECTORY buffer.
2F79-3078
Buffer for Swapper. Swapped 0-page and stack put here.
3179-317A
Source file start address. (317F if no disk buffers, 3D7F for one buffer, and 497F for two buffers. Address as low byte - high byte.
317B-317C
Source file end address. Address as low byte - high byte.
317B-317C
Source file end address. Address as low byte - high byte.
317D
Number of disk tracks needed to store source file.
317E
Null (00).

OS65D COMMAND DIRECTORY

LOCATION	COMMAND	ROUTINE	ADDRESS
2E30	AS	DD	2A
2E34	BA	E5	2A
2E38	CA	10	2B
2E3C	D9	BF	2A
2E40	DI	28	2B
2E44	EM	2E	2B
2E48	EX	36	2B
2E4C	GO	45	2B
2E50	HO	62	26
2E54	IN	54	2B
2E58	IO	82	2B
2E5C	LO	A6	2B
2E60	ME	C5	2B
2E64	PU	DC	2B
2E68	RE	FC	2B
2E6C	XQ	21	2C
2E70	SA	27	2C
2E74	SE	42	2C

COMMAND PROCESSOR ($2A84)

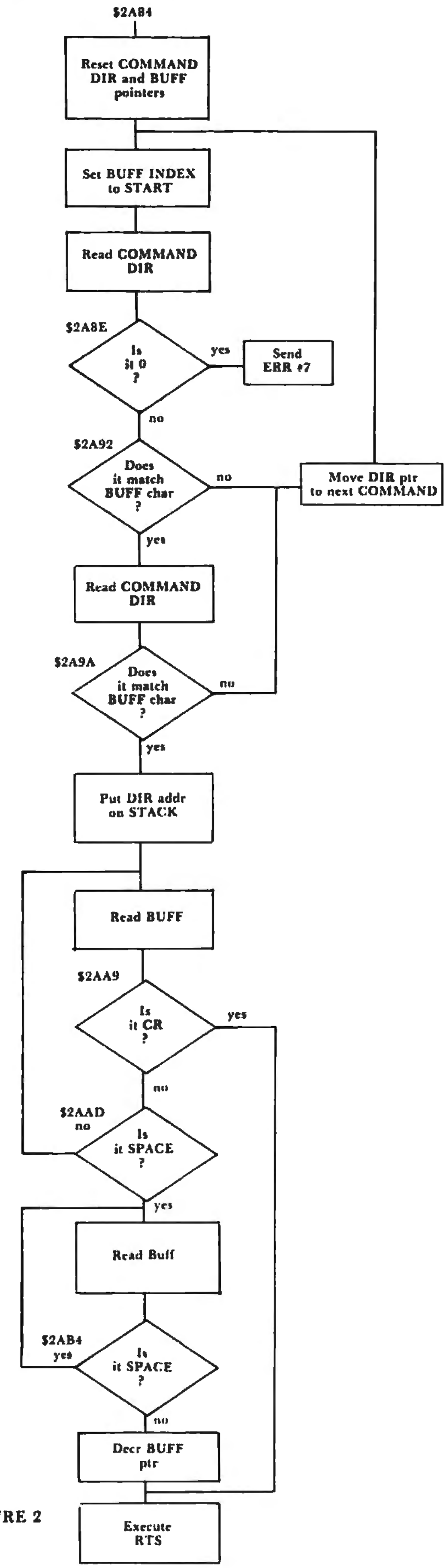


FIGURE 2

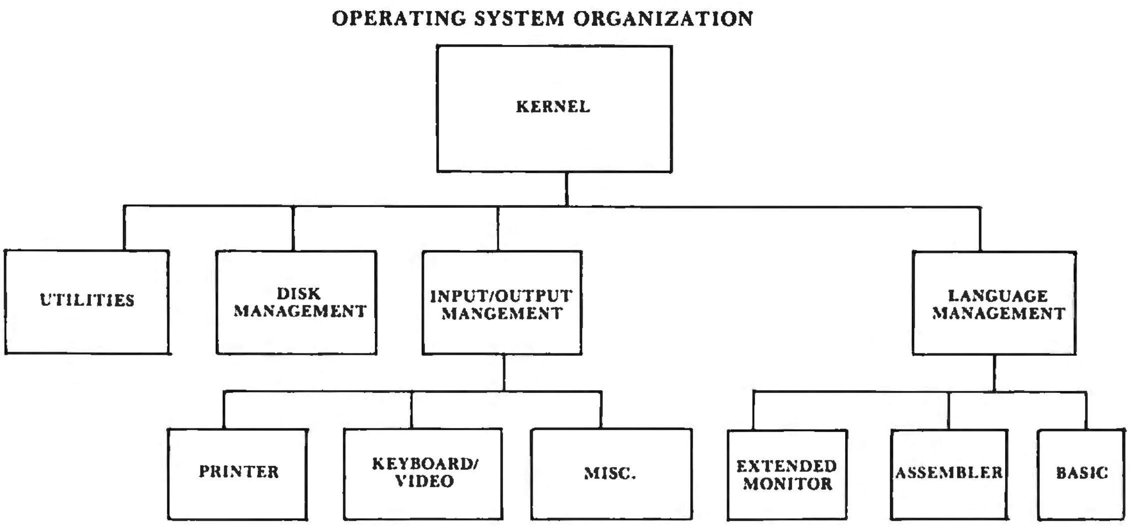


FIGURE 1

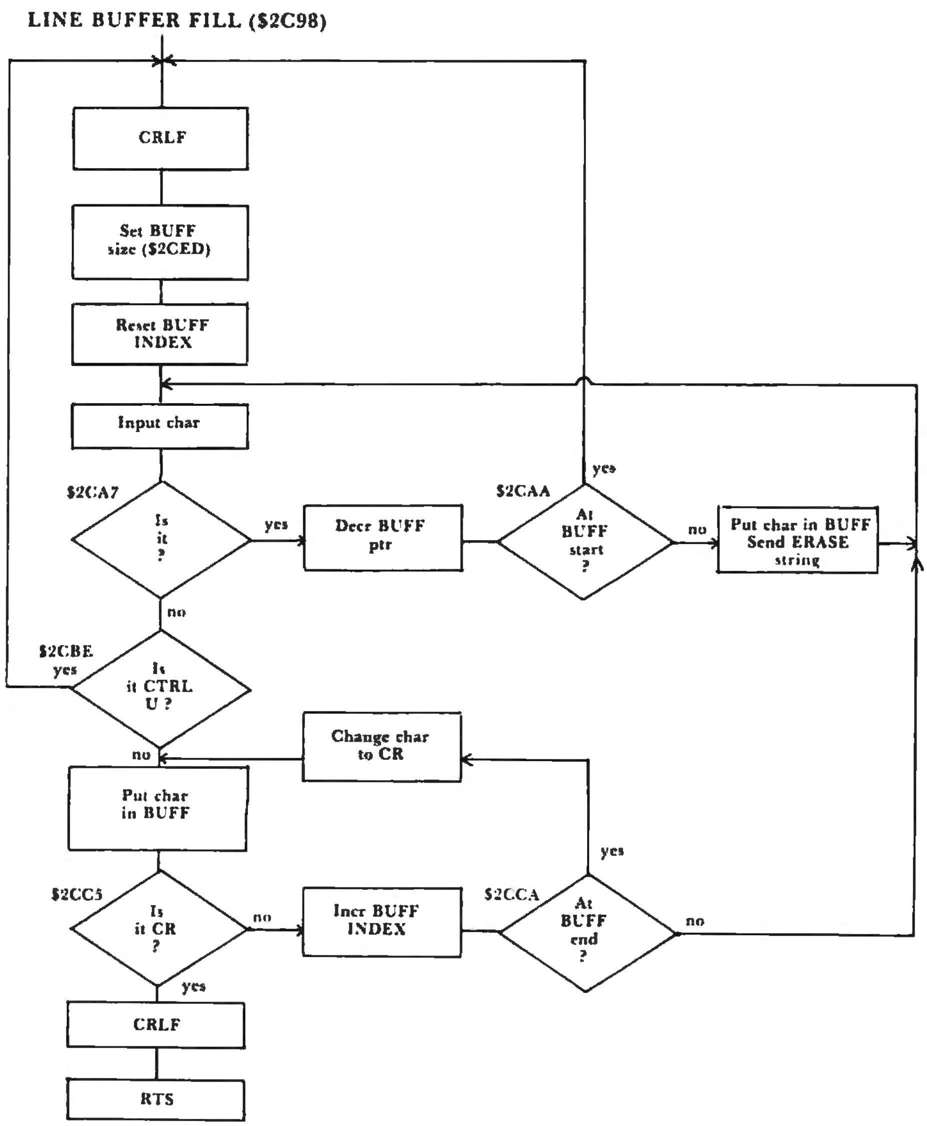


FIGURE 3

DIRECTORY SEARCH ($2DA6, $2DCE)

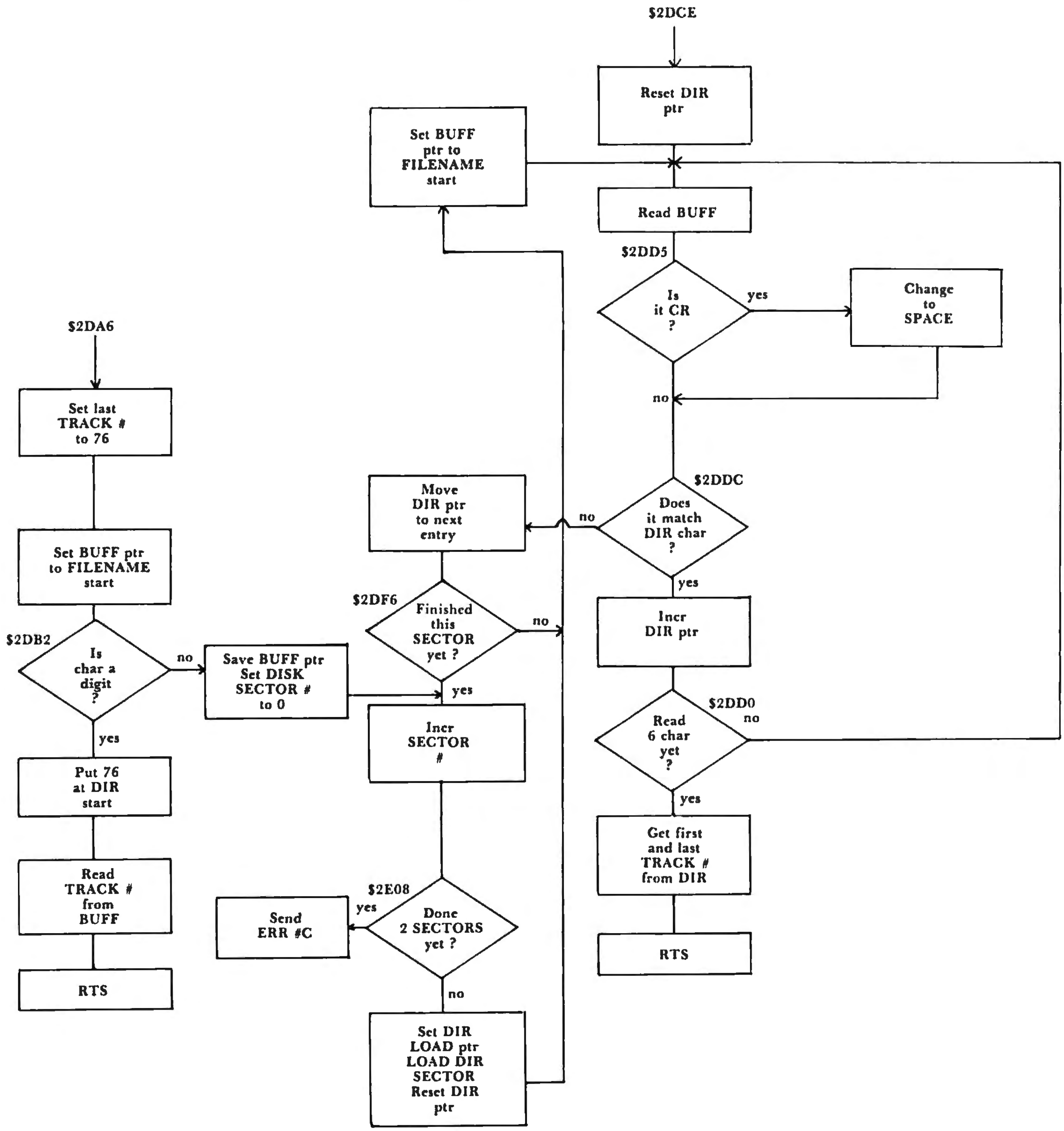


FIGURE 4

©

Book Review

SERVICING DATA FOR COMPUTER BOARDS 600 AND 610

Published by Howard W. Sams and Company, Inc.
8½ x 11" Soft cover, 36 Pages, $7.95

Review by Charles L. Stanford

Howard W. Sams has long been the premier publisher of electronic service data. Their Photofact series covers virtually every audio and audio-visual component available in the free world today. Their technical book line is likewise extremely comprehensive. Collaborating with them in the production of this service manual (and one for the C4P) may well have been one of the wiser moves OSI has made in the area of documentation.

Don't get the idea that this booklet is all things to all people. As implied by the title, only the basic data needed for effective servicing of the machinery are included. But it's all there, including schematics, block diagrams, oscilloscope waveforms, parts lists, and annotated photographs of the boards. The text includes servicing precautions, disassembly instructions, and a troubleshooting guide.

The guide assumes a fairly thorough knowledge of servicing techniques. Beyond that, enough information is provided to isolate defective components or board sections, including a chip-level memory test. To aid in tracing signals, various components on the achematic are color-coded by function, such as video signals, RAM, crystal oscillator section, etc.

The schematics and photographs are on three-or four-section fold-out sheets, which minimizes tracing signals from one side of a page to another. The 600 and 610 boards are shown separately, with jack J1 as the common connector.

The parts list shows both the OSI designation for each component and a cross selection chart for most. For example, the IC chart lists eight manufacturers, and the capacitor chart three. Only a few items such as the ROMs and PROMs, rare ICs, some connectors, etc., show only OSI's part number.

If you never expect to open the case of your C1P, don't bother with this book. But if, like me, you enjoy the "hardware" side of microcomputing, don't pass it up.

©

THE PET GAZETTE

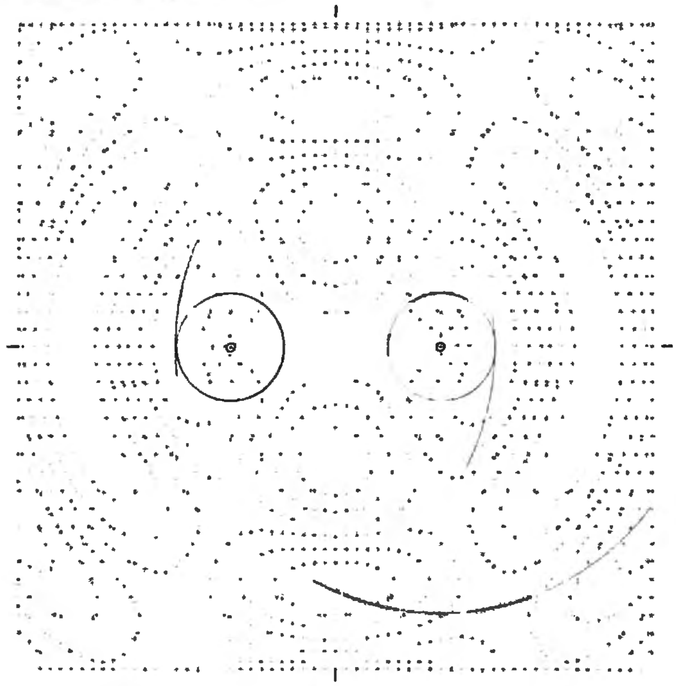
Contour Plotting

Neal E. Reid
Parkside High School,
Dundas, Ontario

In the July/August issue of COMPUTE (p. 73) John Winn showed how to use the 2022 printer to produce graphs of functions of one variable, f(x). Two features of the printer made it possible to increase the resolution from that of a printed character to that of a single printer matrix dot. First of all, the user can define special characters to print any one of the matrix dots alone in the character space. Secondly, because of the variable line spacing on the printer, the character and thus the matrix dot can be printed anywhere on the paper.

The object of this article is to show how to produce graphs of functions of two variables, f(x,y). I am using a 2023 friction feed printer which does not have the variable line spacing capability. As a result, gaps sometimes appear in the plotted lines, but these are readily filled in by the eye and do not seem to be a serious defect.

One cannot approach the problem of graphing a function of two variables in the same way that one does a function of a single variable. In f(x,y), x and y are *both* independent variables. A particular pair (x,y) represents a point in a plane and the value of the function represents heights above or below the plane. I like to picture f(x,y) as a physiographic map. X and y correspond to distances in the east-west and north-south directions, respectively. The function corresponds to the elevation at a particular point (x,y) on the map. Points at which the value of the function is zero are at sea level. Positive values of f(x,y) are above, negative values, below sea level. To represent such a function on a two dimensional sheet of graph paper, one plots lines of *equal* elevation-contour lines. A line connecting the points where f(x,y) = 0, for instance, would be the shore line on a physiographic map. It is customary on such a map to show a constant change in elevation from one contour to the next. Then in regions where the contour lines come very close together, the slope of the land must be very steep - moving a short distance horizontally changes the elevation a great deal. Alternatively, if the contour lines are very widely spaced, then the terrain is relatively flat.



```
INTERFERENCE OF TWO CIRCULAR WAVES
RANGE OF X  -6  TO  6      X-INCREMENT:  .2
RANGE OF Y  -6  TO  6      Y-INCREMENT:  .337333333
CONTOURS:  1.5  .9  .3 -.3 -.9 -1.5
PLOTTING TIME:  40.4  MIN
```

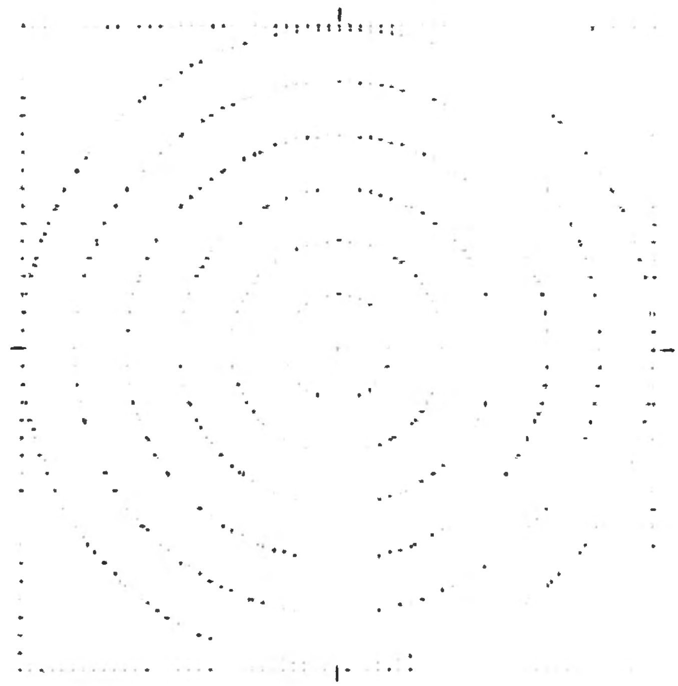
In setting out to draw contours of a function of two variables, the first things to consider are the scale and the position of the graph. The position is fixed by choosing the center of the graph to be at some particular point (X0Y0) in the xy plane. Then by adjusting the scale, one can display a large area or a small neighborhood of that point. Giving the width XR of the graph from the center to the edge will fix the scale. This restricts x and y both to a limited range of values. Thus it is not necessary to print out the graph standing on its side in order to let x have an unlimited range as is usually done with functions of a single variable.

The physical size of the final graph depends on the number of characters per line and the number of lines printed, and this in turn determines the values of the increments in the x and y directions. The printer character matrix is 6 dots wide and 7 dots high with a 3-dot spacing between the lines. By choosing the dimensions of the graph to be 60 characters wide and 36 lines long, we end up with a 360x360 dot square, 6 inches on a side, which fits nicely on an 8½x11 sheet of notebook paper. The boundary of this region is made using a special character MK$ consisting of the single dot in the upper left hand corner of the print matrix. It is printed around all four sides of the square. Moving across the page, the x increment, the change in x from one dot to the next, is XR/30 (center to edge width/30 characters). If the graph boundary were a perfect square, then the y increment would be XR/18. On my printer, however, the vertical dimension comes out about 1/16 inch longer than the horizontal. For truly precision work, one should correct for this; thus, the peculiar factor in line 1035 arrived at purely by trial. The scale, increments, etc., are all taken care of in the SET UP subroutine starting at line 900. The increments and ranges of x and y are printed out at the end of the plot. With this information, these dots on the boundary provide an accurate scale for the final graph. In addition, there are tick marks on the edges to locate the exact center of the graph.

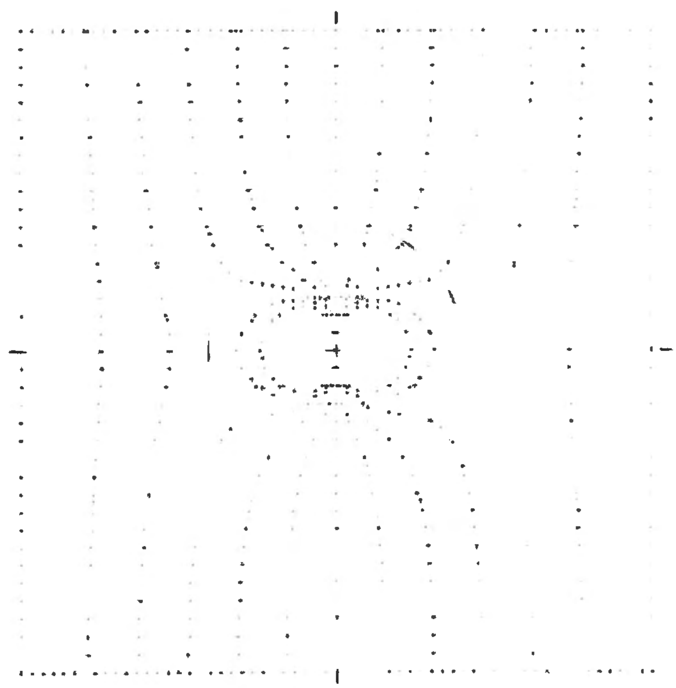
The procedure to create a contour plot for a given function is now straightforward. First, values of the function are calculated at points spaced uniformly over the entire page using the increments of x and y previously worked out. Then we examine every pair of points to see if the contour passes between them. The points at which the function is calculated are those corresponding to the dot in the upper left hand corner of each character. Actually this is done only one line at a time in the subroutine beginning on line 400. After finishing two lines, we have the situation depicted in the diagram. The open circles are the points at which the value of the function is now known.

To find out if the contour we are interested in cuts through the space occupied by the first character, we test to see if the value of the contour lies between the values F1(1) and F2(1) on the horizontal line. This test is made in line 600. If the test is passed, then the particular dot at which the contour cuts the top edge of the matrix is found by linear interpolation. This test is repeated in line 610 for the vertical line between F1(1) and F2(1). Each character space is examined in the same manner, and whenever a contour crossing is found, a dot is printed on the left and/or top edge of the matrix. Dots on the left edge which fall into the space between two lines cannot be printed. Other dots in the matrix are never printed. When the first line of characters is completed, the functional values in F2 are shifted into F1 and the next line of values is put into F2. This way the program never requires more than two lines of values in memory at any one time.

I have used the function $f(x,y) = x^2 + y^2$ as a sort of test pattern to check out the program. The contours of this function are circles centered at the origin. The accuracy in the positions of the plotted points can be checked with a compass. (The user should check this for himself.) If the width XR is set equal to 3 and the contour values are taken to be 0.25, 1.00, 2.25, 4.00, 6.25, and 9.00, then the radii of the circles differ by ½ inch from ½ to 3 inches.



TEST PATTERN – CONCENTRIC CIRCLES
RANGE OF X: -3 TO 3 X-INCREMENT: .1
RANGE OF Y: -3 TO 3 Y-INCREMENT .168666667
CONTOURS .25 1 2.25 4 6.25 9
PLOTTING TIME· 23.9 MIN



FLUID FLOW AROUND A SPHERE
RANGE OF X: -2.5 TO 2.5 X-INCREMENT .0833333333
RANGE OF Y: -2.5 TO 2.5 Y-INCREMENT· .140555556
CONTOURS: 2 1.6 1.2 .8 .4 0 -.4 -.8 -1.2 -1.6 -2
PLOTTING TIME: 38.6 MIN

The user should set his own function equal to F2(J) in line 440.

The second diagram is an example of a typical contour plot. It shows the equipotential lines in the magnetic field of a bar magnet. The north pole of the magnet, which has been drawn on the plot, is at x = -1. The south pole is at +1. The potential at an arbitrary point in the plane is given by the function.

$$f(x,y) = \frac{1}{\sqrt{(x+1)^2 + y^2}} - \frac{1}{\sqrt{(x-1)^2 + y^2}}$$

The lines of force start at the north pole, end at the south pole, and are everywhere perpendicular to the contours. The program does not label the contour lines, but it is not difficult to figure out which is which. For one thing you can plot one contour at a time until you see where they lie. In this plot the contours increase toward the left from zero on the y axis to .1 which is the innermost one. They decrease to the right. The plotting time increases with the complexity of the function and with the number of contours. Each plotted point requires an excursion of the print head across the page.

One potential source of trouble is the possibility of a division by zero. In this plot, for instance, this could happen at the positions of the poles. In fact, around the north pole there is a peak which becomes infinitely high at the pole, and at the south pole there is an infinitely deep hole. (Notice the cliff between the poles where the contours coincide!) In practice it is a simple matter to select the center of the graph and the scale so as to avoid having to evaluate the function at these troublesome points. The point (.00001,.00001) is indistinguishable from the point (0,0) to the eye but it makes a difference to the computer.

This plotting routine can be used for any relationship that can be expressed as a funciton of two variables. Some examples which the reader might find interesting to try out are given below. I am sure there are many others. On the other hand, it is of great interest just to experiment with functions of all sorts and see what turns up.

1. Interference of circular waves. Two pebbles are dropped into a pond at the points x = +2 and -2. Draw circles of radii 1 and 5 around these points on the finished graph. These show where the peak of each circular wave lies. Maximum values of the function occur where two of these circles intersect. The wave troughs are half way in between at radii of 3 and 7.

$$f(x,y) = SIN(\tfrac{\pi}{2}\sqrt{(x-2)^2 + y^2}) + SIN(\tfrac{\pi}{2}\sqrt{(x+2^2)+y^2})$$

Center: (0,0) Width: 6
Contours: 1.5, .9, .3, -.3, -.9, -1.5

2. Fluid from around a sphere. (Draw a circle of radius 1 on the finished plot.) The contours are lines of the velocity potential. The flow of the fluid is from right to left. The stream lines of the flow are everywhere perpendicular to the contour lines. Near the sphere they hug the surface.

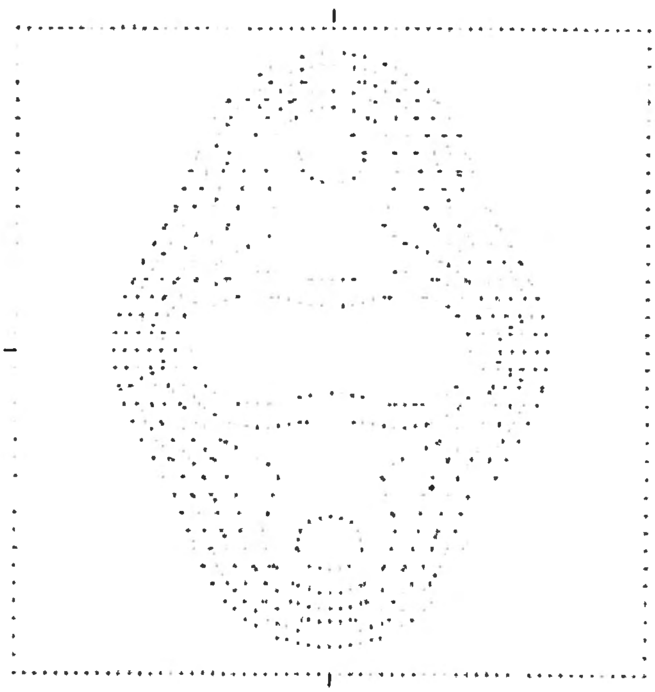
$$f(x,y) = x + \frac{x}{2(x^2 + y^2)^{3/2}}$$

Center: (0,0) Width: 2.5
Contours: 2.0, 1.6, 1.2, .8, .4, 0, -.4, -.8, -1.2, -1.6, -2.0

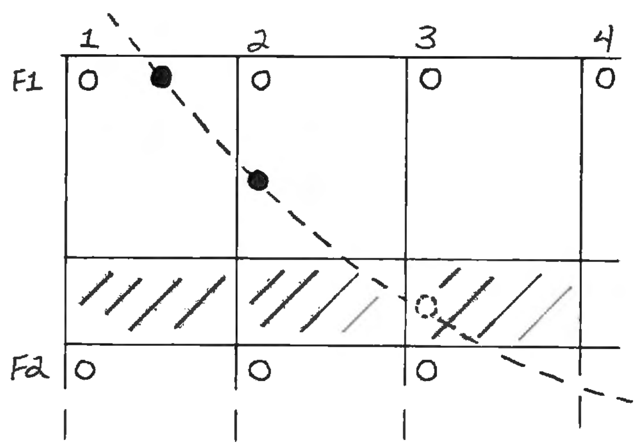
3. The distribution of matter in the nucleus of Neon-20. This is what you could expect to see if you could slice open the nucleus like an apple.

$$f(x,y) = (1.5 + 2x^2 + 2y^4)\,EXP(-(x^2 + y^2))$$

Center: (0,0) Width: 2
Contours: 1.4, 1.3, 1.2, 1.1, 1.0, 0.9, 0.8



```
DISTRIBUTION OF MATTER IN NEON-20
RANGE OF X: -2  TO  2      X-INCREMENT    .0666666667
RANGE OF Y: -2  TO  2      Y-INCREMENT:   .112444444
CONTOURS:   1.4  1.3  1.2  1.1  1  .9  .8
PLOTTING TIME:  33.2  MIN
```



```
50 REM * * * * * * * * * * * * * * *
51 REM *                           *
52 REM *     CONTOUR PLOTTER       *
53 REM *                           *
54 REM * * * * * * * * * * * * * * *
60 :
61 REM NEAL E. REID
62 REM PARKSIDE HIGH SCHOOL
63 REM DUNDAS, ONTARIO
100 :
101 REM MAINLINE
110 GOSUB 900 : REM SET UP
120 PRINT#1, TAB(31)"&" : GOSUB 800
130 LN=0 : PRINT "↓LINE NUMBER:↓↓"
140 GOSUB 400 : REM COMPUTE F2
150 LN=LN+1 : PRINT "↑"LN
160 FOR I=0 TO 60 :F1(I)=F2(I) :NEXT
170 GOSUB 400 : GOSUB 500
180 IF LN<36 GOTO 150
190 GOSUB 800 : PRINT#1, TAB(31)"&"
200 :
210 PRINT#1 : PRINT#1
215 PRINT#1, TL$ : PRINT#1
220 PRINT#1, "RANGE OF X: ";X0-XR;
230 PRINT#1, " TO ";X0+XR;
240 PRINT#1, "      X-INCREMENT: ";XI
250 PRINT#1
260 PRINT#1, "RANGE OF Y: ";Y0-XR;
270 PRINT#1, " TO ";Y0+XR;
280 PRINT#1, "      Y-INCREMENT: ";YI
290 PRINT#1
300 PRINT#1, "CONTOURS:  ";
310 FOR K=1 TO NC
320 PRINT#1, CT(K); : NEXT
330 PRINT#1 : PRINT#1
340 T%=(TI-TM)/360
350 PRINT#1, "PLOTTING TIME: ";
360 PRINT#1, T%/10;" MIN"
370 CLOSE 1 : CLOSE 5
380 PRINT "PLOT COMPLETED"
399 END
400 :
401 REM CALCULATION OF FUNCTIONS
402 REM ON 60X36 GRID
403 REM ONE LINE AT A TIME
404 REM COORDINATES ARE (X,Y)
410 Y=Y-YI
420 FOR J=0 TO 60
430 X=XS+J*XI
440 F2(J)=X^2+Y^2
450 NEXT J
499 RETURN
500 :
501 REM PLOT CONTOURS
510 REM BOUNDARY AND X MARKER -
520 PRINT#5, MK$
530 IF LN<>19 THEN PRINT#1, SP$SC$TAB(59
   ¬)SC$CR$;
540 IF LN=19 THEN PRINT#1, "‡"SC$TAB(59)
   ¬SC$"‡"CR$;
550 REM LOCATE CONTOURS -
560 FOR J=0 TO 59
570 A=F1(J) : B=F1(J+1) : C=F2(J)
580 FOR K=1 TO NC : NV=0 : NH=0
590 CN=CT(K)
600 IF A<=CN AND CN<B OR A>=CN AND CN>B ¬
   ¬THEN NV=INT(6*(CN-A)/(B-A))+1
610 IF A<=CN AND CN<C OR A>=CN AND CN>C ¬
   ¬THEN NH=INT(10*(CN-A)/(C-A))+1
620 IF (NH=O OR NH>7) AND NV=0 GOTO730
630 REM CREATE SPECIAL CHARACTER -
640 A1=2^(7-NH) :IF NV=1 THEN A1=A1+64
650 A2=-64*(NV=2)
660 A3=-64*(NV=3)
670 A4=-64*(NV=4)
680 A5=-64*(NV=5)
690 A6=-64*(NV=6)
700 A$=CHR$(A1)+CHR$(A2)+CHR$(A3)
705 A$=A$+CHR$(A4)+CHR$(A5)+CHR$(A6)
710 PRINT#5, A$
720 PRINT#1, TAB(J+1)SC$CR$;
730 NEXT K
740 NEXT J
799 PRINT#1 : RETURN
800 :
801 REM PRINT BOUNDARY
810 PRINT#5, MK$
820 PRINT#1, SP$;
830 FOR I=1 TO 61
840 PRINT#1, SC$; : NEXT
850 PRINT#1, CR$;
899 RETURN
900 :
901 REM SET UP
910 DIM F1(60),F2(60),CT(12)
920 DATA 64,0,0,0,0,0
930 FOR I=1 TO 6 : READ A
940 MK$=MK$+CHR$(A) : NEXT
945 INPUT "♥TITLE";TL$
950 PRINT "↓X,Y COORDINATES OF"
960 INPUT "CENTER OF PLOT";X0,Y0
970 INPUT "↓CENTER TO EDGE WIDTH";XR
980 INPUT "↓NUMBER OF CONTOURS";NC
990 PRINT "↓CONTOUR VALUES:↓"
1000 FOR K=1 TO NC
1010 INPUT CT(K) : NEXT
1020 XI=XR/30     :REM X-INCREMENT
1030 XS=X0-XR     :REM X INITIAL VALUE
1035 YR=XR*1.012
1040 YI=YR/18     :REM Y-INCREMENT
1050 YS=Y0+YR     :REM Y INITIAL VALUE
1060 Y=YS+YI
1070 SP$=CHR$(29)  :REM SPACE
1080 SC$=CHR$(254) :REM SPEC. CHAR.
1090 CR$=CHR$(141) :REM CAR. RETURN
1100 OPEN 1,4 : OPEN 5,4,5
1110 PRINT "↓INSERT PAPER"
1120 PRINT "↓PRESS ɪRETURNî";
1130 PRINT " TO CONTINUE"
1140 GET P$ : IF P$="" GOTO 1140
1150 IF ASC(P$)<>13 GOTO 1140
1160 TM=TI       : REM SET TIMER
1170 PRINT "↓STARTING"
1199 RETURN
```

©

Relocate

R. D. Young
Ottawa, Ontario

Mysteries can be solved, eventually. This one begins with the article by Harvey B. Herman on 'Memory Partition of BASIC Workspace' (**COMPUTE!**, Issue 2, Jan./Feb. 1980, p. 18). Harvey made reference to a previously written article in MICRO which was to describe the procedure for relocating or loading programs to portions of memory other than from the normal beginning of memory. Unfortunately, I did not have immediate access to his reference, so the loading of saved programs into memory partitions had to wait... indefinitely.

Some hints on the required procedure became available when Roy Busdiecker outlined the relocation of the monitor used by us 'old ROMers' in his article, 'Relocate PET Monitor Almost Anywhere' (**COMPUTE!**, Issue 4, May/June 1980, p. 115). I did not pay much attention to it at the time, but memory expansion suddenly made it useful. When 'Quadra-PET' came along (**COMPUTE!**, Issue 6, Sept./Oct. 1980, p. 90), I was able to piece the puzzle together, but it would have been nice if the procedure had been included in the article.

I suspect that there are still a few 'old ROMers' like me out there; I hesitate to buy a new ROM set as long as the old one is still functioning. For them, I present RELOCATE: a machine language routine that loads programs anywhere as simply as a normal LOAD. Such a routine may well have been published in the past, but repetition can be useful for those whose resources are limited. For new ROMs, I suspect that the routine can be appropriately revised, and I have liberally commented the listing.

With this routine, memory partitioning becomes a reality. A routine like 'Quadra-PET' that would permit switching from one program to another without destroying either is the missing link.

The mystery was solved, exposing yet another.

RELOCATE

```
0348 20 36 E2   Begin  JSR 57910 ; clear screen
034B 20 AE F5          JSR 62894 ; INIT LOAD
034E AE 3A 03          LDX 826 ; start location
0351 AD 7E 02          LDA 638 ; end load high byte
0354 38                SEC ;
0355 ED 7C 02          SBC 636 ; start load high byte
0358 8D 3B 03          STA 827 ; offset
035B A9 04             LDA #4 ; normal start high byte
035D E0 01      Incre  CPX #1 ; ready to load?
035F F0 07             BEQ Load ; YES
0361 18                CLC ; NO
0362 69 04             ADC #4 ; increment by 1K
0364 CA                DEX ;
0365 4C 5D 03          JMP Incre ;
0368 8D 7C 02   Load   STA 636 ; new start high byte
036B 85 7B             STA 123 ;new start BASIC pointer
036D 18                CLC ;
036E 6D 3B 03          ADC 827 ; add offset
0371 8D 7E 02          STA 638 ; new end high byte
0374 A9 37             LDA #55 ; #7 for screen store
0376 8D 18 81          STA 33048 ; Line 77 hopefully
0379 8D 19 81          STA 33049 ; not in program
037C A9 01             LDA #1 ; dynamic RETURN
037E 8D 0D 02          STA 525 ; to enter 77
0381 A9 0D             LDA #13 ; as a line to
0383 8D 0F 02          STA 527 ; reset line links
0386 4C C3 F3          JMP 62403 ; LOAD
0389                   .END
```

LOAD 'RELOCATE'
NEW

Rewind cassette to EXACT position for desired program.

POKE 826,X where X is the desired starting location for the load in increments of 1K (minimum 1K). Eg. if X = 7, the program will be loaded beginning at the 7K (7168 decimal) location.

Load with SYS 840. Note the contents of locations 123, 124, 125, 134, 135 to be able to return to this program. To return to the beginning of memory, POKE 135, PEEK(123):POKE 123,4:POKE 124,4: POKE 125,4:CLR. ©

Mixing and Matching Commodore Disk Systems

Jim Butterfield

The computer and the disk are separate devices. They communicate only over the IEEE-488 bus. Any Commodore disk can be worked by any PET/CBM system (except the original ROM systems which have an IEEE bus problem).

The newer computers and newer disks seem to work well together. But you can mix and match older systems with the new to suit your own special requirements.

First, a little terminology. "New disk" doesn't just mean the 77-track 8050 unit. The 2040 units can be fitted with equivalent logic which provide auto-initialization, file append, and relative files. New 2040's will be shipped that way, and old ones can be retrofitted with new ROM sets. Disk units that have the new features will be referred to as "DOS 2.0" systems; the original 2040 units without the extra features will be "DOS 1.0" systems.

Similarly, the 80-column computers give you new Basic commands such as SCRATCH or APPEND; but you can also get these features on newer 40-column machines, and some older machines can be retrofitted. Systems with the new Basic commands will be called "Basic 4.0"; the earlier upgrade ROMs will be referred to as "Basic 2.0". The very first PET units with original ROMs won't be mentioned here; they don't work disk at all.

Why keep the old?

There are a number of reasons that a user might prefer to stay with an older disk or computer ROM.

On his computer, he might have machine language programs that might be difficult to upgrade. He might need all of his spare ROM sockets. Or he might just like the old system and see no reason to pay extra money to go to the new. If he has an early model PET, the new Basic 4.0 might not fit — it requires an extra ROM socket that just isn't there.

On his disk, he might not want to give up a little capacity on the new system: DOS 2.0 gives only 664 blocks as compared to 670 on DOS 1.0, and the directory capacity is trimmed to 144 entries as compared to 152. He might have direct access files which depend on the old allocation patterns of the DOS 1.0 system, and views conversion as too much trouble.

My personal view is that disk upgrade is desirable, but computer upgrade is optional and a matter of preference.

New Computer, Old Disk

It's quite easy to work a DOS 1.0 disk unit with a new Basic 4.0 computer.

You must remember to initialize each new diskette as it's inserted into the unit. The usual way is:

OPEN 15,8,15,"I0" — or any similar sequence.

All of your new Basic commands will work well, except APPEND and RECORD. These will be sent along to the disk, but the disk unit won't understand and will return a SYNTAX ERROR message.

Of course, you can't open a file using the L option: relative files are unknown to a DOS 1.0 unit.

But everything else will work nicely, and you'll have the convenience of commands like CATALOG or SCRATCH to make things easy.

One caution: If you should happen upon a disk that has been initialized on a DOS 2.0 drive, don't try to write on it with your DOS 1.0 system. It might work, but it might also wreck the diskette information. Copy the files over to a disk of your own and you'll be free to make all the change you like.

Old Computer, New Disk

All of the old disk features are preserved. You won't need to initialize, which is a great convenience.

You'll probably want to use that old standby, the DOS Support Program (the "wedge") to help in cataloging and error checking. No problem; everything is as it was before.

When you want to exploit the new features of your DOS 2.0 disk unit, you'll have a little more work. Appending is quite easy. As an example, suppose you have a sequential file called RABBIT and you want to tack some records onto the end. You just open with:

OPEN 1,8,3,"0:RABBIT,A"

..and you're ready to write the extra records. As usual, don't forget to close the file when you're finished.

Handling the new Relative files requires careful coding. You should, of course, read up on this type of file in the manual first. In some ways a relative file can be handled in the same way as a sequential file. The big differences are in two areas: opening the file; and at a later time, positioning so as to read or write a specific record.

To open a relative file the first time, you use a conventional OPEN statement. An example will illustrate the method. Suppose we want to write a relative file called RANDFIL, with each record to be no longer than 25 characters. We would write:

OPEN 1,8,3,"0:RANDFIL,L," + CHR$(25)

Following this, as usual, we would write records to

this file — as many as we think we need — and then CLOSE 1. The records may be blank, but we should write then anyway since we are building the framework into which data will later be placed.

At a later time, we will wish to read or write a specific record in the file. We open the file with a conventional statement:

OPEN 2,8,4,"RANDFIL" : OPEN 15, 8,15

and now we want to position the file to read or write a given record. Let's say we want to write to record-number 30. We code:

PRINT#15,"P" + CHR$(96 + 4) + CGR$(30) + CHR$(0) + CHR$(1)

What's happening here? Well, the P stands for Position; it's the same as the RECORD command in Basic 4.0.

The CHR$(96 + 4) identifies the file as secondary address number 4. The disk unit needs this to identify the file that's needed; going back to the OPEN statement, it will see that file RANDFIL is the one that's wanted.

CHR$(30) + CHR$(0) says that we want to go to record number 30. The second value is the high-order byte (multiples of 256). If we wanted record number 800, this group would be CHR$(32) + CHR$(3).

Finally, the CHR$(1) means that we want to read starting at the first character in the record.

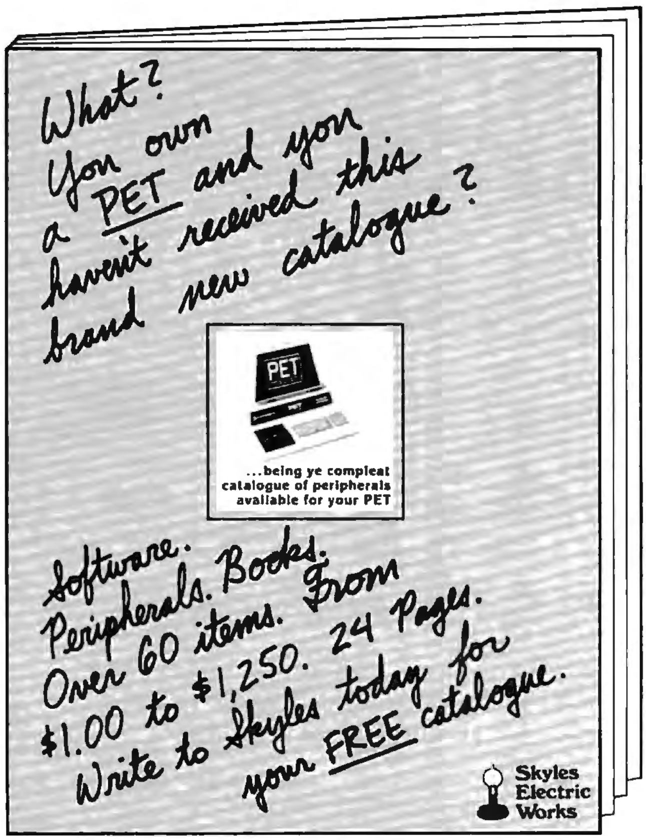
After the positioning is complete, you can then INPUT# or PRINT# in the same way you would for a sequential file.

Summary

You can mix and match disk and computer if you wish. Sometimes it's a little more work to get the most out of the available features, but it's all there.

I sometimes wonder if Basic 4.0 isn't a little too cosmetic. Users may forget (or never find out) that COLLECT is translated to V (for Verify), or that HEADER becomes N (for New). And perhaps they won't need to know such things — their computer will take care of it all for them.

But dedicated users who plunge into the underworld of Machine Language programming will need to know these details. If they know the secret codes, they too can mix and match — but that's another story. ©



Memory Calendar

Peter Spencer
West Hill, Ontario

Have you ever forgotten an important date, anniversary, or payment deadline? Is there a famous person, say Charles Babbage or Jim Butterfield, whose birthday you would like to remember?

Memory Calendar lets you create a file of important messages for each month of the year, and a Common file for messages that occur in all months. You can then print out a calendar (see Figure 1) for any month of any year. This printout will automatically incorporate the messages for the month you have selected, plus the Common messages if you wish, plus any specific one-time messages that you want to add.

I have tried to make Memory Calendar as foolproof as possible, so that it can be used with ease by people who can type but who know little about computers. My intention was to write a useful utility

MEMORY CALENDAR
March 1981

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
1 First successful parachute jump from an airplane 1912	2 Platt Amendment makes Cuba a US protector-ate 1901.	3	4	5	6 First frozen food sold 1930	7
8	9	10	11	12	13	14
15 First escalator patented 1892	16	17	18	19 US votes to stay out of League of Nations 1920	20 Sir Isaac Newton died 1727	21 Persia changes name to Iran 1935 First day of spring
22	23	24	25 First passenger railway service 1807	26	27 Gesner patents kerosene 1855	28
29 Santa Anna takes power in Mexico 1823	30 Crimean War ended 1856	31 Perry signs treaty opening Japan to West 1854				

Figure 1. A typical Memory Calendar printout.

that would work for anyone.

Writing a program that maintains and merges files is not that difficult, as most of Compute's readers know. The main problem I found in coding Memory Calendar was in the printout section, where each line of print must have pieces of as many as seven different messages in it, and each piece must line up with the day it was intended for.

If you would like a copy of the program (see Figure 2) without having to type it in yourself, send me a diskette and I will make you a copy of Memory Calendar for no charge. Hearing from other Compute readers has so far been a pleasure.

Figure 2. Program Listing of Memory Calendar.

```
10 CLR
20 REM  COPYRIGHT (C) 1980 BY P.T.SPENCE
     ¬R.  ALL RIGHTS RESERVED.
30 REM  COPY BUT DO NOT SELL
40 REM  P.T.SPENCER
50 REM  7 BRIGHTSIDE DRIVE
60 REM  WEST HILL, ONTARIO
70 REM  CANADA  M1E 3Y8
80 REM  (416) 281-1155
90 REM  1980 12 14
100 REM  WRITTEN FOR BASIC 2.0 AND DOS ¬
     ¬1.0
110 DEF FNFR(X)=PEEK(48) +256*PEEK(49) ¬
     ¬-(PEEK(46) +256*PEEK(47))
120 PRINT"ĥɼMEMORY CALENDARȓ ↓↓(C) P.T. ¬
     ¬SPENCER 1980"
130 POKE 59468,PEEK(59468)OR14
140 DIM A$(42,9)
150 OPEN 15,8,15
160 PRINT"↓↓HIT ANY KEY TO CONTINUE ";
170 GOSUB 1690
180 :
190 REM:  INITIALIZATION
200 N=0
210 MK=1
220 B$="                                    ¬
     ¬          "
230 M1$=""
240 DIM WD$(6)
250 DATA"SUNDAY","MONDAY","TUESDAY",
     ¬"WEDNESDAY","THURSDAY","FRIDAY",
     ¬"SATURDAY"
260 FOR J=0 TO 6 :READ WD$(J) :NEXT J
270 :
280 PRINT"ĥɼSȓTART NEW FILE, OR ɼWȓORK ¬
     ¬ON OLD FILE? ";
290 GOSUB1690
300 IF S$<>"W" AND S$<>"S" GOTO 280
310 PRINT"↓NAME OF MONTH (OR COMMON)"; :
     ¬LL=10:GOSUB2440
320 AA$=IN$
330 FOR I=1 TO 13
340 READ A3$,ND
350 IF A3$=LEFT$(AA$,3) THEN MN=I :
     ¬GOTO 390
360 NEXT I
370 DATA"JAN",31,"FEB",29,"MAR",31,
     ¬"APR",30,"MAY",31,"JUN",30,"JUL",
     ¬31
380 DATA"AUG",31,"SEP",30,"OCT",31,
     ¬"NOV",30,"DEC",31,"COM",31
390 IFS$="W" THEN AC$=AA$ :GOTO 540
400 :
410 PRINT"↓ɼSTARTING NEW FILEȓ"
420 PRINT"ONE MOMENT PLEASE"
430 FOR I=1 TO 42
440 A$(I,1)="ɼ "+MID$(STR$(I),2)+RIGHT$(
     ¬"           ȓ",9-LEN(MID$(STR$(I),
     ¬2)))
450 REM WARNING***FIRST BLANK IN LINE ¬
     ¬ABOVE IS A CHR$(160)
460 NEXT I
470 FORI=1 TO 42
480 FOR J=2 TO 8
490 A$(I,J)=CHR$(160)
500 NEXT J
510 NEXT I
520 GOTO1050
530 :
540 REM:  READ FROM DISK
550 PRINT"↓INSERT DISK WITH ";AC$;" ¬
     ¬FILE ";
560 PRINT"IN RIGHT DRIVE AND TYPE ɼGȓO ¬
     ¬";:GOSUB 1690
570 PRINT#15,"I0"
580 FA$="0:" + AC$ + ",S,R"
590 OPEN 5,8,2,FA$:GOSUB1750
600 INPUT#5,AB$ :RS=ST:GOSUB1750
610 IF RS<>0 THEN 850
620 IF (AC$<>AB$) THEN PRINT"ɼFILE ¬
     ¬MISMATCH" :STOP
630 FOR I=1 TO ND
640 FOR J=1 TO 8
650 INPUT#5,IN$ :RS=ST:GOSUB1750
660 IF MK=1 THEN A$(I,J)=IN$ :GOTO 730
670 G9%=0
680 IF LEFT$(IN$,2)="ɼ"+CHR$(160) OR ¬
     ¬IN$=CHR$(160) GOTO 730
690 FOR K=J TO 8
700 IF A$(I,K)=CHR$(160) THEN A$(I,
     ¬K)=IN$ :K=J+8 :G9%=1
710 NEXT K
720 IF G9%=0 THEN PRINT"ɼDAY";I;"IS ¬
     ¬FULL--DISCARDEDȓ ";IN$
730 IF RS=64 THEN 820
740 IF RS<>0 THEN 850
750 NEXT J
760 NEXT I
770 CLOSE 5
780 IF MK=0 THEN PRINT"↓HIT ANY KEY TO ¬
     ¬CONTINUE "; :GOSUB 1680
790 MK=0
800 GOTO1060
810 :
820 PRINT"ɼEND OF DISK FILE" :FOR I=1 ¬
     ¬TO 1000:NEXT I
830 CLOSE 5:GOTO1060
840 :
850 PRINT"BAD DISK STATUS IS";RS
860 CLOSE 5:CLOSE 15:STOP
870 :
880 REM:  SCROLL ROUTINE
890 INPUT"↓DISPLAY ON PRINTER OR ¬
     ¬SCREEN>>>S<<<";IN$
900 SN=3 :IF LEFT$(IN$,1)="P" THEN SN=4
910 IF SN=4 THEN PRINT"↓ɼSET UP PRINTER,
     ¬ THEN HIT ANY KEY "; :GOSUB 1680
920 OPEN3,SN
930 FOR DY=1 TO ND
940 D1$=STR$(DY)
950 PRINT#3,CHR$(1)+D1$
960 FOR I=2 TO 8
970 IF A$(DY,I)<>CHR$(160) THEN PRINT#3,
     ¬CHR$(17) + A$(DY,I)
```

```
980 NEXT I
990 PRINT#3
1000 NEXT DY
1010 PRINT#3
1020 CLOSE3
1030 REM:  DROPS THROUGH TO MENU
1040 :
1050 REM:  MENU ROUTINE
1060 PRINT"ĥ";B$;B$;B$
1070 PRINT"ĥr̲Tf̂YPE, r̲Cf̂HANGE, r̲Pf̂RINT,
     ¬ ";
1080 PRINT"r̲Mf̂EMORIZE, r̲Df̂ISPLAY,
     ¬ r̲Rf̂ECALL C̲OMMON FILE, r̲Ef̂XIT? ";
1090 GOSUB1690
1100 IF S$="C" THEN M1$="C" :PRINT"ĥr̲CHA
     ¬NGE↓" :GOTO 1240
1110 IF S$="E" GOTO1630
1120 IF S$="T" THEN PRINT"ĥr̲TYPE↓" :
     ¬GOTO 1240
1130 IF S$="M"GOTO1450
1140 IF S$="P"THEN1830
1150 IF S$="R" THEN AC$="C̲OMMON" :
     ¬GOTO540
1160 IF S$="D" THEN 1180
1170 GOTO1060
1180 PRINT"↓DISPLAY SINGLE r̲Df̂AY OR ¬
     ¬WHOLE r̲Mf̂ONTH? ";:GOSUB 1690
1190 IF S$="D" GOTO 2770
1200 IF S$="M" GOTO 880
1210 GOTO 1060
1220 :
1230 REM TYPE ENTRY OR CHANGE ENTRY ¬
     ¬ROUTINE
1240 Z9$="" :LL=10
1250 PRINT"E̲ACH MESSAGE CAN HAVE 7 ¬
     ¬LINES             OF 10 CHARACTERS ¬
     ¬EACH."
1260 INPUT"↓W̲HICH DAY>>>*<<<";DY
1270 FOR I=2 TO 8
1280 IF M1$<>"C" AND (A$(DY,I)=CHR$(160)
     ¬OR A$(DY,I)="")THENN=I-1:I=8 :
     ¬GOTO1300
1290 PRINT I-1;A$(DY,I)
1300 NEXT I
1310 IF M1$="C" THEN INPUT"TYPE MESSAGE ¬
     ¬ON WHICH LINE>>>1<<<";N
1320 PRINT"↓r̲TYPE NEW LINE OR LINESf̂"
1330 PRINT"H̲IT RETURN KEY TWICE TO ¬
     ¬STOP.↓"
1340 GOSUB 2630 :PRINTN;
1350 GOSUB 2440
1360 IF IN$="" THEN PRINT"ĥ" :GOTO 2780
1370 A$(DY,N+1)=IN$
1380 PRINT"↑                    "
1390 PRINT"↑";N;A$(DY,N+1)
1400 IF M1$="C" THEN M1$="":PRINT"ĥ" :
     ¬GOTO 2780
1410 N=N+1
1420 IF N>7 THEN PRINT"ĥ" :GOTO 2780
1430 GOTO 1340
1440 :
1450 REM:  OUTPUT TO DISK
1460 INPUT"↓OUTPUT TO DRIVE #>>>0<<<";DD
     ¬%
1470 DD$=STR$(DD%)
1480 PRINT"↓OUTPUT FILE NAME IS ¬
     ¬r̲";AA$;"f̂ OK? ";:GOSUB1690
1490 IFS$<>"Y" GOTO 1060
1500 PRINT#15,"I"+DD$ :GOSUB1750:
     ¬FI$="@"+DD$+":"+ AA$ +",S,W"
1510 OPEN 6,8,2,FI$
1520 GOSUB1750
1530 PRINT#6,AA$;CHR$(13); :GOSUB 1750
1540 FOR I=1 TO ND
1550 FOR J=1 TO 8
1560 PRINT#6,A$(I,J);CHR$(13);:GOSUB ¬
     ¬1750
1570 NEXT J
1580 NEXT I
1590 PRINT#6,"END OF FILE";CHR$(13);:
     ¬GOSUB1750
1600 CLOSE6
1610 PRINT"DRIVE ";DD$;" HAS ";AA$
1620 GOTO 1060
1630 PRINT"↓r̲SHUT DOWN? ";:GOSUB1690
1640 IF S$="N" GOTO 1060
1650 CLOSE 15
1660 END
1670 :
1680 REM:  GET SUBROUTINE
1690 POKE167,0
1700 GETS$:IFS$=""THEN1690
1710 PRINTS$
1720 POKE167,1
1730 RETURN
1740 :
1750 REM:  READ ERROR CHANNEL
1760 INPUT#15,EN$,EM$,ET$,ES$
1770 IF EN$="00" THEN RETURN
1780 PRINT"DISK ERROR #"EN$"  "EM$"  ¬
     ¬"ET$"  "ES$
1790 INPUT" CONTINUE? >>>N<<<";IN$ :
     ¬IF IN$="Y"THEN RETURN
1800 CLOSE 5:CLOSE 6:CLOSE15
1810 END
1820 :
1830 REM:  OUTPUT TO PRINTER
1840 PRINT"↓r̲PRINTING ENDS THE PROGRAM"
1850 INPUT"HAVE YOU MEMORIZED FILE ¬
     ¬FIRST?>>>*<<<";S$:IF LEFT$(S$,
     ¬1)<>"Y"THEN1060
1860 INPUT"↓ENTER YEAR (EG 1981)>>>*<<<"
     ¬;YR
1870 REM IF FEB NOT IN LEAP YR,ND=28
1880 IF YR/400=INT(YR/400) GOTO 1910
1890 IF (YR/100=INT(YR/100))AND MN=2 ¬
     ¬THEN ND=28 :GOTO 1910
1900 IF (YR/4<>INT(YR/4))AND MN=2 THEN ¬
     ¬ND=28
1910 GOSUB 2670
1920 PRINT"↓r̲GET PRINTER READY, THEN ¬
     ¬HIT ANY KEY ";:GOSUB1690
1930 OPEN 3,4
1940 PRINT#3:PRINT#3:PRINT#3:PRINT#3
1950 PRINT#3,CHR$(1)+"             ¬
     ¬MEMORY CALENDAR"
1960 LZ=INT((40-LEN(AA$+STR$(YR)))/2)
1970 A7$=""
1980 FOR I=1 TO LZ
1990 A7$=A7$+" "
2000 NEXT I
2010 PRINT#3,CHR$(17)+CHR$(1)+A7$+AA$ ¬
     ¬+STR$(YR)
2020 PRINT#3:PRINT#3:PRINT#3
2030 CLOSE 3
2040 FM$="AAAAAAAAAA "
2050 FT$=""
2060 FOR I=1 TO 7
2070 FT$=FT$+FM$
2080 NEXT I
```

```
2090 OPEN3,4,2
2100 PRINT#3,FT$
2110 CLOSE3
2120 :
2130 OPEN3,4,1
2140 FOR I=0 TO 6
2150 PRINT#3,WD$(I);CHR$(29);
2160 NEXT I
2170 PRINT#3
2180 :
2190 IF WD=0 GOTO 2280
2200 FOR I=ND TO 1 STEP -1
2210 FOR J=1 TO 8
2220 A$(I+WD,J)=A$(I,J)
2230 A$(I,J)=CHR$(160)
2240 NEXT J
2250 NEXT I
2260 ND=ND+WD
2270 :
2280 FOR I=1 TO 36 STEP 7
2290 FOR J=1 TO 8
2300 FOR K=I TO I+6
2310 IF K>ND THEN PRINT#3,CHR$(160);CHR$
     ¬(29); :GOTO 2330
2320 PRINT#3,CHR$(17) +A$(K,J);CHR$(29);
2330 NEXT K
2340 PRINT#3
2350 NEXT J
2360 NEXT I
2370 :
2380 PRINT#3
2390 CLOSE 3
2400 PRINT"ñ↓↓↓↓rFINISHED"
2410 GOTO 1650 :REM END PROGRAM
2420 :
2430 REM  INPUT SUBROUTINE
2440 IN$="":IFZ9$<>""THENPRINT"? ";Z9$;:
     ¬POKE167,0:IN$=Z9$:Z9$="":GOTO2460
2450 PRINT"? ";:POKE167,0
2460 GETZ$:IFZ$=""THEN2460
2470 IF Z$=" " THEN Z$=CHR$(160)
2480 IFZ$=CHR$(13) OR Z$=CHR$(141) ¬
     ¬THENPRINT" ":POKE167,1:RETURN
2490 IFZ$=CHR$(20)THENONSGN(LEN(IN$))+1G
     ¬OTO2460,2550
2500 Z8=ASC(Z$)
2510 IF Z8=44 OR Z8=58 OR Z8=22 THEN ¬
     ¬Z$="" :REM ELIMINATE DISK-PRINTER ¬
     ¬TROUBLES
2520 PRINTZ$;:IN$=IN$+Z$
2530 IFLEN(IN$)>=LLTHENGOSUB2560:
     ¬PRINT" ":POKE167,1:RETURN
2540 GOTO2460
2550 PRINTZ$;:IN$=MID$(IN$,1,LEN(IN$)-1)
     ¬:GOTO2460
2560 FORZ9=LEN(IN$)TO1STEP-1
2570 IF (MID$(IN$,Z9,1)<>" ")AND(MID$(IN
     ¬$,Z9,1)<>CHR$(160)) GOTO 2610
2580 Z9$=RIGHT$(IN$,LEN(IN$)-Z9)
2590 IN$=LEFT$(IN$,Z9-1)
2600 Z9=1
2610 NEXTZ9:RETURN
2620 :
2630 REM:  TEST IF GARBAGE COLLECTION ¬
     ¬NECESSARY
2640 IF FNFR(X) < (LL*LL)/2 THEN ¬
     ¬PRINT"↓rONE MOMENT PLEASEr":
     ¬  Q=FRE(0)
2650 RETURN
2660 :
2670 REM FIND WHAT DAY OF WEEK FIRST IS
2680 CY=YR :MP=MN-2
2690 IF MP<1 THEN MP=MP+12 :CY=CY-1
2700 YY=CY-INT(CY/100)*100
2710 CC=INT(CY/100)
2720 WD=YY+INT(YY/4)+INT(CC/4)-2*CC+1+IN
     ¬T(2.6*MP-.1999)
2730 WD=WD-INT(WD/7)*7
2740 RETURN
2750 :
2760 REM DISPLAY DAY ROUTINE
2770 INPUT"↓DISPLAY WHICH DAY";DY
2780 PRINT"___";A$(DY,1)
2790 FOR I=2 TO 8
2800 PRINT I-1;A$(DY,I)
2810 NEXT I
2820 PRINT"↓HIT ANY KEY TO CONTINUE ";
2830 GOSUB 1690
2840 GOTO 1050
2850 :
```

©

Crash Prevention For The Pet

Elizabeth Deal
Malvern, Pa.

This article is for beginners in machine code programming and for those who use machine code subroutines from various sources. It describes several reasons for crashes and how to avoid doing things that make it crash. It is geared mostly to the owners of the upgrade-ROM Pets (Basic 3.0), but some ideas should be useful to all Pet owners.

I define a "crash" as a situation where all control over the Pet is lost. It is an error condition of such severity that the cursor disappears, the keyboard does not work and you have to pull the plug. I also include in the definition of a crash a symptom of Pet behaving "silly", when, for instance, simple commands, like LIST or RUN lead to SYNTAX ERROR condition or a display of monitor registers.

I have heard people imply that these crashes are caused by a defect in the Pet. It is my experience that the overwhelming majority of such crashes are due to a defect in programming.

Let me illustrate some crashes by working in BASIC. These simple illustrations will show what can go wrong and why.

(1) Type WAIT 0,1. Since location 0 always contains 76 the Pet is made to wait forever for a 1. The stop key doesn't work while the Pet is waiting. You can now turn the Pet off and on or use the Butterfield procedure to regain control. (I strongly recommend that you build or buy an uncrashing device - Compute #1, p.89).

(2) Type in or load a very short program, two or three lines is enough. Now **POKE41,7,** and type RUN. You'll get SYNTAX ERROR. Type LIST and you'll get garbage. POKE 41,4 and all will be well again.

Such errors, as silly as they look, are very easy to make, even in Basic. If your variables are undefined, if you failed to add a constant to some address, etc. you will crash.

(3) Type POKE 81,15. Any value different from 76 will do. Now type PRINT PEEK(81) or PRINT FRE(0). The register display in this case tells you where the break occurred and that the Pet doesn't know where to go. Locations 81-83 contain a jump instruction to evaluate functions. Poking wrong values into 81-83 destroys Pet's ability to handle functions of which PEEK and FRE are just two examples. The Pet is alive at this moment and so long as you use no functions everything will work quite well. If you do use functions you will not recover from this sort of a crash even by the Butterfield procedure which preserves memory. Either power off or type POKE 81,76 to get things back to normal.

(4) Type FOR J = 112 TO 118:POKE J,42:NEXT.

The Pet is gone. Reset by the Butterfield procedure. The Pet will work in the monitor mode but not in BASIC mode. You can save the program that caused such a crash using the monitor. But if you exit the monitor by "X" and give a BASIC instruction, like LIST, the Pet will crash again. The only solution is to pull the plug. The reason is that locations 112-118 are one of many vital links between the monitor and BASIC. Destroying the contents of 112-118 destroyed Pet's ability to understand BASIC altogether. It is possible to regain BASIC using a method written by Robert Lando and shown to me by Mr. Wachtel. This method consists of copying the entire contents of the ROM CHRGET routine to locations 112-135 immediately after changing the SP value in the Butterfield procedure (hex: from \$E0F9-\$E110 to \$70-\$87).

I am grateful to Jim Butterfield for showing me those locations that are crucial for supporting BASIC. If the contents of these location are disturbed in any way, only restarting the Pet will allow you to regain control. They are, in decimal, USR vector (0-2), various indicators (13-15), string descriptors (19-21), start of BASIC program (40-41), top of the PET (52-53), garbage yardstick (80), and jump vector for functions (81-83). Further, interrupt system at 144-145, CHRGET routine at 112-135 and location 1024 which must be zero for BASIC to run from its normal position. If the CMD command is on all output goes elsewhere thus you can't communicate with the Pet. This list shows the most important locations. There are many others that if disturbed will cause unrecoverable crashes. Note again, that the Butterfield procedure will let you see what went wrong and permit you to save the offending program. But to be able to use BASIC commands you may have to reset the Pet completely.

As you can clearly see, we caused a lot of trouble without ever leaving BASIC. When you work with machine code, the most frequent reasons for crashing will be of the WAIT variety, jumping or branching to wrong locations and infinite loops. You will recover by the Butterfield procedure and prevent further crashes by fixing the code.

But how can you prevent the hard crashes described above? I have run into a lot of such trouble while trying to adapt machine code subroutines written for an old Pet to my "new" Pet, often without knowing for which Pet the code was written. The most notorious offenders were those routines

that used the old Pet's input buffer (locations 10-89) for storage of variables and addresses. You can see that some of the most important locations in the new Pet are where the old Pet's input buffer is.

Many of the old Pet routines can be changed to run, without crashing, on the new Pet, if you're careful. Many of them do not use any BASIC ROM routines, thus we do not have to bother with that translation. Most of them use zero page addressing and use locations 10-89. It's a good idea to learn just enough about the machine code instructions of the Pet to be able to spot the addresses. You can then find locations in the new Pet that are safe and change the program accordingly. Some locations I think are safe are 1-2 if main program has no USR call, 15-16, 84-89, 60-63 if you're not using DATA lines in the main program, 177-195 if tape is not used. And there are others. When a large block of contiguous locations is needed and 177-195 cannot be used, you will have to redo some of the coding in the following way. Determine the zero-page locations you need, and how many. Attach a bit of code at the beginning of a machine code routine to move the contents of the locations you're interested in into the first cassette buffer. Just prior to exit from the subroutine, move the contents back from the cassette buffer to page zero. When control returns to BASIC nothing has been disturbed and the Pet cannot crash. Please note that inserting more code may require some changes in absolute addresses in the routine itself. This is not difficult to do.

We have now taken care of those problems where machine code routines can destroy important Pet pointers, BASIC connection and so on. But there is another problem, that of strings from the BASIC program destroying machine code routines placed at the top of the Pet. Michael Riley gave me a simple solution: after poking the appropriate top of the Pet pointers (52-53) it is necessary to either say CLR or RUN-next line for all pointers to be set. So if your machine code routine does not perform this operation, you can do it in direct mode or within the BASIC program. Just make sure you do not initialize any variables needed by the program before the CLR or RUN line.

I find it helpful to go over a routine looking for what might cause the Pet to crash and how to prevent it. Some routines work only with a main program they were designed for. They may not work for your calling program because of different BASIC commands you may use (see point 3 above). Adjust them, so they are as general as possible and you'll never have to worry about crashing, no matter what the calling program contains. There are many very useful routines in the press that are worth the trouble of conversion. The side benefit of making changes in well written programs you see in the magazines is that you can learn a lot from them. I did.

References:

1. Jim Butterfield, Compute and personal communication
2. Michael Riley, personal communication
3. Nick Hampshire, The Pet Revealed, Computabits, England
4. Anselm Wachtel, Compute#2 and personal communication.

©

Machine Language Printer Command

Zoltan Szepesi
Pittsburgh, PA

While working on a Machine Language program, it could be advantageous to be able to give a command to the printer in ML instead of going back to BASIC and returning to the Monitor or to some other ML program.

The program, which follows, substitutes the BASIC command:

OPEN4,4:CMD4

and at the end:

PRINT#4:CLOSE4

The program can be loaded anywhere there are 20 bytes free address. Starting at $XXXX:

```
XXXX
START       A9 04   LDA -$04 File and
                    device number
            85 B0   STA z$B0 The out-
                    put to CMD is in $B0
            85 D4   STA z$D4 The
                    device number is in
                    $D4
            20 BA
            F0      JSR OPEN IEEE
            20 2D
            F1      JSR TEST IEEE
            20 D2
            FF      JSR WRT
            00      BRK
XX(X + 1)X  4C CC
            FF      JMP RESTORE I/O
            00      BRK
```

When we want to start the printer, we have to type: .G XXXX (or working with some other program, e.g. Moser's Assembler, print: RUN $XXXX), and to close the printer we have to type: .G XX(X + 1)X.

The screen does not show what the printer prints, but we can give the necessary commands through the keyboard as if the printed text would be on the screen. This way we can continually print out the dumping of a complete ML program or the Assmelbe List.

©

ODDS & ENDS
ON PET/CBM files

Jim Butterfield

Writing data to a file is quite straightforward: OPEN the file, PRINT# to the file as many times as you like, and then CLOSE the file. Reading is pretty easy, too: OPEN the file, INPUT# until the file has given you all its data, and CLOSE the file.

You can also read a file by using GET# instead of INPUT#. The GET# command is especially useful for checking out a file and seeing what's on there. Here's why: INPUT reads everything to the end-of-line; trims the line it has received by taking off leading and trailing spaces and quotation marks, and then scans the line, converting numbers to internal representation, etc. All very handy; but if something goes wrong, you'll want to use GET to look at the characters one at a time.

When you're reading a data file, keep watching variable ST. It will normally be zero; at the time you read the last value it will change to a value of 64. Any other value means you have a read error.

The ST indicator works slightly differently on the original PET ROMs. It does not go to 64 at the time you read the last value; instead, it switches to 64 only when you try for the following value — the one that isn't there. You can handle this with careful coding. But you'll be better off to upgrade your ROM set so that your programs will be compatible with newer machines.

End-of-file on a disk read is shown in ST, but errors are not. On upgrade (2.0) ROMs, use the command channel (Secondary Address 15) to ask the disk unit how it's doing. Newer ROMs give you disk status variables called DS and DS$ to make it easy to check errors.

PRINT# sends to a file in almost exactly the same way that PRINT sends to the screen: as a group of ASCII type characters. INPUT# receives from a file the same way that INPUT receives from the keyboard/screen. Make sure that what you send to a file will be seen as a good input when it comes back.

Let's pick up more detail on the previous item. If X is five and Y is two, and you say PRINT#3,X;Y the file will be written as:

(space) 3 (space) (space) 2 (space) (return)

Think about it. What would happen if you typed the above sequence in response to an INPUT? Answer: PET would see a single number — not two — whose value is 32. That's exactly what would happen if you later tried to read with an INPUT#. Solution: say PRINT#3,X : PRINT#3,Y and the two numbers will be neatly separated with a RETURN character.

For exactly the same reasons. You shouldn't say PRINT#3,X,Y ... you'll put more spaces on the file, but you won't solve the problem.

Best practice: Use a separate PRINT# statement for each variable.

Early PETs — everything before ROM 4.0 — write both RETURN and LINEFEED at the end of a line. The RETURN is handy — in fact, it's vital — but the LINEFEED can give trouble and should be taken out. You do this by coding something like:

PRINT#3,X;CHR$(13);

The CHR$(13) is the RETURN character. Don't forget the semicolon at the end, or PET will stick another RETURN and LINEFEED behind the whole thing and you'll have a mess.

On 4.0 and subsequent ROMs, the LINEFEED will normally be supressed, and you can go back to PRINT#3,X. Cassette tape files have a special feature which avoids writing the LINEFEED character.

Programs using cassette tape files are quite easy to convert to disk. To open a file for writing change, for example, OPEN 1,1,1, "INVENTORY" to OPEN 1,8,3,"0: INVENTORY,S,W". The 8 means device 8, usually disk; the 3 is an internal disk channel number (pick anything from 3 to 14); 0: means drive zero, and ,S,W means we plan to Write a Sequential file. Everything else for writing the file can remain as before (PRINT# and CLOSE), so long as you watch to make sure you don't write LINEFEEDs with your PRINT#.

Switching over to disk for reading a file is even easier. Change OPEN 1,1,0,"FILENAME" to OPEN 1,8,3,"FILENAME" and you're in business.

In cutting over from tape to disk files, it doesn't hurt to add error checking, of course — secondary address 15 or variables DS and DS$, depending on your system.

Never use the TAB function in writing to a file — or to the printer, for that matter. PET will try to calculate the proper place on the screen for the information — and then sends that type of information to the file. It almost invariably botches the job.

Make sure that any file you write is always closed properly. It's all too easy to write a program that stops or goes into a special routine in certain cases — leaving a file open forever.

Get into the habit of protective CLOSE statements. It's perfectly allowable to say CLOSE 1 even if you're not sure that file number 1 was ever opened. And it doesn't hurt.

Don't forget that you can use a variable to indicate the logical address you want to use. You can say, PRINT#J . . . and if J is one, you'll send to logical device number one, etc. This is a very effective way to split a file into several smaller files.

Remember, too, that you can open the screen as a file (it's device 3), so that you could send some things to the screen and others to disk. ©

Three Pet Tricks

John F. Garst
Department of Chemistry
The University of Georgia

This magazine and others have published numerous PET programming methods that are not evident from reading the documentation provided by Commodore. Sometimes there is a subtle way of using what Commodore tells about. Then there are those things Commodore forgot to mention. Here are three tricks that I use.

On-line Remarks

With a PET, on-line remarks are made as follows, according to Commodore.

```
200 :        GOSUB 500:REM OUTPUT
```

Both the statement-delimiting colon and the REM statement must precede the remark. Other implementations of BASIC allow the use of an apostrophe in the place of both of these, making programs more readable.

```
200 :        GOSUB 500     'OUTPUT
```

The PET actually allows the construction just given! However, the PET does not use the apostrophe as an abbreviation for REM. In fact, the PET allows the following construction.

```
200 :        GOSUB 500      OUTPUT
```

Nonnumeric character strings that follow the target line number of a GOTO, GOSUB, or THEN statement are ignored. This is not true for all other kinds of statements. Nonetheless, it is convenient to be able to tag GOSUB statements with labels reminding the reader of the nature of the target subroutine.

Flashing Cursor For Get

Several notes have appeared showing how GET can be used to advantage instead of INPUT. Deal's recent article (COMPUTE, vol. 1, issue 6, p. 98) illustrates a routine related to some I have used.

Deal uses a BASIC method to flash the cursor. According to C. S. Donahue and J. K. Enger, "PET/CBM Personal Computer Guide," OSBORNE-McGraw-Hill, Berkely, CA, 1980, p. 106, there is a POKE address and a value that turns on the PET's cursor under control of its OS. The location is 548 for the "old" ROM set (version 2.0) and 167 for the "new" ROMs (version 3.0). I assume that the newest (4.0) ROMs use the same address as the 3.0.

The values to be POKEd are 0 to enable the flashing cursor and 1 to disable it.

```
100 POKE 167, 0 (turn on cursor)
110 GET A$
120 IF A$ = "" GOTO 110
130 POKE 167,1 (turn off cursor)
```

This seems to work fine. I have had no problem with its actual operation, but I have had a few "flying cursor" residues (reverse blanks) left here and there at unexpected places after having used these POKEs. I don't know whether these were from my program bugs or from something in the operating system that was upset by the POKEs.

Pretty Printing

The PET system gobbles up spaces that may be left between the line number and the first character of a statement being entered, with the result that all statements in a PET BASIC program are left-justified. One of the features of a readable program is the use of blank lines and statement indentation to emphasize the logical structure of the program. This is "pretty printing" (see P. Nagin and H. F. Ledgard, "BASIC With Style", Hayden Book Company, Rochelle Park, NJ, 1978, or J. M. Nevison, "The Little Book of BASIC Style," Addison-Wesley, Reading, MA, 1978).

By now it is well known that spaces can be inserted at the beginning of a PET BASIC line if a colon (":") is typed in the first or second space following the line number.

```
100 :    FOR I = 1 TO 10
110 :          X = X + 1
120 :    NEXT I
```

What may not be so well known is that there is at least one restriction on this usage. A DATA statement that is not preceded immediately by a colon is ignored! Thus, the following will not work.

```
110 :      READ X, Y
110 :      DATA 1, 2
```

Instead, this can be used:

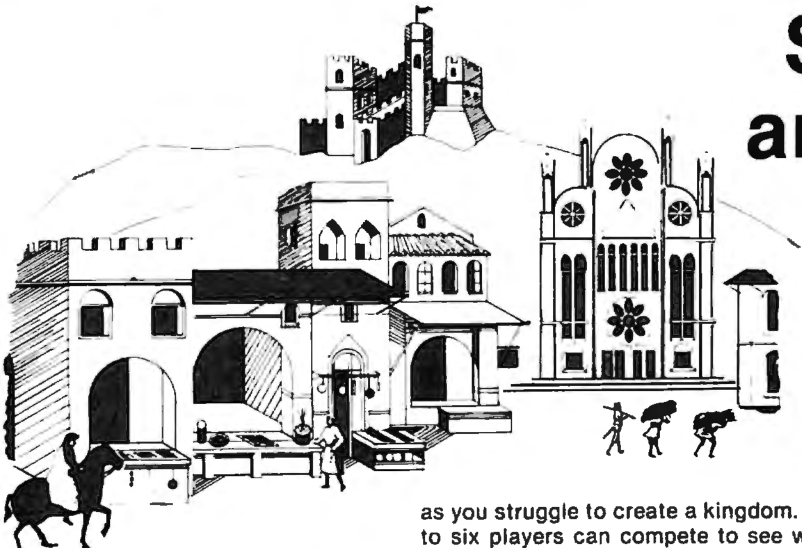
```
100 :      READ X, Y
110 :      :DATA 1, 2
```

©

Captivate Yourself.

Santa Paravia and Fiumaccio

The most captivating and engrossing program ever made for the PET*



SANTA PARAVIA AND FIUMACCIO
Become the ruler of a medieval city-state as you struggle to create a kingdom. Up to six players can compete to see who will become the King or Queen first. This program requires a PET 16K. **Order No. 0175P.$9.95.**

It is the dawn of the 15th Century; you rule a tiny Italian city-state. Your goal: The Crown!

Up to six players can compete as rulers of neighboring cities. You control the grain harvest, feed your serfs, set tax rates, dispense justice and invest in public works.

The future of your realm will depend on your decisions. If they are wise, your city-state will grow and you will acquire loftier titles. If your rule is incompetent, your people will starve and you may be invaded by your neighbors.

How will *you* rule your kingdom? Will you be an enlightened leader—or an unscrupulous despot? Only you can answer that question—with Santa Paravia and Fiumaccio.

CHIMERA

CHIMERA If you think the legendary Chimera was hard to handle, wait until you try the Chimera package. Included are:
- **Reflex**—Round and round the little white ball rolls. Only fast reflexes can guide it into the center of the maze.
- **Dragon**—You'll have to shoot down those pesky, fire-breathing dragons with your cannon. If you succeed your castle will be safe, if not it will mean a call to your fire insurance company. For one player.
- **Dungeon**—A very punctual guard comes down to the dungeon every day to torture you. This means that you have only thirty seconds to dig your way under the castle to freedom. For one player.
- **Dragon Hunt**—You must go forth and slay a fire-breathing dragon. The only thing that will protect you from the flames is your shield, if you know when to use it. For one player.
- **Dropoff**—You must make your opponent's men "dropoff" the board by moving and firing your own men. For one or two players. **Order No. 0110P. $9.95.**

PET DEMO I

PET DEMO I You can give yourself, your family, and your friends hours of fun and excitement with this gem of a package.
- **Slot Machine**—You won't be able to resist the enticing messages from this computerized one-armed bandit.
- **Chase**—You must find the black piece as you search through the ever-changing maze.
- **Flying Pheasant**—Try to shoot the flying pheasant on the wing.
- **Sitting Ducks**—Try to get your archer to shoot as many ducks as possible for a high score.
- **Craps**—It's Snake Eyes, Little Joe, or Boxcars as you roll the dice and try to make your point.
- **Gran Prix 2001**—Drivers with experience ranging from novice to professional will enjoy this multi-leveled race game.
- **Fox and Hounds**—It's you against the computer as your four hounds try to capture the computer's fox.

For true excitement, you'll need a PET 8K. **Order No. 0035P $7.95.**

DOW JONE$

DOW JONES Up to six players can enjoy this exciting stock market game. You can buy and sell stock in response to changing market conditions. Get a taste of what playing the market is all about. Requires a PET with 8K. **Order No. 0026P $7.95.**

We Guarantee It!

Instant Software Guarantee

OUR PROGRAMS ARE GUARANTEED TO BE QUALITY PRODUCTS. IF NOT COMPLETELY SATISFIED YOU MAY RETURN THE PROGRAM WITHIN 60 DAYS A CREDIT OR REPLACEMENT WILL BE WILLINGLY GIVEN FOR ANY REASON.

Prices subject to change without notice.

TO ORDER

SEE YOUR LOCAL INSTANT SOFTWARE DEALER

OR

call now Toll-Free 1-800-258-5473

Instant Software™

* A trademark of Commodore Business Machines

PETERBOROUGH, N.H. 03458
603-924-7296

Review

PASCAL On The Pet

A. J. Bruey
Jackson, MI.

The use of Pascal is becoming more widespread every month if the number of magazine articles and advertisements about Pascal and Pascal products can be used as an indication. Pascal began as a teaching language. It is recognized to be an excellent first language for programming students because the structured features of Pascal make it possible for the student to learn efficient programming techniques. Now more and more business applications written in Pascal are beginning to appear on the market.

I had been studying Pascal from a manual with no computer to try it on. Then Abacus Software (PO Box 7211, Grand Rapids, MI 49510) announced the availability of a PET and APPLE version of Tiny Pascal. This version is based on the Yuen/Chung series in the September-November, 1978 issues of Byte Magazine. The Abacus Software system is produced through a licensing agreement with SuperSoft of Champaign, Illinois.

The Pascal Package

The Pascal package contains three programs:

A. A line editor for developing and maintaining Pascal source programs.

B. A compiler for compiling the source code into p-code.

C. An interpreter to interpret and execute the p-code.

The editor and compiler are written in BASIC and the interpreter is written in 6502 assembler. Source programs and p-code files can be saved on either disk or tape in the PET version. The APPLE version requires disk operation.

Writing a Pascal Program

First the Pascal source program is written using the line editor. The compiler is used to convert the source code to p-code. The p-code is then run interpretively using the interpret program. The p-code program executes much faster than a BASIC Program performing the same function.

Advantages of this system

Inexpensive. At $35.00 for the disk version and $40.00 for the tape system, it's a good buy for anyone who wants to try Pascal.

Structured constructs. This version contains all the structured features for which Pascal is noted:

FOR...DO
WHILE...DO
REPEAT...UNTIL
CASE
IF...THEN...ELSE

Simple to use. Excellent documentation including both the source code and p-code for two sample programs. Step-by-step operating instructions make it easy to learn.

Abacus Software provides excellent customer support.

Disadvantages

Like other "tiny" language implementations, this version is an integer-only implementation. The only data types are integer and integer array.

There are no built-in functions. None of the usual Pascal functions such as SQR (square) and SQRT (square root) are available.

Rather slow. In the limited testing that I've done, I've found that the compiler takes three to four seconds to compile each line of source code into p-code.

Poor I/O facilities. There is no provision for disk or tape input or output during program execution. There is also no way to direct program output to a printer.

A Sample Program

```
 1 : [SQRT - INTEGER SQUARE ROOT]
 2 : CONST CR=13;
 3 : VAR X,NUMBER,MEMORY,COUNT,A:INTEGER;
 4 : FUNC SQRT(X);
 5 : BEGIN
 6 : MEMORY:=1;
 7 : A:=0;
 8 : WHILE X>=0 DO
 9 :    BEGIN
10 :       X:=X-MEMORY;
11 :      A:=A+1;
12 :      MEMORY:=MEMORY+2;
13 :     END;
14 :  SQRT:=A-1;
15 :  END;
16 :  BEGIN
17 :      NUMBER:=1;
18 :        WHILE NUMBER>0 DO
19 :           BEGIN
20 :             WRITE('ENTER A NUMBER ');
21 :             READ(NUMBER#);
22 :             COUNT:=SQRT(NUMBER);
23 :             WRITE(CR,'SQUARE ROOT IS ',COUNT#,CR);
24 :           END;
25 :  END.
```

The listing shows a sample Pascal program that was developed and run under this system. It was the first Pascal program that I wrote and thus the coding is probably far from optimum. It is an integer square root routine based on the method described in my previous (November, 1979) MICRO article "Performing Math Functions in Machine Language". The reader may either refer to that article or may discover the algorithm for himself by following through the coding.

A brief description follows for those of you who are not familiar with Pascal.

Line 1: A remark line. Not executed.

Line 2: Defines the carriage return character. All constants must be defined in a CONST section.

Line 3: Declares all variables that will be used in the program. All variables must be declared in the VAR section.

Lines 4 to 15: This function is the actual square root routine. X is the dummy argument which is passed to the function from the calling statement in line 22.

Lines 16 to 25: This is the main section of the program. Line 20 prompts for an integer, line 22 calls the function, and line 23 displays the answer.

Recall that this is an integer Pascal. You will always get just the integer part of the answer. For example, you will (correctly) get 25 as the square root of 625, but you will also get 25 as the square root of all numbers from 626 to 675. To get more accuracy, you must develop multiple precision routines just as you would have to do in machine language.

The listing was printed using the line editor. Other functions of the editor are append, delete, list, change, replace, load, and save.

Conclusion

After weighing the advantages and disadvantages listed above and using the system for a few days, I have concluded that this program is well worth the price. It is quickly becoming one of my favorite software packges. Those of you who are not used to structured languages will find it interesting to solve program design problems without the use of a GOTO statement. ©

Review
The PEDISK From CGRS Microtech

Dr. J. A. Dilts
Department of Chemistry
University of North Carolina
at Greensboro, Greensboro NC 27412

The addition of a disk to ones microcomputer system is a need that becomes evident after a short period of loading programs from cassette tape. Any attempt at even modest data handling amplifies this need. For the owners of the original 8K PET computer, this presents a problem in that Commodore's disk system is incompatable with the original ROM operating system. For those of us who purchased the early PETs, there is an alternative to updating the ROM operating system or selling our old PETs in order to add disk capability. The CGRS PEDISK goes beyond offering a high speed means of loading programs and saving data in that it also provides access to the S-100 buss. This offers expansion not only in terms of memory, but a host of other possibilities such as analogue to digital boards, modem boards, etc.

In the standard configuration the CGRS PEDISK system offers two free S-100 slots. It is possible to add another S-100 connector at additional cost. The 5.25 inch drive uses soft-sector diskettes. The format is the IBM 3740 standard. This provides 80K of storage per drive. A total of 4 drives can be handled by the S-100 disk controller board.

Although this system was originally designed to operate with the original PETs, it is now available for the newer 16 and 32K versions.

The Disk Operating System

The disk operating system provided by CGRS is partially in ROM, but is basically a RAM oriented DOS. Although there is an overhead with such an approach (about 2K of memory is taken up by the DOS), the advantage of having the DOS in RAM rather than ROM is that it is possible to make modifications, be they up-dates or correction of bugs.

The DOS is initialized by a SYS call to a boot starting in ROM at hex B000. The disk operating system is loaded into the top 2K of RAM and is subsequently protected.

Loading and saving programs is accomplished by preceeding the usual BASIC command with a " ! ", for example:

!LOAD"program name:0"

where 0 is the number of the disk drive. Unlike program names on tape, program names on disk are limited to 6 characters. Provision is made for renaming programs or data files. As with any disk system, duplicate programs are not allowed.

One of the most appealing features of the DOS lies in its file handling capabilities. Files may be opened as serial access or indexed types. In the first case, files are written and read from the first entry to the last. In the case of indexed files, any record may be written and/or read back in any order. Once a file has been opened for writing purposes, it may be reopened for reading and/or editing. This allows one to examine a file record by record and perform editing (rewriting) on a record by record basis. There are two reserved variables for checking on successful file write and on encountering the end of file. File closing is automatic with the command !CLOSE or when the system is initialized. Up to four disk files may be opened at one time and each maintains its own index counter through a common reserved variable. Commands normally used in dealing with tape files are proceeded by "!" in the case of disk files, i.e. !INPUT F$ Z$ where F$ is the string containing the file name.

The command !SYS transfers control to the disk monitor. Here commands can be entered as a single character without using the return key. In this mode keying in "H" will list the currently available single key commands. It should be noted that in this mode, any current BASIC program will remain intact and pressing "R" will return one to the BASIC operating mode.

Commands in the DOS monitor mode include DUMP which will cause a formatted dump of either memory or disk sectors, GO to execute a machine language program, KILL to delete a file from the disk, LOAD to load a program without execution, MEMORY, a command to examine and change locations in memory, PRINT the volume table of contents of the disk, RENTER the BASIC operating system, SAVE to save a BASIC or assembly language routine and UTILITY to access routines to compress disk files, copy disks, read or write a disk sector or initialize a diskette. All of these commands are actuated by typing in the first letter of the command.

In short, the DOS affords a very neat package especially with respect to data file manipulation. Documentation is adequate, especially if you have had some experience with other disk operating systems. As with any new system, some experimentation will be necessary for the user to become familiar with all the features of the system. Not all of the commands mentioned under the monitor will work from the BASIC control mode but this minor bug will doubtless be corrected in future editions of this program. This is a great advantage in a RAM oriented DOS. I found the software to be relatively free of BUGs.

Because of the 2K overhed in RAM, it would be advisable to have a minimum of 16K RAM for such a system. The potential user should also be aware that if a change in memory size occurs, an updated version of the operating system must be obtained.

In brief, the CGRS PEDISK offers in a neat package both disk capabilities and access to S-100 buss boards. This latter point has been most important in terms of applications of this reviewer. ©

Review

A Disk Operating System for the CGRS PEDISK

Dr. J. A. Dilts

When I first saw the PEDISK in operation nearly three years ago, I was not overly impressed. The capabilities at the time were limited to saving and loading programs with no data file handling. It was not long before this original disk operating system, KMMM (by Wilserv Industries, PO Box 115 Haddon Field, NJ 08033) had been expanded to include full file handling abilities.

The great advantage of any operating system in RAM is relative ease of updating and incorporating improvements. When working with a disk system, the time spent in loading software is not a major problem.

The basic configuration involves initialization via a SYS call to a ROM based boot which loads the DOS into the top end of memory. The user must specify his memory configuration when ordering the DOS software for, although a 24K version will work on a 32K configuration, the top 8K will be unavailable for normal basic programs.

After initialization, the user has a chance to specify a change in the date or his configuration (i.e. number of drives, printer, maximum number of files, etc.).

After any changes have been made, the user may return to the BASIC operating mode or to the DOS monitor. The DOS resides in about 3K of memory so a good minimum memory to use with this system is 16K.

The usual BASIC commands such as LOAD, SAVE, etc. are preceeded by a SYS 999 when used with the disk. This saves the user the task of remembering the address of the entry point of the DOS. If the second cassette buffer is being used for an assembly language routine, the 999 address can be replaced by the actual address of the DOS entry point.

All special disk commands can be executed from BASIC. Routines for printing the volume table of contents, compressing a disk, deleting a program, etc. are included here, but the routines are loaded from disk into low memory and may write over a resident BASIC program.

The volume table of contents gives address information on the disk as well as memory. It also provides the date of creation of the disk file and in the case of program files, how many times, and date of, updates.

Provision is made for re-naming files and altering the file load point. The copy/compress routine offers the capability for copying individual files or the total disk. Copying is possible with only a single drive. The format on the disk is the IBM 3740 standard and the capacity is about 80K.

Data files in the present version are sequential only. These may be opened as read or write files but not both and the number of sectors reserved for a file must be specified at the time the file is created. Provision is made when initializing a diskette for omitting the boot, thereby saving more room when only files will be stored. File commands are like tape file commands except they are preceded by the SYS 999 command. Closing a file does not automatically write an end of file mark; this must be done under program control before the file is closed.

Up to 9 disk files can be open at one time (or the maximum number specified at the time the system was initialized). When a disk is formatted without the bootstrap capability, the maximum number of files for that diskette is specified (from 3 to 67).

The only means of updating a file with the present version is to open a second file for writing (with a different name or on a different drive) then read from the first file and write the modified or added information on the second file.

One neat feature of this system is its ability to chain programs. When the SYS999 LOAD' file name' is executed from a BASIC program, the named program will load and run with variables from the first program intact as long as the calling program is at least one sector greater than the program.

Either the contents of a disk or the VTOC may be routed to a printer if the printer option is specified when the system is initialized. The printer must operate on the IEEE port and its specified address.

In the year that I have been using this current and an earlier version of the KMMM DOS, I have found it to be very flexible. It has offered an excellent solution for disk capabilities for PETs with old ROMs. ©

THE SBC GAZETTE

A Terminal For "KAOS" (Kim, Aim, OSI, Sym

Bruce Land
Baltimore, MD

A "terminal" is what you use to send messages to the computer and to receive messages from it.

KIM and SYM have a terminal built in -- a hex keypad to send messages, and a 6-digit hex LED panel to receive them. The arrangement is simple, economical (in initial cost, at least), and slow. Sooner or later, one tires of using only the onboard hex pad and 6-character LED display, and yearns for an ASCII keyboard and CRT display.

Rockwell's AIM has a keyboard, 20-character display, and 20-character printer, and Ohio Scientific's Superboard has a keyboard, video board, and RS232 output; even so, owners of these other popular 6502 systems sometimes want an external video board to display longer lines.

Many articles on how to attach different combinations of keyboards and displays have been published. Let's look at some of the pros and cons of different systems, and then at the one I chose. I believe the one I chose is, for a one-board system, about the most cost-effective method of obtaining a very versatile ASCII-plug keyboard input and a memory-mapped video output to a CRT display.

Of all the ways to obtain ASCII I/O, the simplest and perhaps the cheapest is to use a parallel-connected keyboard and a video RAM display. Hal Chamberlin, in "Software Keyboard Interface with a Pittance of Hardware" (Kilobaud, January 1978), discusses how to install an unencoded keyboard as a software scanned device connected to a PIA-type parallel input port. This uses a minimum of hardware, and not much CPU time. The OSI C1P and C4P, the Apple, the PET, and others use a similar method to connect their keyboards.

Chamberlin gives complete schematics and KIM software. Software for other 6502 systems would be very similar. The hardware will work with any port and should cost less than $30.

Don Lancaster announced the first KIM pseudo "video RAM" in Kilobaud (June 1977) and in Popular Electronics (July 1977). Complete schematics were published, and some software. Kits were marketed for about $35 by PIA Electronics, Inc., 1020 W. Wilshire Blvd., Oklahoma City, OK 73116. This system relied on the CPU to run the display, and while the CPU was busy elsewhere the video was blank. For continuous display it was necessary to write software to have the CPU maintain the display and run the program at the same time. A foreground/background type of operation is needed, and this can get quite complicated.

The amount of CPU time required for the Lancaster display varies, but you can get an idea from the hex keyboard scan and display of the basic KIM. There, about 20% of the CPU time is spent on I/O software. To use the Lancaster system, decide how much delay you can tolerate in keyboard response, how long you want to display, and how often you will scan the keyboard for an entry -- five times a second, ten, or more -- and write your software accordingly.

Anything you store in a true video RAM memory location will be output as a composite video signal and displayed. The display is refreshed with TTL logic, not CPU time. A software-scanned keyboard and a video RAM are the fastest way to make an entry and get an ASCII character displayed. A video RAM is about the only practical way to do animated graphics.

M.T.U., P.O. Box 12106, Raleigh, N.C. 27605, now sells a true video RAM for approximately $300, assembled and tested. The M.T.U. board has 320 X 200-bit resolution (64,000 bits, or about 8K of RAM), which is the highest I have seen.

The big disadvantage of a video RAM driven CRT display is the lack of softwre compatibility. Almost all, maybe 95% of the software published for KIM, AIM, or SYM, is built to run with the respective ROM-based monitor program. That means you will have to rewrite the I/O of the software to run with a parallel keyboard and a video RAM. If you expect to write or adapt most of your software, then this method is very attractive; if you don't want to write a lot of special I/O programs, you should think twice before going this way.

A "6502 Video Driver Routine" software package is available for KIM from Forethought Products, 87070 Dukhobar Rd., Eugene, OR 97402, (503) 485-8575. It furnishes cursor movement, line and page functions, scrolling, etc., and should save the good programmer some time. Video RAM cards are made by several other manufacturers: Matrox

(5800 G Andover Ave., Montreal, Quebec H4T 1H4, Canada, telephone (514) 735-1182) has several models from $225-$500; The Computerist (34 Chelmsford St., Chelmsford, MA 01824 (617) 256-3649) has one for $245.

You want hard copy? A popular hard-copy output device is a teletype, known to several generations of ham radio operators as a TTY. KIM, AIM, and SYM have built-in monitor routines for TTY's and other serial devices. (I get tired of writing KIM, AIM, or SYM. We need a symbol to refer to all three systems. Try KAS. Or we could add OSI, another popular 6502 system, and call it KAOS, pronounced "Chaos.")

A used TTY sells for $500 up, and will furnish readable, dependable, noisy, all-caps, 110-baud output. A TTY may also have a paper tape reader and punch for mass storage, but don't bother with it. The KAOS cassette tape storage is quieter, more reliable, and faster. The graphics capabilities of a TTY are very limited.

Other printers are available with parallel or serial I/O, grpahics capabilities, upper and lower case, and better print quality. Of course, they usually cost more. Among them are Centronics terminals, the Texas Instruments Silent 700, Decwriters, Diablo, Qume, etc.

The great advantage of a serial terminal is that it works directly with the KAOS ROM's; no RAM is required to run it, and software purchased for any of the KAOS systems will run as a "black box" --just hook it up (which brings to mind the simplicity of this operation for a serial device: only three wires are needed. Hook up signal in, signal out, and ground, and you're ready to go.)

Hard copy output is a real plus, but there is something anomalous in paying three or four times as much for a printer as for the computer that drives it. Anomalous it may be, but a $3000 Diablo dances nicely to the tune played by a KIM that only cost $245 four years ago when it was shiny and new.

A video terminal such as the Lear-Siegler ADM-3 has all the serial advantages of a TTY, but no hard copy. Telecommunications, Alexandria, VA 22303, (703) 683-4019, sells rebuilt Datapoint video terminals for $500 up. New terminals can be found from $750 to $3000. But why buy a $750 terminal for a $180 CPU? You can buy a complete PET or OSI computer for not much more!

The answer, of course, lies in your purpose. If you're going to use the computer occasionally, for no more than a few hours a day, then limited line length and readable print quality may be all you need. On the other hand, if you're going to do extensive word processing or software development, and will be looking at the display for hours at a time, you may be willing to pay a lot more for a sharper, cleaner display, with 80-character lines.

Even if hard copy will be needed at some stage, there are advantages in being able to work with a CRT screen up to the point of the print-out. A CRT screen displays text much faster than a TTY, displays it quietly, and does not waste paper.

The video RAM is one way to get ASCII I/O; another is with a serial video system, sometimes called a "glass TTY." This is a video board with a parallel keyboard port and a serial ASCII I/O port. You add:

(1) your own monitor, modified TV, or RF modulator with an oridnary TV'
(2) power supply;
(3) parallel ASCII keyboard;
(4) and some kind of case.

Now you have a serial video terminal with capabilities similar to those of the ADM-3.

These video boards come in all stages of completeness, price, and features. You can get RS232 or 20ma I/O; 32, 64, or 80 characters per line; upper case only, or up to 128 ASCII characters; all-TTL logic (a very old design), or ROM-based CPU; slow or fast; with or without keyboard; one to three power supplies required; $150 to $500. Be careful in your choice; a high price may signify an old, expensive design.

Any of the KAOS machines will think this kind of terminal is an ordinary serial terminal, and most software will run without any modification. Such systems can be purchased from many suppliers. I know of these:

Electronic Systems, San Jose, CA 95151 (408) 448-0800 ($200 for kit; keyboard needed);

Xitex Corp., 9861 Chartwell Drive, Dallas, Texas 75243, (214) 349-2490 ($175 kit; keyboard needed; or $375 for full kit including keyboard, case, etc.);

Electrolabs, Box 6721, Stanford, CA 94305, (415) 321-5601 ($239 A & T; keyboard needed);

Mostek Corp., 1215 W. Crosby Rd., Carrollton, TX 75006 (214) 242-0444 ($195; keyboard needed;)

Synertek Systems (who also make SYM), Box 552, Santa Clara, CA 95052 (408) 988-5600 ($389 - $450 complete);

Riverside Electronics Design, 1700 Niagara St., Buffalo, N.Y. 14207 (716) 875-7070 ($225 A & T; keyboard needed; $150 complete kit including keyboard.)

Netronics R & D, Ltd., 333 Litchfield Rd., New Milford, CT 06776, (800) 243-7428 ($149.95 + $3 postage.)

After much looking and reading, and several long-distance telephone calls, I chose the "Stand Alone ASCII/Baudot Computer Terminal" by Netronics R & D, Ltd. This unit will provide 64 or 32 characters per line -- 64 for TV direct or video monitor, and 32 for use with a modulator and plain TV. The baud rate is 110 or 300 ASCII, 45.45 or 74.2 Baudot. Output is either RS232 or 20 ma current loop (TTY "similar"). All printable ASCII characters are available (upper and lower case) as well as 32 special characters (Greek letters, symbols, superscripts, and graphic characters).

Complete cursor control is provided, including absolute and relative X - Y addressing. This allows low-resolution graphics and computed relative cursor jumps. At 300 baud you cannot do animation.

The Netronics video board has an on-board +5V regulator, and draws about 450 ma. If it is used with their keyboard, you supply +8VDC (or +5VDC) at 500 ma and 6.3VAC at about 50 ma (most keyboard inverter chips require -12V DC; the Netronics circuit eliminates the need for this supply. It uses a voltage doubler to convert the 6.3VAC to -12VDC for the keyboard encoder chip and the RS232 I/O levels.)

The video board mounts underneath the keyboard and both fit into the Netronics $20 keyboard case, leaving room for the necessary transformers and capacitors. When the keyboard and the video board are assembled and housed in the case, they provide full ASCII or Baudot input with some interesting extras, and everything needed for the output display except a monitor.

The Netronics documentation is a little on the light side; nevertheless, assembling the kit should be relatively easy for anyone with kit-building experience. The copper traces and pads are very small, so a small-tip, low-wattage soldering iron is a must. Take your time, and inspect each of the more than 1000 joints for proper solder flow and absence of solder bridges. There are many plated-through jumper holes in the board, and it is easy to insert a component in the wrong hole. The component numbers are marked on the board, but the jumper holes do not have a silkscreened outline around them as Heathkit boards do. If you have any doubt about the proper placement of a component, trace the schematic and follow the foil traces. (The first-time kit builder is advised to get some expert supervision in positioning the components. It's discouraging to have to back up.)

A good photograph showing correct placement of components on a completed board should be included with the documentation, but is not. The kit does not include an RS232 connector.

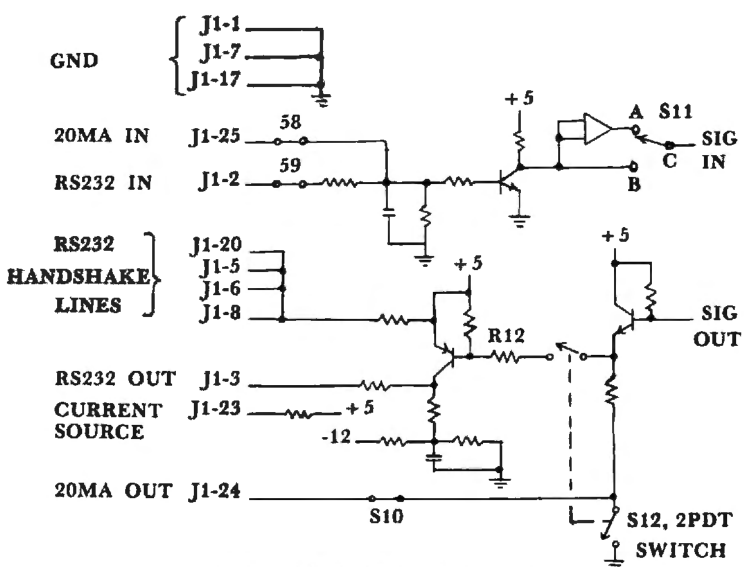
My group of five electrical engineers built 9 of these terminals. Five of the boards failed to work at first because of poor solder joints or misplaced jumpers. One board had a bent IC pin, and one had 3 jumpers missing. One, assembled by a good solderer with a known good board for reference, worked the first time it was hooked up.

A few modifications to the board might be considered. If you replace jumper S10 with a normally closed pushbutton switch, you can generate the BREAK command like a TTY.

Put a SPDT switch in place of J3-J4 on the keyboard, and you can switch easily between all-caps

with numbers (TTY mode) and typewriter mode, with both upper and lower case. When you are writing or running programs in BASIC you will probably find the all-caps mode most convenient. If you intend to do any word processing, you will find that ability to change easily to upper or lower case is very helpful.

If you think you may want to change from RS232 to 20 ma loop, install a SPDT toggle switch at S11, and a DPDT switch at S12. One pole of toggle switch S12 should be in series with R12, and the other replaces jumper S12. These switches permit you to change from one system to the other without changing 6 jumpers. (Fig. 1). For RS232, set switch S11 to position B, close S12, and use pins 2 and 3 for I/O. For 20 ma current loop, set switch S11 in position A, switch S12 open, and use pins 24 and 25 for I/O. Jumpers S8, S9, and S10 are installed as shown.



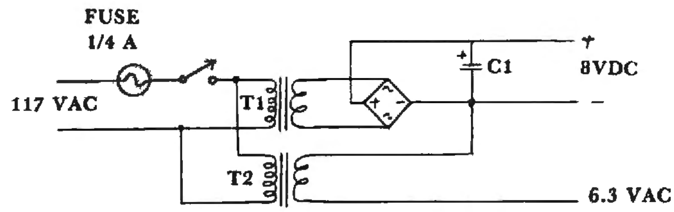
SWITCHES SHOWN IN 20 MA POSITION

I/O MODIFICATION FIGURE 1

Other lines in J1 will have to be connected to your terminal, but different computers and terminals may require different lines. KIM has a 20 ma current loop I/O, but the input to the terminal needs to be inverted. Set Switch S11 to Position B, and open Switch S12. Wire KIM A-1 to Netronics J1-1, KIM A-T to Netronics J1-24. Jumper KIM A-S (S acts as a current source) to A-U and wire A-U to Netronics J1-25. (You can use Netronics Pin J-1 jumpered to J1-25 as a current source; if you do this, then do not use KIM A-S.)

You could make a simple RS232 adapter for KIM to talk over, but that is another story. Pins J1-1-5, 6, 8, 17, and 20 are handshake lines for talking to a modem, and will not be used by KIM.

The power supply shown in Fig. 2 may not be ideal, but it works and fits inside the keyboard case.



POWER SUPPLY FIGURE 2

PARTS LIST

T1	Transformer 6.3 VAC @ 1.2 A	Radio Shack #273-050	$3.49
T2	Transformer 6.3 VAC @300 ma	Radio Shack #273-1384	2.49
D1	Diode bridge, 1A, 50 PIV	Radio Shack #276-1161	.79
6	Capacitor, 3300 uf, 35V	Radio Shack #272-1021	2.99

The Netronics kit has a few bad features. One is the lack of enough detail in documentation. The next may be only a personal idiosyncrasy, but I strongly prefer to use a complete set of IC sockets; Netronics provides sockets only for the 24- and 40-pin IC's.

I wish they had provided an RS232 chassis connector -- perhaps even as an option -- so I wouldn't have had to order one from another company.

The printed circuit board for the Netronics keyboard is a little flimsy for key pounding. If it is mounted properly it is perfectly OK, but the mounting instructions are included only with the optional case, not with the keyboard itself.

There is no line feed key; Control J yields a line feed. If your computer echoes a line feed when you send it a carrige return, you're okay; otherwise you have either a programming problem or a minor pain in the neck.

No serial video board I have seen -- Netronics included -- has high-resolution graphics like a memory-mapped video board. This could be provided with a RAM character generator, but it really isn't expected at this low price. The Netronics 20 ma current loop is not isolated like the Xitex, and so may not work well with some devices. It does not work well with all the devices I have tried, including KAOS systems.

Granted these deficiencies, why am I glad I bought the Netronics? To summarize:

Quick delivery via an 800 phone number and credit card.

Complete cursor control.

TTY mode, with upper/lower case eaisly available.

The full ASCII character set plus the Greek alphabet, other characters, and some graphic symbols.

Shift lock, control key, and escape key.

A true delete key (Some delete keys only back up the cursor; this one also erases the unwanted character.)

The board works directly with my KIM TTY monitor ROM -- no special software support.

My KIM now has a video terminal which cost less than the KIM. It is a complete, working terminal which will talk not only with KIM but also with time-sharing systems anywhere. I consider it a very efficient and cost-effective means of obtaining ASCII input/output for any of the four KAOS systems.

Given the delay between writing and publication, by the time you read this there may be something better and/or cheaper on the market. These comments should help you to analyze the data sheets and schematics. I can testify that a careful kit-builder, in a few evenings of work, can put together a very attractive and efficient terminal at a very reasonable price. ©

SYMple Clock

A. M. MacKay
Owen Sound, Ontario

There are lots of clock programs around, but this one is a little different, and a lot more useful than most. It is written for a SYM-1 with 4K memory, but will work with a bare board. It is similar in some ways to the one in Issue 1 of Compute II, but doesn't require Basic or a CRT.

While most clock programs using the LED readouts won't do anything else while the clock is running, this one sits at the top of your memory, out of the way of most programs, and quietly ticks away while you and your SYM do other and better things. But when you want the time, either visually or for use by a program, just call it and there it is.

Since a program such as this will likely be stored on tape and loaded when required over the years, no page zero slots are used so you don't have to remember or keep track of them.

The clock runs in 24 hour format. If you prefer 12, change location 0FAB to 12. Then to start the clock, enter the hours, minutes and seconds, with fifteen or twenty seconds lead time, into locations 0FFD, 0FFE and 0FFF respectively. Then enter "GO F3F" and at the exact second hit "CR" and presto! Nothing happens! Ah, but it does. Your clock is running, quietly minding it's own business, eagerly awaiting your summons. Now, to see the time, all you do is hit "SHIFT CALC CR" or "SHIFT 0 CR" or any other "UNRECOGNIZED" command. The time will be displayed for a few seconds, then the readouts will be blanked except for a row of dots. As soon as you see the dots, you can go back to whatever you were doing with your SYM. If you want the time displayed during, and as part of, a program, just use "JSR B9 0F" and there it is.

Most importantly, though, if you want the time for controlling purposes, just call it at 0FFD, 0FFE and 0FFF with your program. It can, at the proper time, sound an alarm, turn off the lights, turn on your lights, and/or whatever makes you happy.

If you don't want clock time, but just the time since your SYM was turned on (actually since the clock started), don't enter anything in 0FFD-F. Just hit "GO F3F CR" and the clock will automatically start at 00 hours 00 minutes 00 seconds.

If your SYM is new, this is a good chance to experiment, changing things to suit your purposes. For example, try moving "DLY" from line 1090 to line 1010 (change "EB" to "D4" at location 0FEA). Your SYM now looks like a cheap digital clock. Now try changing "0A" at location 0FD2 to "1A", then move lines 1070 and 1080 to a new location between lines 1000 and 1010. Do you prefer the display this way?

The theory of operation is similar to that given for my clock article in Compute II no. 1, which required Basic and a CRT. However, the program is somewhat different because the clock in that article kept time in hex, while this one keeps time in decimal. To work in decimal with "SED" you must use ADC or SBC. "INC" just doesn't work.

Your SYM-1 is very powerful by itself, and is the basis for an extremely complex and powerful system. To get the most from it, I urge you to join the SYM-1 Users' Group, P.O. Box 315, Chico, CA 95927. And, of course, subscribe to and keep reading COMPUTE!

```
0005            .OS
0010 ;      ******************************************
0020 ;      ******************************************
0030 ;      ***                                    ***
0040 ;      ***    > > > SYMPLE   CLOCK < < <      ***
0050 ;      ***                                    ***
0060 ;      ***    BY A. M. MACKAY                 ***
0070 ;      ***    CO-ORDINATOR, SURVEY DEPT.      ***
0080 ;      ***    GEORGIAN COLLEGE A.A.T.         ***
0090 ;      ***    1150 EIGHTH STREET EAST         ***
0100 ;      ***    OWEN SOUND, ONTARIO CANADA      ***
0110 ;      ***    N4K 5R4        JUNE 18, 1980    ***
0120 ;      ***                                    ***
0130 ;      ******************************************
0140 ;      ******************************************
0150 ;
0160 ;
0170            .BA $0F3F
0180 ;
0190 ;      * * * DEFINITION OF LABELS * * *
0200 ;
0210 OUTBYT     .DE $82FA
0220 SCAND      .DE $8906
```

```
                   ACCESS      .DE $8B86
                   IFR2        .DE $A405       ;FLAG FOR DISPLAY TIMER
                   DISBUF      .DE $A640
                   URCVEC      .DE $A66D
                   IRQVEC      .DE $A67E
                   CLRINT      .DE $AC04
                   TICH        .DE $AC05
                   TILL        .DE $AC06
                   ACR         .DE $AC0B
                   IFR         .DE $AC0D
                   IER         .DE $AC0E
                   ;
                   ;          * * * INITIATE TIMER * * *
                   ;
0F3F- 20 86 8B     START       JSR ACCESS      ;UNWRITE PROTECT SYS RAM
0F42- A9 71                    LDA #L,CLOCK    :SET IRQ
0F44- 8D 7E A6                 STA IRQVEC      ;  VECTOR
0F47- A9 0F                    LDA #H,CLOCK    :     TO
0F49- 8D 7F A6                 STA IRQVEC+1    :      "CLOCK"
0F4C- A9 B9                    LDA #L,TIME     ;SET UNRECOGNIZED
0F4E- 8D 6D A6                 STA URCVEC      ;  COMMAND VECTOR
0F51- A9 0F                    LDA #H,TIME     ;     TO
0F53- 8D 6E A6                 STA URCVEC+1    :      "TIME"
0F56- A9 C0                    LDA #$C0        ;SET BITS 6 & 7
0F58- 8D 0B AC                 STA ACR         ;  FOR FREE RUNNING MODE
0F5B- 8D 0E AC                 STA IER         ;    AND T1 INTERRUPT ENABLE
0F5E- AD 0D AC                 LDA IFR         ;CLEAR T1 FLAG BIT 6 BUT
0F61- 29 BF                    AND #$BF        ;  DON'T DISTURB OTHER
0F63- 8D 0D AC                 STA IFR         ;    IFR BITS
0F66- A9 50                    LDA #$50        ;SET
0F68- 8D 06 AC                 STA TILL        ;  TIMER
0F6B- A9 C3                    LDA #$C3        ;    FOR 1/20 SEC AND
0F6D- 8D 05 AC                 STA TICH        ;      START TIMER
0F70- 60                       RTS
                   ;
                   ;          * * * INTERRUPT SERVICE ROUTINE * * *
                   ;
0F71- 48           CLOCK       PHA  ;          ;SAVE ACCUMULATOR
0F72- F8                       SED  ;          ;TIME IS IN DECIMAL MODE
0F73- CE FC 0F                 DEC COUNT       ;SEE IF 1 SEC HAS PASSED
0F76- D0 3B                    BNE EXIT        ;IF NO, EXIT
0F78- A9 14                    LDA #20         ;IF YES,
0F7A- 8D FC 0F                 STA COUNT       ;  RESTORE COUNT
0F7D- 18                       CLC  ;          ;    AND
0F7E- A9 01                    LDA #01         ;      ADD 1
0F80- 6D FF 0F                 ADC SECS        ;        TO
0F83- 8D FF 0F                 STA SECS        ;          SECS
0F86- C9 60                    CMP #$60        ;SEE IF 60 SECS HAS PASSED
0F88- D0 29                    BNE EXIT        ;IF NO, EXIT
0F8A- A9 00                    LDA #00         ;IF YES, RESET
0F8C- 8D FF 0F                 STA SECS        ;  SECS TO ZERO
0F8F- 18                       CLC  ;          ;    AND
0F90- A9 01                    LDA #01         ;      ADD
0F92- 6D FE 0F                 ADC MINS        ;        ONE TO
0F95- 8D FE 0F                 STA MINS        ;          MINS
0F98- C9 60                    CMP #$60        ;SEE IF 60 MINS HAS PASSED
0F9A- D0 17                    BNE EXIT        ;IF NO, EXIT
0F9C- A9 00                    LDA #00         ;IF YES, RESET
0F9E- 8D FE 0F                 STA MINS        ;  MINS TO ZERO
0FA1- 18                       CLC  ;          ;    AND
```

```
0FA2- A9 01     0830                 LDA #01      ;     ADD
0FA4- 6D FD 0F  0840                 ADC HOUR     ;       ONE TO
0FA7- 8D FD 0F  0850                 STA HOUR     ;         HOUR
0FAA- C9 24     0860                 CMP #$24     ;SEE IF 24 HOURS HAS PASSED
0FAC- D0 05     0870                 BNE EXIT     ;IF NO, EXIT
0FAE- A9 00     0880                 LDA #00      ;IF YES, RESET
0FB0- 8D FD 0F  0890                 STA HOUR     ;  HOUR TO ZERO
0FB3- AD 04 AC  0900 EXIT            LDA CLRINT   ;ENABLE TIMER INTERRUPT
0FB6- D8        0910                 CLD  ;       ;BACK TO HEX
0FB7- 68        0920                 PLA  ;       ;RESTORE ACCUMULATOR
0FB8- 40        0930                 RTI
                0940 ;
                0950 ;        * * * DISPLAY ROUTINE * * *
                0960 ;
0FB9- 20 86 8B  0970 TIME            JSR ACCESS   ;UNWRITE PROTECT SYS RAM
0FBC- 48        0980                 PHA  ;       ;SAVE ACCUMULATOR
0FBD- 8A        0990                 TXA  ;       ;  AND
0FBE- 48        1000                 PHA  ;       ;    X-REGISTER
0FBF- AD FD 0F  1010                 LDA HOUR     ;PUT
0FC2- 20 FA 82  1020                 JSR OUTBYT
0FC5- AD FE 0F  1030                 LDA MINS     ;  TIME ON
0FC8- 20 FA 82  1040                 JSR OUTBYT
0FCB- AD FF 0F  1050                 LDA SECS
0FCE- 20 FA 82  1060                 JSR OUTBYT   ;    DISPLAY
0FD1- A9 0A     1070                 LDA #$0A     ;SET NUMBER OF
0FD3- 8D FB 0F  1080                 STA CNT1     ;  TIMEOUTS FOR DISPLAY
0FD6- A9 FF     1090 DLY             LDA #$FF     ;SET LENGTH OF
0FD8- 8D 1F A4  1100                 STA $A41F    ;  TIMEOUT
0FDB- 20 06 89  1110 DISPL           JSR SCAND    ;LIGHT LEDS
0FDE- AD FA 0F  1120                 LDA MASK     ;CHECK TIMER
0FE1- 2C 05 A4  1130                 BIT IFR2     ;  IRQ
0FE4- 10 F5     1140                 BPL DISPL    ;IF NO IRQ REPEAT
0FE6- CE FB 0F  1150                 DEC CNT1     ;ELSE START AGAIN
0FE9- 10 EB     1160                 BPL DLY      ;FINISHED?
0FEB- A2 05     1170                 LDX #$05     ;CLEAR
0FED- AD FA 0F  1180 CLR             LDA MASK
0FF0- 9D 40 A6  1190                 STA DISBUF,X :  DISPLAY
0FF3- CA        1200                 DEX
0FF4- 10 F7     1210                 BPL CLR
0FF6- 68        1220                 PLA  ;       ;RESTORE
0FF7- AA        1230                 TAX  ;       ;  X-REGISTER AND
0FF8- 68        1240                 PLA  ;       ;    ACCUMULATOR
0FF9- 60        1250                 RTS
                1260 ;
                1270 ;        * * * STORAGE DEFINITIONS * * *
                1280 ;
0FFA- 80        1290 MASK            .BY %10000000         ;BIT 7 ONLY
0FFB-           1300 CNT1            .DS 1        ;PROVIDE SPACE FOR CNT1
0FFC- 14        1310 COUNT           .BY 20       ;SET COUNT TO 20
0FFD- 00        1320 HOUR            .BY 00       ;START TIME AT 00 HOURS
0FFE- 00        1330 MINS            .BY 00       ;              00 MINUTES
0FFF- 00        1340 SECS            .BY 00       ;              00 SECONDS
                1350                 .EN
LABEL FILE:  [ / = EXTERNAL ]
```

```
/OUTBYT=82FA       /SCAND=8906      /ACCESS=8B86
/IFR2=A405         /DISBUF=A640     /URCVEC=A66D
/IRQVEC=A67E       /CLRINT=AC04     /TICH=AC05
/TILL=AC06         /ACR=AC0B        /IFR=AC0D
/IER=AC0E          START=0F3F       CLOCK=0F71
EXIT=0FB3          TIME=0FB9        DLY=0FD6
DISPL=0FDB         CLR=0FED         MASK=0FFA
CNT1=0FFB          COUNT=0FFC       HOUR=0FFD
MINS=0FFE          SECS=0FFF
//0000,1000,1000
>
```

Expanding KIM-Style 6502 Single Board Computers

Hal Chamberlin

Editor's Note: Hal ended his first installment with this . . .

"The real question at this point then is: How many expansion boards can the unbuffered microprocessor bus drive before becoming overloaded? The 6502 microprocessor is rated to drive slightly more than 1 standard TTL load (equivalent to five low power shottky loads) on its address and data busses while most of the RAM's and ROM's tied to the data bus can drive two standard TTL loads. The 6520, 6522, and 6530 I/O chips have the same drive capability as the microprocessor. Thus in general the answer is at least four boards provided that the expansion boards themselves buffer the bus such that only one low power shottky load (.36MA in the zero state) is presented to the bus by the board. Many boards on the market and particularly those designed for an unbuffered bus do this. Actually, any well designed board would be expected to buffer the bus in order to provide clean signals for the remainder of the board logic. The reason that only four boards can be driven instead of five is that some of the address lines are loaded by a low power Shottky decoder IC on the computer board itself.

Part 2 of 3 The Great Experiment

Of course loading the microprocessor with a full five loads puts the system right at the limit of rated drive current. One of the problems with testing digital circuitry is that there is no obvious indication of marginal operation that may later develop into a full fledged failure as components age. In order to determine the actual drive limit, the author took a fully stuffed AIM-65 (4K on-board RAM, assembler ROM and BASIC ROM's) and started adding Micro Technology K-1016 16K memory boards, the idea being to add boards until failure due to bus overload occurred. These boards use low power Shottky buffers onboard so each one would be expected to add a .36MA load to the bus.

Since the AIM's 40K of free addresses would only accomodate two of these boards, the most significant address bit was cut away from the bus at each socket position and instead connected to parallel output bits on the AIM's application connector. The boards were then jumpered to respond to addresses between 2000 and 5FFFF (hex). By programming only one output bit to be low at a time, a rudimentary bank switching setup was implemented. When the system was reset, all output bits automatically go high thus disabling all of the boards and preventing interference with the AIM monitor (since A15 was ignored, an enabled board would also respond to A000-DFFFF). A proper bank switch setup would have required a two-input OR gate (negative AND) to be tied to each of the A15 pins. In any case, it was adequate to run a memory test program.

The first trial was to install 4 of the 16K boards which worked fine as expected. Next, another card file was placed below the first and jumper wires added between the two motherboards. This gave a total of 9 bus slots which were filled with 16K memory boards. Again the memory test program (which wrote all 144K of memory with random data before reading any of it back) indicated no problem and the AIM monitor and BASIC continued to work flawlessly. A check with an oscilloscope revealed minimal signal degradation.

Finally, a third card file was added and bus jumpers installed to give a total of 14 slots. Three additional 16K memory boards were scrounged (I had no idea that more than 9 or 10 boards could be driven) to give a total of 192K of RAM. Again there were no obvious problems and the bus was being loaded to three times rated capacity! Figure 3 shows what the stack of card files looked like which is obviously impractical unless one cuts a hole in the tabletop to let the two extra card files hang below (I simply sat on a drafting stool to use the system). The rear view in figure 4 shows the interconnected motherboards and individual Board Enables from the application connector. Note the gridwork of copper braid between motherboards which makes the groundplane essentially continuous between the motherboards.

Photographs of the address and data bus signals were taken while running the memory test program and are shown in figure 5. About the only visible loading effect on the address bus is a long tail on the zero-to-one transition during phase 1 of the clock. The data bus appears to be even cleaner with just a shade over 100NS required for the data to stabilize after the leading edge of phase 2. The microprocessor was driving the data bus for the data bus for this photo (scope synced to read/write line on the bus). The zero logic levels, which one would think show the effect of gross overloading most, were still in the 0.3 volt range although the one levels were down to only 3 volts from a normal lightly loaded value of nearly 4 volts. Note the almost complete absence of noise. These "overloaded" signals actually look far better than most S-100 bus signals!

While these results are encouraging and certainly show that a four board load does not bring a system to the brink of failure, it does not mean that loading rules can be disregarded altogether. Some AIM's, as well as SYM's and KIM's, can be expected to have a weak component on-board that may not be able to drive a 12 board load adequately for reliable operation. Thus the "official" recommendation is to stick with the spec book and limit unbuffered systems to four boards. However, individual hobbyists should be able to go one or two boards over the limit with little probability of problems. Actually, addressing limitations are more likely to limit system size than bus drive capability with today's dense boards.

FIG.3.FRONT VIEW OF 192K RAM TEST SYSTEM

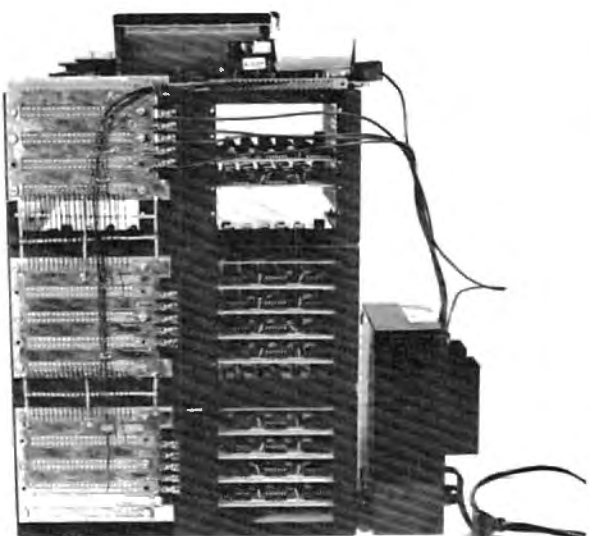
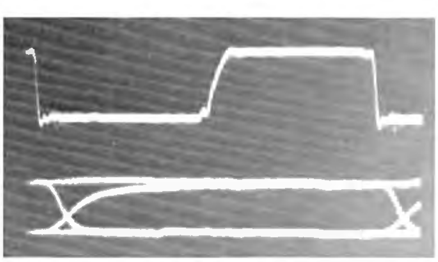


FIG. 4. REAR VIEW OF 192K TEST SYSTEM SHOWING MOTHERBOARDS WIRED TOGETHER



A. ADDRESS BUS

B. DATA BUS

FIG. 5. BUS SIGNAL WAVEFORMS IN 192K TEST SYSTEM. TOP WAVEFORM IN EACH PHOTO IS PHASE 2 CLOCK.

©

Load And Save KIM BASIC Programs On Your SYM

George Wells
LaVerne, CA

The SYM and KIM microcomputers are close cousins. Their hardware and tape interface compatibilities are well known. Not so well known is the fact that the BASIC interpreters on the two systems use the same tokens for their reserved keywords which makes transfer of BASIC programs between the SYM and KIM almost trivial.

Mike Hanna, a friend of mine who has had a KIM with BASIC for much longer than I have had my SYM with BASIC, has offered to share his library of BASIC programs with me. We had considered implementing a telephone/modem interface to accomplish this transfer but after comparing the disassembly listings of the two interpreters we decided a tape transfer would be easier. The scheme we finally settled on allows the SYM to create and read tapes in the original low speed KIM format since the SYM does not support any of the faster versions. Going from the SYM to the KIM is particularly simple; going the other way requires a short BASIC program (see listing).

SYM To KIM Transfer

STEP 1: Load the BASIC program to be transferred into the SYM.

STEP 2: Exit BASIC and return to the Monitor (by way of reset, for example).

STEP 3: Determine the end of the BASIC program by examining the two-byte pointer stored at $7D/$7E by entering .V 7D-7E. The SYM will respond with:

007D uv wx,yz

where wxuv is the end of the program (qrst) plus one. The monitor will calculate qrst for you if you can't do it in your head by entering .C wxuv-1.

STEP 4: Save the program on tape in KIM format by entering:

.S1 1,201-qrst

where qrst is the value from STEP 3.

STEP 5: Load the program into KIM BASIC in the normal manner.

KIM To SYM Transfer

In order to load KIM formatted BASIC programs into your SYM you will need to have a copy of the KIM BASIC PROGRAM LOADER listed with this article. Save this program on tape (in high speed format, of course) so that you will have it whenever you need it. NOTE: This program will not work with Monitor Version 1.0 which has an error in the KIM Load routine.

```
LIST: REM    KIM BASIC PROGRAM LOADER

 100 A=USR(&"8B86",0)
 110 A=42572: POKE A,1: POKE A+1,2: POKE A+2,255
 120 FOR I=0 TO 29
 130 POKE 300+I, PEEK(35960+I)
 140 NEXT I
 150 POKE 330,96
 160 PRINT "AFTER 'LOADED' MESSAGE, ENTER:"
 170 PRINT "POKE 125,PEEK(254): POKE 126,PEEK(255): CLEAR"
 180 PRINT USR(300,&"C6C5",&"8CAC",0)
OK
```

STEP 1: On the KIM, save the program to be transferred in the normal manner; but make sure it is saved at the original tape low speed.

STEP 2: Initialize BASIC on your SYM and LOAD and RUN the KIM BASIC PROGRAM LOADER.

STEP 3: Play the tape with the KIM program in your recorder. If you have implemented a second cassette control for your read-only recorder you will have to over-ride it since this program will only activate the original cassette control.

STEP 4: After the LOADED message, enter the command printed by the program and then SAVE a copy of the KIM program in high speed format. In case you get a BAD LOAD message, start over again at STEP 2.

SYM/KIM BASIC Incompatibilities

The obvious hardware related incompatibilities due to different address availability in the two systems require careful use of the PEEK, POKE and USR commands. Of course, different terminals may also have special requirements for cursor controls or graphics capabilities. Not so obvious are the following additional potential problem areas.

GO: SYM treats GO as a reserved word so don't enter GOTO as two words. Also make sure that GO does not appear in any variable names such as DRAGON.

GET: SYM does not implement this function but it does reserve the same token as KIM. (See MICRO 24:15 if you want to implement GET on your SYM.)

USR: The multiple parameter versions of USR will not work on the KIM. The single parameter version will require a different set of POKE commands prior to the USR but otherwise it works the same in both systems.

& "ABCD": KIM does not support hexidecimal notation.

Theory Of Operation

The key to the success of this scheme is in the fact that the Microsoft BASIC interpreters automatically recalculate the internal line pointers whenever a BASIC program is loaded. Theoretically, this means that a program that was originally saved at one address could be loaded into a different address if the tape load routine is given the ID value of $FF and the address where the program is to be loaded. The problem is that in some versions of KIM BASIC the ID value of $FF is used to save programs which means that since the IDs match when loading, the programs get loaded into their original address instead of the new specified address. There are two ways to fix this problem. First, if you have access to the KIM you can change location $2744 from $FF to $01 before the program is saved. This is part of the sequence LDA *$FF, STA $17F9.

The second method is the one the KIM BASIC PROGRAM LOADER uses which will work with any ID. It requires making a copy of the beginning of the SYM Monitor Load routine on page one of the SYM up to the point where the ID test is made. The FOR/NEXT loop in the LOADER program copies the code between address $8C78 and $8C95 and then an RTS instruction is attached to the end ($60 = decimal 96).

The jump to continue into the Monitor Load routine is performed by an interesting technique which Hans W. Gschwind of West Germany wrote about in SYM-PHYSICS 4-20. It involves using the multi-parameter version of the USR function by pushing two return addresses on the stack so that when the first subroutine finishes it returns to the address equal to the third parameter of the USR command plus one which is the continuation point in the Monitor Load routine. The next RTS instruction encountered returns to the address of the second parameter plus one which is the normal return point for BASIC high-speed tape loads.

With this background in mind it is possible to understand the following line by line explanation of the KIM BASIC PROGRAM LOADER.

LINE 100: Calls the Monitor ACCESS routine to allow passing of tape parameters to System Ram.

LINE 110: Passes tape start address of $201 and ID of $FF to tape parameters.

LINES 120 to 140: Copies first part of Monitor Tape Load routine to page one.

LINE 150: Ends page one copy with an RTS.

LINES 160 and 170: Prints message to be entered after a good load. The command must be entered manually since the KIM BASIC program will overwrite the LOADER program.

LINE 180: Jumps to address 300 (first parameter) with Y index register equal to zero (fourth parameter) indicating KIM tape format. The RTS at address 330 jumps to address $8CAD (third parameter plus one). The RTS at the end of the Monitor Tape Load routine jumps to address $C6C6 (second parameter plus one) in the BASIC interpreter which modifies the line pointers to fit the new location in the SYM.

Conclusion

Hopefully this scheme can be used to advantage by anyone having access to both a SYM and a KIM. If you find that it just doesn't work for you, try a different tape recorder. Mike and I spent many frustrating days trying to get the SYM to KIM transfer to work and it wasn't until I used a different recorder with my SYM before we finally did have success! Now we are able to transfer our BASIC programs with ease. ©

Advertisers' Index

SEND FOR OUR FREE CATALOG

COMPUMART LETS YOU PUT APPLE ON TRIAL

WITH OUR EXCLUSIVE 10-DAY FREE RETURN ON ALL THESE GREAT APPLE COMPUTERS AND ACCESSORIES.

CompuMart carries complete lines of microcomputers from Apple to Zenith, as well as thousands of microcomputer peripherals and accessories. Write for our **FREE 36 page catalog**

Apple Computer
We carry the most complete inventory of Apple computers, peripherals, and software. CALL!

Our **Christmas Apple Special**
over $250 on our most popular Apple System. System includes a 48K Apple II, Apple Disk & Controller, and a Sup R Mod RF Modulator.
List: **$2,020**
Compumart Sale Price: **$1,769**

New from Apple for the Apple II
DOS 3.3 Convert disks to 16 sector format for 23% more storage and faster access **$60**

Apple Plot. The perfect graphic complement for Visicalc **$70**
Dow Jones News & Quotes **$95**
Adventure (Uses 48K) **$35**
DOS Tool Kit **$75**
Apple Fortran **$200**
Silentype Printer w/Xface **$595**
Visicalc **$149**
Tax Planner **$120**
From Symtech & Info Unlimited
Super Sound Generator (mono) **$159**
(stereo) **$259**
Light Pen **$249**
X-10 Controller (plugs into paddle port) **$49**
Apple Sync Controller **$49**
From Personal Software
Visicalc **$149**
Desk top plan **$99**

New from Videx! — Video Term
80 Col. x 24 line
7 x 9 matrix, plug in compatible board for the Apple II. Price **$325**
without graphics EPROM.
With graphics EPROM ..... **$350.**

New from MUSE
The Voice **$39.95**
Super Text **$99.00**
Address Book **$49.95**

Mountain Hardware — Expansion accessories for your Apple
Introl/X-10 System **$289**
Super Talker **$299**
The Music System **$545**
ROM plus board w/keyboard filter **$199**
Clock Calendar **$280**
16 Channel A to D Converter **$350**
Apple Expansion Chassis **$650**
ROM Writer **$175**

Miscellaneous Apple II Accessories
Easy Writer (80 col. need Videx) **$249**
Easy Mover **$49**
Easy Mailer **$69**
Dysan Diskettes ..... **ea. $5**
S.S.M. Serial & Parallel Apple Interface **$225**
ABT's Numeric Key Pad **$110**

270 THIRD ST. DEPT 307 **COMPUMART** CAMBRIDGE, MA 02142
TOLL FREE 1-800-343-5504 **IN MASS 617-491-2700**

apple computer
Authorized Dealer

The Great American Solution Machine.

Meet Commodore. The business computer that's providing solutions for more than 100,000 people all over the world. Built by one of the pioneers in office machines. With a reputation for quality that can only come from vertical integration of the total manufacturing process. Commodore builds, not assembles.

Compare Commodore's word and data processing capabilities with computers costing twice or even three times as much. You'll see why so many small businesses are turning to Commodore for solutions to problems as varied as these:

☐ A car leasing company's customers were terminating too early for account profitability. Solution: A 16K Commodore. It analyzes cash flow on over 1200 accounts, identifies those for early penalties, and even writes up lease contracts. Commodore paid for itself within weeks.

☐ A law firm needed a high quality, easy-to-use, affordable word processing system. Solution: Commodore plus its WordPro software package. At a $6,000 savings.

☐ A gasoline retailer needed to inventory, order and set prices; determine Federal and state income taxes; and comply with Federal pricing and allocation regulations. All done daily, weekly, monthly and yearly. Solution: Commodore. It keeps his business on track—and Uncle Sam off his back.

☐ A paint and wallpaper store had to inventory over 600 expensive wallpaper lines for profitability, monitor distributor sales, set and track salesmen's goals, and help the customer select the right size, pattern and quantity. Solution: Two 32K Commodore computers, floppy disk and printer. Commodore does it all—and accounting, too.

In applications like these, and many more, Commodore solves the problems that stand in the way of increased profitability. Commodore can provide the solution in your Great American business, too. Find out more by calling or writing any of Commodore's District Sales Offices.

COSTA MESA, CA 2955 N. Airway Avenue 92626. (714) 979-6307.
SANTA CLARA, CA 3330 Scott Boulevard 95051. (408) 727-1130.
DECATUR, GA 5360 Snapfinger Woods Drive 30035. (404) 987-3311.
BENSENVILLE, IL 790 Maple Lane 60106. (312) 595-5990.
NORRISTOWN, PA 950 Rittenhouse Road 19401. (215) 666-7950.
DALLAS, TX 4350 Beltwood Parkway South 75234. (214) 387-0006.

Commodore Business Machines, Inc., Computer Sales Division, Valley Forge Corporate Center. 950 Rittenhouse Road, Norristown, PA 19401.

commodore

COMMODORE