NEW

ATARI®
SOFTWARE
By PMS

MULTI-PURPOSE BUSINESS PACKAGES FOR ATARI® 800

- **General Ledger**
  Receipts, Disbursements, Balance Sheet, Income Statements, General Journal, and more for small to medium businesses
- **Cashier - Inventory Control**
  Point-of-sale retail cashier with inventory control in one powerful package — or use Inventory Control alone!
- **Accounts Payable and Receivable**
  Subsidiary journals to integrate with General Ledger

PMS can help **design a complete business system **specifically for you** - call for details!*

DISTRIBUTED BY
IMAGE WORKS
SOFTWARE

MC - VISA Accepted
$2.00 shipping & handling
add $1.00 for COD

3124 South Parker Road
Suite C-222
Aurora, CO 80044

(303) 690-6372
Orderline 9-7 MST
Free catalog on request

® Atari is the Trademark of Atari, Inc.
Dealer inquiries invited
Colorado residents add appropriate tax

OCTOBER SPECIALS
good thru 11/15/81

25% OFF ANY **CRYSTALWARE** (Atari 800 / 48k disk) in stock

Fantasyland $45.00 • WWIII $22.00 • Mars $30.00 • call for others

ALSO INTEC & AXLON MEMORY BOARDS!
32k INTEC $145.00 • 48k INTEC $245.00 • 128k AXLON $CALL

ATARI®

Software for Personal Computers

A collection of 10 challenging programs created to provide a unique entertainment value — and 2 personal/business programs with broad functional value. In disk and/or cassette as indicated.

GAMES	• HELICOPTER BATTLE	Req. — 16K RAM/Cassette	$ 9.95
		— 16K RAM/Disk	14.95
	• HORSE RACING	Req. — 16K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• KENO	Req. — 8K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• LIGHTNING BOLTS and REACTION	Req. — 16K RAM/Cassette	9.95
		— 24K RAM/Disk	14.95
	• THE MAD MARBLE	Req. — 8K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• MUSIGAME (2 Games)	Req. — 16K RAM/Cassette	9.95
		— 24K RAM/Disk	14.95
	• SUPERMASTER	Req. — 8K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• TAG	Req. — 16K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• TRACTOR BEAM	Req. — 8K RAM/Cassette	9.95
		— 16K RAM/Disk	14.95
	• WAR AT SEA	Req. — 16K RAM/Cassette	14.95
		— 24K RAM/Disk	19.95
BUSINESS	• CCA Data Management System	Req. — 48K RAM/Disk	99.95
	• LETTER WRITER	Req. — 24K RAM/Disk	19.95

Mastercard & VISA Accepted

DIVISION OF CUSTOM ELECTRONICS, INC.
SOFTWARE

238 Exchange St., Chicopee, Massachusetts 01013
(413) 592-4761

• Dealer And Distributor Inquiries Invited
• Closed Mondays — Open Daily 'Til 5:30 — Fridays 'Til 8

SOFTWARE for the ATARI 800* and ATARI 400* from QUALITY SOFTWARE

STARBASE HYPERION™
By Don Ursem

Become absorbed in this intriguing, original space simulation of war in the far future. Use strategy to defend a front line Star Fortress against invasion forces of an alien empire. You create, deploy, and command a fleet of various classes of space ships, while managing limited resources including power generators, shields and probes. Real time responses are sometimes required to take advantage of special tactical opportunities. Use of color, sound, and special graphics add to the enjoyment of this program. At least 24K of RAM is required.

On Cassette — $19.95 **On Diskette — $22.95**

NAME THAT SONG
By Jerry White

Here is great entertainment for everyone! Two players listen while the Atari starts playing a tune. As soon as a player thinks he knows the name of the song, he presses his assigned key or joystick button. There are two ways to play. The first way requires you to type in the name of the song. Optionally, you can play multiple choice, where the computer asks you to select the title from four possibilities. The standard version requires 24K of RAM (32K on diskette) and has over 150 songs on it. You also get a 16K version that has more than 85 songs. The instructions explain how you can add songs to the program, if you wish. Written in BASIC.

On Cassette — $14.95 **On Diskette — $17.95**

QS FORTH
By James Albanese

Want to go beyond BASIC? The remarkably efficient FORTH programming language may be just for you. We have taken the popular fig-FORTH model from the FORTH Interest Group and expanded it for use with the Atari Personal Computer. Best of all we have written substantial documentation, packaged in a three ring binder, that includes a tutorial introduction to FORTH and numerous examples. QS FORTH is a disk based system that requires at least 24K of RAM and at least one disk drive. Five modules that may be loaded separately from disk are the fig-FORTH kernel, extensions to standard fig-FORTH, an on-screen editor, an I/O module that accesses Atari's operating system, and a FORTH assembler.

Diskette and Manual — $79.95 **Manual Only — $39.95**

FOR OUR COMPLETE LINE OF ATARI SOFTWARE
PLEASE WRITE FOR OUR CATALOG

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

ASK FOR QUALITY SOFTWARE products at your favorite computer store. If necessary you may order directly from us. MasterCard and Visa cardholders may place orders by calling us at (213) 344-6599. Or mail your check or bankcard number to the address above. California residents add 6% sales tax. *Shipping Charges:* Within North America orders must include $1.50 for 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.

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

©

Atari 400/800 Variable Name Utility

Arthur McGraw
Whitehall, OH

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

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

User Instructions

1. Clean out the computer's memory using the "NEW" command.

2. Type in the variable name table dump utility using line numbers that you do not normally use in writing BASIC programs. Modify the GOTO statements in lines 6 and 7 if you do not use my numbering scheme.

3. Save the utility on tape or disk using the "LIST" command.

4. Read in the desired BASIC program.

5. Read in the variable name table dump utility using the "ENTER" command. This will overlay your program with the utility program.

6. Start the utility by entering "GOTO x" where "x" is the first line number of the utility.

7. Observe the variable name table as it prints on the screen. The last character of each variable name will be printed in inverse video.

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

Address List

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

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

Notes

1. Normally, the variable name table contains names no longer used by the program. Because these variable names occupy memory area, it would be desirable to remove them. Unused variable names can be removed from the variable name table by saving the program with the "LIST" command, clearing memory using the "NEW" command, and then reloading the program using the "ENTER" command.

2. If you do not have room to add this utility, you can delete as many lines of your program as needed by typing in their associated line numbers followed by RETURN. Deletion of program statements in this manner will have no effect on the variable name table.

©

Insight: Atari

Bill Wilkinson
Optimized Systems Software
Cupertino, CA

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

Nonexistent Subroutines ... Or Having Fun While GOSUBing Nowhere

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

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

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

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

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

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

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

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

The techniques discussed in this and prior articles have actually been used to write a "PICO-ADVENTURE." The most amazing aspect of the program is the speed with which it responds: it seems as fast or faster than even machine language adventures. Try it. Let us know about your efforts.

Drawing Tablet

VersaWriter operates on a simple principle, but produces graphics which match or exceed those of other digitizers. Its rugged yet precision construction makes it easy to use and trouble free. Operation is mastered in minutes. It plugs directly into your ATARI personal computer.

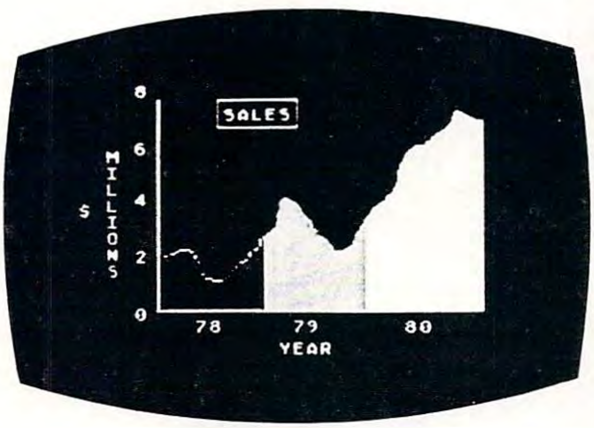
VersaWriter

FOR ATARI PERSONAL COMPUTERS

Suggested Price $299.00

Graphics Software

Easily the most capable, complete, and usable graphics software for ATARI personal computers available. Designed for hobbyists, but engineers, artists, doctors, and educators are also finding exciting ways to expand their computer's value with VersaWriter.



UNIQUE OFFER

Send us YOUR disk and $1. We will promptly return the disk with a slide package of 10 color pictures drawn with VersaWriter.

☐ Enclosed is $1 and my disk. Send me the slide package.

☐ ATARI ☐ APPLE II

☐ Send more information including VersaWriter dealers in my area.

DEALER INQUIRIES INVITED.

NAME

ADDRESS

CITY STATE ZIP

Versa Computing, Inc. • 887 Conestoga Circle • Newbury Park, CA 91320 • (805) 498-1956

Unreadable Programs

There follow two program lines that can be added to any existing Atari BASIC program and which, when executed, will make a program virtually unLISTable. The first line simply changes the names of all your variables to RETURN characters, and can be used with or without the second line. The second line actually produces a BASIC SAVEd program that can only be RUN — not LISTed, LOADed, etc.

Atari BASIC version:

```
32766 FOR I=PEEK(130)+256*PEEK(131) TO
      PEEK(132)+256*PEEK(133) : POKE I,155 :
      NEXT I
32767 POKE PEEK(138)+256*PEEK(139)+2,0 : SAVE
      "<filename>" : NEW
```

BASIC A+ version:

```
32766 for i=dpeek(130) to dpeek(132) : poke i,155 :
      next i
32767 poke dpeek(138)+2,0 : save "<filename>"
      : new
```

To use these gems, simply enter them and then type GOTO 32766. The line numbers are not important, but the second line must be the last line of the program. To use the resulting program, simply type RUN "‹filename›". The program should *not* end with STOP or END; instead, it should exit via NEW (yes, "NEW" can be used from within a program). The ‹filename› may be "C:", but CLOAD will not work (you must use RUN"C:").

VARIABLE, VARIBLE, VARABLE

Perhaps one of the more common mistakes when using the long variable names allowed by Atari BASIC is to make a typo when entering the name (I tend to leave off the plural, "s"). How to know you have committed this sin? Try the following program segment:

Using Atari BASIC:

```
32700 I=0 : FOR J=PEEK(130)+256*PEEK(131) TO
      PEEK(132)+256*PEEK(133)-1
32710 IF PEEK(J) < 128 THEN PRINT CHR$(PEEK
      (J)) ; : GOTO 32730
32720 PRINT CHR$(PEEK(J)-128) : I=I+1
32730 NEXT J : PRINT : PRINT I ; " VARIABLES
      IN USE" : STOP
```

Using BASIC A+:

```
32700 i=0 : for J=dpeek(130) to dpeek(132 + 1 : print
      chr$(peek(j) & 127) ;
32710 if peek(j) > 128 : print : i=i+1 : endif : next j
32720 print : print i ; " variables in use" : stop
```

It would be advisable to LIST this program segment to DISK or CASSETTE and then ENTER it into any program that needs it. To use, simply type GOTO 32700 ; all your variables will be listed and a count displayed. Obviously, this output could be sent to the printer by first OPEN #7,8,0, "P:" and then replacing all the PRINTs with PRINT #7. If you do this, it is advisable to CLOSE #7 before the STOP.

```
1000 REM ***** SET UP ********
1010 DIM VERB$(4),NOUN$(4),TEST$(4)
1020 VERBDATA=9000:NOUNDATA=9100
1030 LOOP=2000
2000 REM ***** MAIN LOOP ******
2010 PRINT "GIVE ME A VERB ";:INPUT VERB$
2020 RESTORE VERBDATA:VERB=0
2030 FOR CNT=1 TO 3:REM CHANGE TO MATCH DATA
2040 READ TEST$
2050 IF TEST$=VERB$ THEN VERB=CNT:CNT=99
2060 NEXT CNT
2070 IF  NOT VERB THEN PRINT "INVALID VERB":
     GOTO LOOP
2100 REM VERB DONE, DO NOUN
2110 PRINT "GIVE ME A NOUN ";:INPUT NOUN$
2120 RESTORE NOUNDATA:NOUN=0
2130 FOR CNT=1 TO 3:REM CHANGE TO MATCH DATA
2140 READ TEST$
2150 IF TEST$=NOUN$ THEN NOUN=CNT:CNT=99
2160 NEXT CNT
2170 IF  NOT NOUN THEN PRINT "INVALID NOUN":
     GOTO LOOP
2200 REM ***** THE TRICKY STUFF *****
2210 TRAP 2250
2220 GOSUB 10000+VERB*1000+NOUN*10
2230 GOTO LOOP
2240 REM WE GET TO 2250 ONLY ON TRAP
2250 GOSUB 10000+VERB*1000
2260 GOTO LOOP
9000 REM A LIST OF ALL VERBS
9010 DATA LOOK,KISS,DROP
9100 REM A LIST OF ALL NOUNS
9110 DATA ROOM,BEAR,BABY
10000 REM ***************************
10010 REM *    THE VERB-NOUN ACTION *
10020 REM ***************************
11000 REM >>> LOOK <<<
11001 PRINT "I SEE NOTHING SPECIAL":RETURN
11010 PRINT "I SEE A WINDOW AND A DOOR":RETURN
12000 REM >>> KISS <<<
12001 PRINT "THAT'S SILLY...BUT SMACK":RETURN
12020 PRINT "BEAR BITES OFF YOUR LIPS":RETURN
12030 PRINT "YOU HAVE JUST BEEN ELECTED
      MAYOR":RETURN
13000 REM >>> DROP <<<
13001 PRINT "HOW? I COULDN'T HAVE LIFTED
      THAT.":RETURN
13030 PRINT "IT'S A BOUNCING BABY BOY!!"
      :RETURN
```

©

Overview:

"Letter Perfect" Wordprocessing On The Atari

This is the first **COMPUTE!** *Overview. We feel that our readers deserve the most objective, comprehensive analysis of new products possible. We have established review panels, each consisting of three reviewers whose background qualifies them to analyze new items of software or hardware.*

To better prepare you for your buying decisions, and to present fair reviews, we collect the three independent opinions of our panelists into one Overview. Lawyers will test legal packages in the environment of a legal office, doctors will test medical systems, and this issue's Overview panel is made up of professional writers. We believe that you will find **COMPUTE!** *Overviews complete, informative, and balanced. — KEM*

The "LETTER PERFECT" program by LJK Enterprises, Inc. is one of the first word processor programs for Atari. It requires 24K of memory and one disk drive and costs $149.

The program is very well documented, and easily followed, with 85 pages of instruction and indexing. This material is supplied in a handsome binder.

As the manual states, the control functions are relatively simple to remember (Control A for "go to beginning of a line," for example, or Control Z for "go to the end of a line," and control T for "go to the top of the page"). This doesn't mean you can remember all 42 functions after one sitting, of course, but the letters are not just utilized arbitrarily and it doesn't take long to catch on. (See Table 1).

In its present format, Letter Perfect is configured for the Atari 825 Printer and the Epson MX 80.

The Epson will not permit underlining, subscripts or superscripts, but the program does permit use of standard, condensed, and enhanced fonts. On the Atari 825, you may use standard, boldface or proportional fonts, although you will not get right hand justification when you use the proportional font.

Printer Considerations

It seems clear that anyone considering the purchase of a printer would do well to check on just what Word Processing systems are currently available that support the various printers, and what they do. In that context, Ken Leonhardi of LJK Enterprises, who puts out Letter Perfect, says that improvements are planned that include the capability of using Letter Perfect with just about any printer on the market.

It is very flexible in the format of the print and allows the user to set the format several times within the text. This could be used to single space for quotations in a double spaced letter or to indent a list of items within a manuscript.

Other printout features include auto page numbering, header and footer capability, auto line centering and all the standard features such as single/double line spacing and left margin adjust.

An Unusual Disk Capability

Letter Perfect also has its own disk operating system, which for some uses could be a disadvantage. Leonhardi says, however, a utility program will be released that will enable the program to be used with the Atari DOS.

Future disk versions of Letter Perfect will be available at a small additional cost for those who buy the current version.

One small warning: there are some early Atari disk drives on which Letter Perfect won't operate. This is apparently true for only a handful of the first Atari 810s that were released, but, if you've had one for some time, there could be a problem in running this program.

There are good and bad points to the unusual requirement that disks be specially formatted for Letter Perfect text storage. The manual describes "packing" — removing unnecessary spaces (such as the spaces between the period which ends one paragraph and the start of the next paragraph). This permits more text on a given disk. WordPro, for the Commodore machines, for example, saves *every* space to the disk. On the other hand, few of us are novelists, nor would we care very much about saving great amounts of text to individual disks. If you were writing a book, you might find that such packing saved a little time, but normal word processing uses do not benefit much from any special disk storage efficiency. The manual also suggests that such storage prevents problems should Atari change its DOS in the future. The special Letter Perfect formatting takes about 1 minute per disk. However, you usually need to format only rarely in word processing application.

Text Entry

First, some mechanics.

When you first load the program you are given a menu of commands, such as Editor, Load, Save, Printer, Delete, Lock, Unlock, etc. You go to the Editor command to begin writing, and can return to the menu by hitting the escape key.

All of the Letter Perfect processing functions are done by using the Atari control key. This works out quite well, since most Atari users already are practiced at using the up, down, right and left

arrows and the insert and delete functions, and they operate in Letter Perfect just as you would normally use them to write a program on your Atari. There are some things to watch out for, however.

One frequently uses the tab key to move to the middle of a line. If you use the tab key in Letter Perfect on a line that has text on it, the tab will erase as it goes along. This was seen as a serious weakness by two panel members.

There are 42 control key functions and they are listed on a handy card that Letter Perfect provides that fits neatly on top of your Atari. It also lists 13 commands for changing the default format (margin settings and the like) and the commands for subscripts and superscripts.

The functions include scrolling, saving paragraphs in memory, and inserting them afterward, replacing misspelled words automatically, tab setting, etc.

The program does not permit you to scroll upward. If you want to back up and read two paragraphs earlier in your text — you must return to the top of the text. Another time consuming feature is the fact that whenever you go from a main menu function (such as disk access) back to the text mode for writing, you must wait for the entire text to scroll down the screen in front of you. Such features quickly become tiresome when you are forced to wait for them.

The Atari screen permits only ½ page (a standard, typewritten 8½ x 11 sheet) to appear at one time. Though the panel felt that this was not fatal, it was seen as a constraint for many wordprocessing applications. In general, the more text available to you at a time, the easier it is to edit and review.

One of the primary differences between using a typewriter versus a wordprocessor is that, on the latter, you do not hit a carriage return until you reach the end of a *paragraph*. The computer will later break your text into individual lines when printing it on paper. Typing becomes more convenient and faster.

Related to this is a "parsing" feature whereby a word will not break at the rightmost side of the screen. If you type through the right side (your word flows onto the next line), Letter Perfect jumps the entire word down to the start of the next line. Some word processors parse, some do not. It is sometimes thought that parsing makes it easier to read a video text since no words are ever broken in half. The majority of our panelists felt, however, that the jumping action during typing is more distracting than the minor inconvenience created by randomly hyphenated words on screen. What is more, parsing wastes screen space.

After, or during, text entry a process called editing is used to correct errors, change wording, move paragraphs, etc. Editing using "LETTER PERFECT" is a simple process identical to editing BASIC programs. The cursor is moved using both the control key and a cursor direction key. Then characters are inserted/deleted using the control and insert/delete keys.

More sophisticated editing capabilities are also provided:

Search — Automatically searches for a specific text string. Allows easy location of certain area of text.

Search/Replace — Automatically searches for a specific string and replaces it with another string. Very useful for changing misspelled names, etc.

Scroll 1 Page — Automatically displays the next complete page of text. This is more convenient than line scrolling which is also available.

Go To End/Beginning Of Line — Moves the cursor to the end/beginning of the current line. This saves time in moving the cursor.

Also provided is the ability to edit blocks (paragraphs or as much as two pages) of information. This allows moving paragraphs within the text without retyping. This capability is useful.

The General Overview

•**Panelist #1:** "This is my first use of a word processor as a writer. What do I think of it? On that score, the only thing I can say is sensational.

After this first effort, I am fully prepared to kiss those old manual Underwoods and Royals goodbye.

But, remember, I am doing this review from the standpoint of a professional writer, not a businessman or an engineer. For the kind of final polishing I want to do, the editing functions of this program are everything I could ask for. Formatting is not particularly important to me, since most of the writing I do consists of fairly simple blocks of text.

For writing articles, papers, simple business letters, and the usual run of home uses, I find the Letter Perfect program to be quite adequate. It also happens to be fun to use, something which is a lot more important than it sounds.

Like many people who get paid for writing, I find one of the hardest things in the world to do, and one of the easiest things to avoid, is just plain sitting down and writing. Since I've had this program in my home, I've discovered that whenever a new idea comes to me, I can't wait to get at it."

•**Panelist #2:** "I feel that Letter Perfect (with exceptions such as parsing, no upscrolling, etc.) makes extensive and positive use of the computer as a writing tool. I would wonder, though, if the Atari, with its 1/2 printer-page screen limitation, would be the best computer to use for lengthy writing tasks. However for shorter, more common, word processing (essays, letters, mailings and the like), Letter Perfect on the Atari does an admirable job."

•Panelist #3: "Space does not permit a discussion of all the features of "LETTER PERFECT" or a complete description of the keystrokes necessary to accomplish the functions. However, let me say that "LETTER PERFECT" is a very powerful word processor which has been written with ease of operation in mind and should be considered by Atari owners with word processing applications."

Letter Perfect
LJK Enterprises, Inc.,
P.O. Box 10827
St. Louis, MO 63129
$149.00

An Overview of Textwizard is currently being prepared. Watch for it in an upcoming issue of **COMPUTE!**.

Letter Perfect Atari Functions

KEY	FUNCTION
CTRL A	GO TO BEGINNING OF LINE
CTRL B	BOLDFACE TOGGLE
CTRL C	CENTER NEXT LINE
CTRL D	DELIMITING CHARACTER
CTRL E	(END) GO TO END OF TEXT
CTRL F	FORMAT LINE
CTRL G	FOOTER
CTRL H	HEADER
CTRL I	IMPROVE TEXT
CTRL J	(JOIN) ADD TO BUFFER
CTRL K	(KILL) DELETE BUFFER
CTRL L	(LIFT) INSERT FROM BUFFER
CTRL M	MOVE TO BUFFER
CTRL N	(NEXT) DELETE NEXT BLOCK
CTRL O	(ON, ON, ON) CONTINUOUS SCROLL
CTRL P	FORCED END OF PAGE
CTRL Q	SCROLL ONE PAGE FORWARD
CTRL R	(REPLACE) SEARCH AND REPLACE
CTRL S	(SEARCH) SEARCH ONLY
CTRL T	(TOP) GO TOP OF SCREEN
CAPS LOWR	SHIFT LOCK
CTRL U	UNDERLINE TOGGLE
CTRL V	SPECIAL PRINT CHARACTERS
CTRL W	DELETE ALL BEFORE CURSOR
CTRL X	DELETE ALL TEXT
CTRL Y	DELETE ALL AFTER CURSOR
CTRL Z	GO TO END OF LINE
ESC	EXIT EDITOR
CTRL ↑	MOVE CURSOR UP
CTRL ↓	MOVE CURSOR DOWN
CTRL ←	MOVE CURSOR TO LEFT
CTRL →	MOVE CURSOR TO RIGHT
(RETURN)	INSERT CARRIAGE RETURN
sft-DEL	DELETE NEXT LINE
sft-INS	INSERT LINE AT CURSOR
CTRL TAB	CLEAR TAB AT CURSOR
sft-TAB	SET TAB AT CURSOR
CTRL DEL	DELETE A CHARACTER
CTRL INS	INSERT A CHARACTER
sft-CLEAR	GO TO BEGINNING OF TEXT
DEL	DELETE LAST CHARACTER
TAB	TAB TO NEXT TAB STOP

SPECIAL PRINT CHARACTERS

*	SUBSCRIPT	$n	PRINT STRING n
^	SUPERSCRIPT	#n	PRINT NUMBER n

FORMAT REPRESENTATIONS		DEFAULT VALUES	
d	default values	d	(no number needed)
r	reset standard	r	(no number needed)
t	top margin	t	5 spaces
m	left margin	m	10 spaces
j	justification	j	1 (justify)
w	set line width	w	64 characters
l	line spacing	l	l (single spacing)
p	printed lines/page	p	56 lines
s	stop	s	0 (no stop)
f	set type font	f	0 (10 cpi)
a	margin adjust	a	0 (no adjust)
b	bottom margin	b	5 spaces
n	set page number	n	0 (not printed)

Editor's Note: The manufacturer provided the following updates, now included in the standard Letter Perfect 2.0. Our review panelists did not work with version 2.0. — RTM

Manufacturer's Update

Letter Perfect Version 2.0 differs from the earlier version in the following manner:

1. Fonts — The following enhancements have been made with regard to the two different printers supported by this program.

A. Atari 825 Printer and Centronics 737-739 have the following changes. The PROPORTIONAL FONT of this printer is not the default font in the program. The PROPORTIONAL FONT is right justified as a default value. Font 1 is the condensed font of 16.7 characters per inch and Font 2 is now the 10 character per inch font. All of these fonts are right justified as a default value, and can be used as indicated in the manual. Boldface or expanded print can now be used within the body of a line without regard to other types of fonts also appearing in that line. Right justification will remain in effect. The left margin is now set at a default value of 12 and the width defaults to 78. The use of the adjusted margin may cause minor difficulties. The left justification of the adjusted margin may cause some variance because all spaces in the margin are twelve dots wide. This font allows for variation in the individual character and therefore the alignment may not be as straight as desired. To obtain optimal results it is best to experiment.

B. Epson MX-80, MX-100, GRAFTRAX — These printers may now be used with this program with the following changes noted. If you have the new GRAFTRAX Proms as sold by Epson, you may use the additional characteristics of Italics characters. Italics are turned on by using the superscript command (CTRL-V:) as described in the manual. Italics are turned off by doing a subscript (CTRL-V*). Underlining may be performed on short words by using the underline toggle (CTRL-U). If an excess number of characters are underlined the printer may time out. Boldface may now be used in the main body of the text.

2. Header and Footer Spacing — The spacing between headers and footers and the main body of the text is now variable. The defaults are 4 spaces for headers and 4 spaces for footers. They may be changed by entering a lower case h followed by the spacing for a header in a format line (starting with a CTRL-F) and footer spacing can be changed by using a lower case z in a format line. ©

Atari Disk File Dump

Robert W. Baker
Atco, NJ

Here's another handy utility program for the Atari 800, for anyone with an 810 or 815 disk drive. It provides a hexadecimal dump of *any* disk file along with an ASCII representation of any valid ASCII characters. With this program you can quickly examine how Atari BASIC stores programs and data on diskette. This could be extremely valuable when debugging programs that write or read disk data files.

The program was written to provide only printed output, since most dumps would be too large for the display. Also, the printed output was formatted for the 80-column Atari 825 printer. If you only have the 40-column Atari 820 printer, the program can be easily modified for the shorter line length. Simply shorten the heading lines in program lines 302 and 310, stopping at "7." Then change the loop count in line 600 from 16 to 8 to print eight bytes per line instead of 16. You might also want to shorten the filename printout and remove the CHR$(15) and CHR$(14) from line 300. That should be all you have to change for the 40-column format.

To use the program, enter the filename for the disk file to be dumped, such as: FILEDMP.BAS The drive number always defaults to 1. The filename will be printed at the top of the listing along with a byte count heading. The dump will then follow with 16 bytes per line. At the end of each line is the ASCII representation of any data that is a valid ASCII character. All unprintable characters are printed as periods in the ASCII field.

As the dump is being printed you can press any key on the keyboard to halt the output. Then press "C" to continue the dump, "R" to restart and select another file to be dumped, or "S" to stop the program and terminate the dump.

If the end of the disk file is reached, the program will indicate on the dump the end of file (EOF) was reached. You may want to note the TRAP statement in program line 228. When an error is detected, the program branches to line 900. A PEEK of location 195 checks for error number 136. If an end of file (error #136) was detected, the program returns to ask for another filename. Otherwise, the detected error is indicated on the display and the program terminates after closing all files.

```
10 REM ******************************
25 REM HEX DISK FILE DUMP
35 REM BY: ROBERT W. BAKER
40 REM 15 WINDSOR DR, ATCO NJ 08004
60 REM ******************************
70 GRAPHICS 0
100 DIM H$(16),S$(16),F$(16)
110 H$="0123456789ABCDEF"
130 F$="D1:"
150 OPEN #1,4,0,"K"
200 PRINT CHR$(125);"      H E X   F I L
 E   D U M P    ":? :?
210 PRINT "ENTER DISK FILE NAME"
220 INPUT S$:F$(4,14)=S$
225 IF S$="" THEN 800
228 TRAP 900
230 OPEN #5,4,0,F$
280 OPEN #2,8,0,"P"
290 PRINT CHR$(125);"DEPRESS ANY KEY TO
HALT PRINTER":PRINT #2
300 PRINT #2;CHR$(15);"HEX DUMP OF FILE
-->   ";F$;CHR$(14):PRINT #2
302 PRINT #2;"BYTE#  0  1  2  3  4  5  6
  7  8  9  A  B  C  D  E  F  "
310 PRINT #2;"-------------------------
-------------------------"
320 POKE 764,255
370 V=INT(A/256):GOSUB 1000
375 V=A-(V*256):GOSUB 1000
380 PRINT #2;": ";
400 S$=""
600 FOR X=1 TO 16:GET #5,V
610 GOSUB 1000:PRINT #2;" ";
615 S$(X)=".":IF (V>31) AND (V<123) THEN
 S$(X)=CHR$(V)
620 A=A+1:NEXT X:PRINT #2;"    ";S$
640 IF PEEK(764)=255 THEN 370
650 GET #1,X
700 POKE 752,1:PRINT
705 PRINT "CONTINUE, RESTART, OR STOP (C
,R,S) ?";
710 GET #1,X:IF X=67 THEN 290
730 IF X=82 THEN 990
740 IF X<>83 THEN 710
800 POKE 752,0:CLOSE #1:CLOSE #2:CLOSE #
5:END
900 V=PEEK(195):IF V<>136 THEN PRINT "ER
ROR# ";V:GOTO 800
910 FOR A=X TO 16:PRINT #2;"   ";:NEXT A

920 PRINT #2;"    ";S$:PRINT #2:PRINT #2
;"EOF"
990 CLOSE #2:CLOSE #5:GOTO 200
1000 H=INT(V/16):L=V-(H*16)
1010 PRINT #2;H$(H+1,H+1);H$(L+1,L+1);
1020 RETURN
```

©

Atari Program Library

Ron and Lynn Marcuse
Freehold, NJ

One of the most difficult aspects of owning a home/personal computer is maintaining an accurate catalog of programs and data files. We first attacked this problem through the use of a home-grown data base which handled the program library in addition to other data. But, alas, it became too time consuming to manually update the data base as new programs were added or changes made. The logical extension to this concept was to automate the cataloging process. To accomplish this, one must compare the index of the disk to the program library file, adding or deleting records on the file as the comparison warrants. The ease of OPENing the ATARI's DOS directory from BASIC greatly facilitated the programming, but more on this later.

Besides the automatic cataloging function, other required features were:

1. Listing the directory from the program.

2. Cataloging Non-DOS disks.

3. Inquiry, Browse and formatted Print output of catalog records.

4. Maintaining other data not supplied by the disk directory (source, description, type, and date).

5. Sorting the file on any field (in machine language, for speed).

6. An auto-locate function to RUN any program.

7. Variable search criteria to locate any program.

8. Creating an internal label to

Table 1. List of variables.

NAME	SIZE	DESCRIPTION
AP$	21	"ATARI PROGRAM LIBRARY"
REC$	62	Program Library Record
VOL$	4	Disk (or other) Volume Number
DSN$	12	Data Set Name (Filespec)
DES$	22	Description
TYP$	7	Type (Game, Data, etc.)
SRC$	7	Source
DAT$	6	Date (Entry, Version, etc.)
SEC$	3	Number of Sectors
X$	var	Input String (Main Prog and Sort)
IN$	20	Temporary String Storage
PL$,PC$		Filespecs (D:PROGLIB.DB & D:DISK.CAT)
SV$	20	Search Value
SS,SE		Search (and Sort) Start, End Positions
SK		Search Key
B		Transaction Type
R		Return Line Number from TRAPped Error
ST		IOCB STATUS Value
I,J,L,N		Counting for Loops, Lines, etc.
NP		Number of Files in Directory
IO		Input/Output Type
SEC,BYT		Sector, Byte values for NOTE, POINT
P		Output Type (1=Inqy, 2=Browse, 3=Print)
EOF		Record Counter

Table 2. Key conversion (Epson MX-80 printer).

SYMBOL	ASCII VAL (Dec)	KEY SEQUENCE (ATARI)	DESCRIPTION
[A]	0	CON; , (comma)	Null; End of Tab Set Seq
[B]	9	CON; I	Horizontal Tab
[C]	14	CON; N	Print Double Width Characters
[D]	27,68	ESC;ESC;D	Set Tab (followed by Tab Positions and NULL Char)
[E]	253	ESC;CON;2	Console Bell
[F]	125	ESC;CON;CLR	Clear Screen
[G]	29	ESC;CON; =	Move Cursor Down
[H]	10	CON; J	Line Feed

Note: The string of X's in line 3090 are the printer tabs. Type the ASCII characters for these decimal values: 6,20,39,53,62,70 (i.e. 6 is CON;F)

Program 1.

```
10 DIM AP$(21),PC$(10):AP$="ATARI PROGRAM LIBRARY"
12 REM ** RON MARCUSE, FREEHOLD NJ **
30 OPEN #2,4,0,"K:":POKE 82,0:? "[E]"
50 DIM REC$(62),VOL$(4),DSN$(12),DES$(22),TYP$(7),SRC$(7),DAT$(6),SEC$(3)
60 DIM X$(500),IN$(20),SV$(20),PL$(12):PC$="D:DISK.CAT":PL$="D:PROGLIB.DB"
100 GRAPHICS 0:POKE 16,64:POKE 53774,64:? :? ,AP$:? ,"[G] CATALOG OPTIONS"
110 ? "[G] 1 AUTO CATALOG        5 UPDATE RECORD"
120 ? " 2 LIST DIRECTORY      6 SORT LIBRARY"
130 ? " 3 ADD DISK (NON DOS)  7 RUN PROGRAM"
140 ? " 4 INQUIRY/LIST        8 END SESSION"
160 GOSUB 6900:TRAP 160:B=VAL(CHR$(A)):TRAP 40000:IF B<1 OR B>8 THEN 160
170 B=B+1:IF B>3 THEN 500
180 ? :IF B=3 THEN 300
190 ? " INSERT DISK TO BE CATALOGED":GOSUB 6910
200 TRAP 240:OPEN #3,4,0,PC$:TRAP 40000:INPUT #3,IN$
210 IF LEN(IN$)<14 OR IN$(1,10)<>PC$ THEN ? " ERROR- ";PC$:GOTO 250
220 B=1:VOL$=IN$(11,14):GOTO 290
240 CLOSE #3:R=200:STATUS #3,ST:IF ST<>170 THEN 9000
250 ? " ENTER DISK # (DNNN) =>";:INPUT VOL$:IF LEN(VOL$)=0 THEN 100
260 R=260:TRAP 9000:OPEN #3,8,0,PC$:? #3;PC$;VOL$
290 CLOSE #3:XIO 35,#3,0,0,PC$
300 R=300:TRAP 9000:OPEN #3,6,0,"D:*.*":L=0
310 TRAP 400:INPUT #3,IN$:TRAP 40000
320 IF LEN(IN$)<17 THEN 400
330 IF B=3 THEN ? ,IN$:L=L+15:GOTO 310
```

automatically identify a disk. All of these goals were achieved in the program which, incidentally, requires at least 24K RAM, one disk drive (810 or 815) and DOS II.

You did notice that there is a "II" after "DOS." ATARI has finally released the new version and, to say the least, it is a vast improvement. Not that we were unhappy with its predecessor, but it did tend to hide whenever one walked by with a can of "RAID" or "BLACK FLAG." Yes, there are bugs in DOS I, one of which leads the NOTE and POINT commands (needed to update any record) somewhere into *the twilight zone*. This program can be modified to work under DOS I, but the explanation would probably take up the rest of the magazine. If you plan to do any serious file processing, it would be advisable to pick up a copy of DOS II. There are other advantages as well, such as less RAM used through the auto-swap feature (the program and DOS share the same area).

The three listings represent the main program, the sort program and the machine language sort routine. The sort program is executed by a RUN statement, allowing the DIMensioning of the rather large string necessary to sort the file in. It loads the file into X$ and calls the machine language sort through the USR function in line 70. You may POKE the routine into storage using the third BASIC program. You must do a BINARY SAVE (DOS II function "K") with AUTORUN.SYS as the file name, 0600 and 066D (hex) as the starting and ending addresses. A possible alternative to this would be to key the FOR/NEXT loop and DATA statements into the BASIC sort program, with the loop as line 14.

The main program begins with the DIMensioning of strings and OPENing of the keyboard in lines 30-60. The strings and other variables are detailed in the accompanying table. Graphics mode 0 is set and the <BREAK> key is disabled (POKE 16,64:POKE 53774,64) in line 100. This begins the start of the primary option menu as well. An option (from 1 to 8) is selected and held in variable B. Note that B is increased by 1 in line 170, allowing "Auto-catalog" to split into two functions depending on whether or not the disk has already been cataloged. A "D:DISK.CAT" file, containing the volume number, is written on each as an internal label. If this file is found on any disk, the volume number is retrieved and a "re-catalog" function is performed. If not found, a new disk is assumed and the volume number is requested in line 250. One important note: there is a liberal use of subroutines in the program and, for expediency, we will get into them later.

The options that do not require a DOS directory (B>3) are shunted to line 500. Those that do are sent through the routine (lines 300-430) that OPENs the disk directory (IO=6) and stores it as substrings of X$. The file name is translated into the more familiar format FILENAME.EXT in lines 340-360. File names equal to DOS.SYS, DUP.SYS, MEM-.SAV or DISK.CAT are dropped at this point (Why catalog them?). Line 500 (ON B GOTO) then transfers program control to the routine that will process the requested option.

The "Re-catalog" function (B=1, lines 1000-1110), after OPENing the library file for both input and output operation (IO=12), extracts all records having that volume number. These are then compared to the directory string (X$). Changes and deletions are posted to the file using the NOTE and POINT commands. An asterisk (*) is moved to each directory member that was successfully matched to the library. At the conclusion of this procedure, control is passed to "Auto-catalog."

The "Auto-catalog" function

```
340 FOR I=3 TO 10:IF IN$(I,I)<>" " THEN NEXT I
350 DSN$(1,8)=IN$(3,10):DSN$(9,9)=" ":DSN$(I-1)=IN$(11,13)
360 IF DSN$(I-1)<>"   " THEN DSN$(I-2,I-2)="."
370 IF DSN$="DOS.SYS" OR DSN$="DUP.SYS" OR DSN$="MEM.SAV" OR DSN$="DISK.CAT" THEN 310
380 X$(L+1,L+12)="            ":X$(L+1,L+12)=DSN$
390 X$(L+13,L+15)=IN$(15,17):L=L+15:GOTO 310
400 CLOSE #3:NP=L/15
410 ? :? " FILES FOUND= ";NP;",  FREE SPACE= ";IN$(1,3)
420 IF B<3 THEN ? " DISK IS # ";VOL$:? " INSERT D:PROGLIB.DB"
430 GOSUB 6910:IF NP=0 THEN 100
500 ON B GOTO 1000,1200,100,2000,3000,4000,5000,5500,900
900 GRAPHICS 0:END
950 GOSUB 6250:GOTO 100
960 POP :GOTO 100
990 FOR I=1 TO 300:NEXT I:RETURN
1000 REM ** RE-CATALOG
1010 IO=12:GOSUB 6200:SV$=VOL$:SK=1:SS=1:SE=4:EOF=0:L=0
1020 GOSUB 7000:IF SK=9 THEN 1190
1030 FOR N=1 TO NP:IF X$(N*15-14,N*15-3)=DSN$ THEN 1100
1040 NEXT N:REC$(62)="D":? " ";DSN$;" DELETED ON ";VOL$:GOTO 1110
1100 REC$(59,61)=X$(N*15-2,N*15):X$(N*15,N*15)="*":L=L+1
1110 POINT #3,SEC,BYT:? #3;REC$:GOTO 1020
1190 GOSUB 6250:IF L=NP THEN 100
1200 REM ** AUTO CATALOG
1210 IO=9:GOSUB 6200
1220 FOR N=1 TO NP:IF B=1 AND X$(N*15,N*15)="*" THEN 1240
1230 DSN$=X$(N*15-14,N*15-3):SEC$=X$(N*15-2,N*15):GOSUB 6800
1240 NEXT N:GOTO 950
2000 REM ** MANUAL ADD
2010 IO=9:GOSUB 6200:GOSUB 6800:GOTO 950
3000 ? "[F][G] ";AP$;" - INQY/LIST":N=8:GOSUB 6500:?
3020 ? " OUTPUT:","1. INQUIRY":? ,"2. BROWSE":? ,"3. LISTING"
3040 GOSUB 6900:TRAP 3040:P=VAL(CHR$(A)):TRAP 40000:IF P<1 OR P>3 THEN 3040
3050 L=0:IO=4:GOSUB 6200:EOF=0:IF P<3 THEN 3100
3090 R=3090:TRAP 9020:OPEN #4,8,0,"P:":TRAP 40000:? #4;"[D]XXXXXX[A]"
3100 GOSUB 7000:L=L+1:IF SK>8 THEN 3300
3110 ON P GOTO 3120,3160,3200
3120 GOSUB 7600:?
3130 L=0:? " (E=END) OR";:GOSUB 6910:IF CHR$(A)="E" THEN 950
3140 GOTO 3100
3160 IF L=1 THEN ? "[F][G]VOL  FILE ID      DESCRIP":?
3170 ? VOL$;" ";DSN$;DES$:IF L=19 THEN 3130
3180 GOTO 3100
3200 IF L>1 THEN 3220
3205 ? #4;"[B][C]   ATARI PROGRAM LIBRARY[H]"
3210 ? #4;"DISK#[B]PROG/FILE ID[B] DESCRIPTION[B]TYPE[B]SOURCE[B]DATE  SECTORS[H]"
3220 ? #4;VOL$;"[B]";DSN$;"[B]";DES$;"[B]";TYP$;"[B]";SRC$;"[B]";DAT$;"[B]";SEC$
3230 IF L>55 THEN L=0:? #4;CHR$(12)
3240 GOTO 3100
3300 IF P=3 THEN CLOSE #4
3310 IF P=2 THEN GOSUB 6910
3320 GOTO 950
4000 ? "[F][G] ";AP$;" - RECORD UPDATE":N=8:GOSUB 6500
4010 IO=12:GOSUB 6200:EOF=0
4020 GOSUB 7000:IF SK>8 THEN 950
4030 GOSUB 7600:? :? " TYPE FIELD # TO UPDATE, D TO DELETE"
4050 GOSUB 6910:IF CHR$(A)="D" THEN REC$(62)="D"
4060 TRAP 4300:C=VAL(CHR$(A)):TRAP 40000:IF C<1 OR C>7 THEN 4300
4100 RESTORE 9910:FOR I=1 TO C:READ IN$:NEXT I:?
4110 ? " ENTER NEW":GOSUB 6040+C:GOTO 4030
4300 GOSUB 6100:POINT #3,SEC,BYT:? #3;REC$:GOTO 4020
5000 ? "[F][G] ";AP$;" - SORT/COMPRESS":N=7:GOSUB 6500
```

(lines 1200-1240) OPENs the file for output (append, IO=9) operation. All new entries, as per the X$ directory, are shuffled through the proper subroutines and then added to the file. Option 1 (B=1 or 2) travels this route, with (1) being "Re-cataloged" first. The significance of the asterisk on matched directory members becomes apparent in the bypass on line 1220. The file is then CLOSEd and we return to the option menu.

Option 2 (B=3) consists of displaying the DOS directory and then returning to the menu. The remaining options (B>3), not requiring any help from the directory, go straight into their respective procedures. Option 3 (B=4) handles the manual addition of library records to the file. This could be utilized for Non-DOS disks or even cassette tapes. Line 2010, helped by several subroutines, does all of the processing.

The inquiry and print option (B=5) is handled in lines 3000-3320. The search strategy (GOSUB 6500) and output mode are selected, the file is OPENed for input (IO=4) and, if applicable, the print tabs are set. The file is actually read in the subroutine at line 7000, with only the records matching the search key being passed back. An inquiry goes to line 3120, the browse (19 records per screen) to line 3160 and print (55 records per page) to line 3200. For the multiple record options, the variable L is the line counter. When end of file is reached at line 7010, the search key (SK) is set to "9" indicating completion, the file and printer are closed, and we are back at the menu.

The Update option (B=6, lines 4000-4300) would be used for changing or deleting library records. The search key is selected, file OPENed (IO=12) and records read like the inquiry, but here only the full record is displayed. At this point (line 4030), typing the field number will cause

```
5010 ? " TYPE Y TO SORT ON FIELD # ";SK:GOSUB 6910
5020 IF CHR$(A)<>"Y" THEN 100
5040 POKE 207,SS-1:POKE 208,SE-1
5050 ? " LOADING SORT PROGRAM":RUN "D:PROGSORT"
5500 REM RUN PROG
5510 ? :? " ENTER PROG ID ==>";:INPUT SV$:IO=4:IF LEN(SV$)=0 THEN 100
5520 GOSUB 6200:EOF=0:SS=5:SE=4+LEN(SV$):SK=2:GOSUB 7000:IF SK=9 THEN 950
5530 ? :? " INSERT DISK ";VOL$;" TO RUN ";DSN$
5540 ? :? " TYPE 'Y' TO RUN":GOSUB 6910:IF CHR$(A)<>"Y" THEN 950
5550 IN$(3)=DSN$:IN$(1,2)="D:"
5560 ? :? " LOADING ";IN$:TRAP 5570:RUN IN$:TRAP 40000
5570 ? :? " ";IN$;" NOT ON DISK":GOSUB 990:GOTO 950
6000 ? "[F][G] TO ADD ";DSN$;", ENTER:"
6010 RESTORE 9910:FOR I=1 TO 7:READ IN$
6020 IF B<3 AND (I=1 OR I=2 OR I=7) THEN 6040
6030 ? :GOSUB 6040+I
6040 NEXT I:RETURN
6041 ? "|----| ";IN$:INPUT VOL$:RETURN
6042 ? "|------------| ";IN$:INPUT DSN$:RETURN
6043 ? "|----------------------| ";IN$:INPUT DES$:RETURN
6044 ? "|-------| ";IN$:INPUT TYP$:RETURN
6045 ? "|-------| ";IN$:INPUT SRC$:RETURN
6046 ? "|------| ";IN$:INPUT DAT$:RETURN
6047 ? "|---| ";IN$:INPUT SEC$:RETURN
6100 FOR I=1 TO 61:REC$(I,I)=" ":NEXT I
6110 REC$(1,4)=VOL$:REC$(5,16)=DSN$:REC$(17,38)=DES$:REC$(39,45)=TYP$
6120 REC$(46,52)=SRC$:REC$(53,58)=DAT$:REC$(59,61)=SEC$:RETURN
6200 R=6200:TRAP 9000:IF IO<>4 THEN XIO 36,#3,0,0,PL$
6210 OPEN #3,IO,0,PL$:TRAP 40000:RETURN
6250 R=6250:TRAP 9000:CLOSE #3
6260 IF IO<>4 THEN XIO 35,#3,0,0,PL$
6270 TRAP 40000:RETURN
6500 ? "[G] KEY:":RESTORE 9910
6510 FOR I=1 TO N:READ IN$:? ,I;" ";IN$:NEXT I:? ,"E END":GOSUB 6900
6540 TRAP 960:SK=VAL(CHR$(A)):TRAP 40000:IF SK<1 OR SK>N THEN 960
6550 IF SK=8 THEN 6590
6560 RESTORE :FOR I=1 TO SK:READ SS,SE:NEXT I:IF B=7 THEN 6590
6570 ? " ENTER VALUE";:INPUT SV$:IF LEN(SV$)<1 THEN 960
6580 IF SS+LEN(SV$)-1>SE THEN 6570
6585 SE=SS+LEN(SV$)-1
6590 RETURN
6800 GOSUB 6000:GOSUB 6100:REC$(62)="*":GOSUB 7600
6810 ? "[G] TYPE 'Y' IF OK ":GOSUB 6910:IF CHR$(A)<>"Y" THEN 6800
6820 ? #3;REC$:RETURN
6900 ? :? " SELECT OPTION ==>";:GET #2,A:? CHR$(A):RETURN
6910 ? " PRESS ANY KEY TO CONTINUE";:GET #2,A:? CHR$(A):RETURN
7000 IF B=1 OR B=6 THEN NOTE #3,SEC,BYT
7010 TRAP 7060:INPUT #3,REC$:TRAP 40000:IF REC$(62)="D" THEN 7000
7020 IF SK=8 THEN 7040
7030 IF SV$<>REC$(SS,SE) THEN 7000
7040 EOF=EOF+1:VOL$=REC$(1,4):DSN$=REC$(5,16):DES$=REC$(17,38):TYP$=REC$(39,45)
7050 SRC$=REC$(46,52):DAT$=REC$(53,58):SEC$=REC$(59,61):RETURN
7060 SK=9:? :? " RECORDS FOUND= ";EOF:GOSUB 990:RETURN
7600 ? "[F][G]";:RESTORE 9910:FOR I=1 TO 7
7610 READ IN$:? " ";I;" ";IN$;:GOSUB 7610+I:NEXT I:RETURN
7611 ? VOL$:RETURN
7612 ? DSN$:RETURN
7613 ? DES$:RETURN
7614 ? TYP$:RETURN
7615 ? SRC$:RETURN
7616 ? DAT$:RETURN
7617 ? SEC$:RETURN
```

updating of that field and "D" the deletion of the record. Any other character will write (using POINT) the record.

The sort (B = 7, lines 5000-5050), as stated earlier, is executed through a RUN "D:PROGSORT statement. Before this is done, the sort key is selected and its offset (beginning and ending positions) is POKEd into locations 207 and 208 (decimal) for use by the machine language program. These addresses, as well as those on page 6 (1536-1791) containing the sort program, are safe from the ravaging effects of RUN, LOAD and NEW. The RUN PROGRAM option (B = 8, lines 5500-5570) adds a little touch of class to the library. By inserting the program name when requested, it will tell you which disk to load and will then RUN it. Obviously, it will only function with BASIC programs that have been SAVEd on disk. The final option (B = 9) terminates the system. The BREAK key, disabled in line 100, could have disastrous effects on the file if used to end the program at the wrong time.

A few notes on the subroutines that do most of the work in the program. The routine starting at line 6000 is used for the input of data for both adding and updating records. The labels for the individual fields, being DATA statements, are READ during the FOR/NEXT loop in line 6010. The library record (REC$) is built from the individual fields at line 6100. Lines 6200 and 6250 OPENs and CLOSEs the file. The search and sort keys are built at line 6500, using the same DATA statements as above for the headings. The positions of the fields are contained in another DATA statement. Lines 6900 and 6910 are prompts. The library record is READ and moved into its elements in line 7000-7060. The variables SV$, SS, SE and SK are used in the search process. The full screen display of the record is taken care of by 7600-7617, once again using the

```
9000 STATUS #3,ST:CLOSE #3:? " CHECK DISK DRIVE ERROR ";ST:GOSUB 6910:GOTO R
9020 STATUS #4,ST:CLOSE #4:? " CHECK PRINTER ERROR ";ST:GOSUB 6910:GOTO R
9900 DATA 1,4,5,16,17,38,39,45,46,52,53,58,59,61
9910 DATA DISK #  ,PROG ID ,DESCRIP ,TYPE   ,SOURCE ,DATE   ,SECTORS ,ALL RECORDS
```

Program 2.

```
10 REM ** ATARI PROGRAM LIBRARY SORT **
11 REM **  R MARCUSE
15 A=FRE(0)-800
20 DIM X$(A),REC$(62),AP$(12):AP$="D:PROGLIB.DB":? " LOADING FILE"
30 TRAP 130:OPEN #3,4,0,AP$:N=0
40 TRAP 60:INPUT #3,REC$:TRAP 40000:IF REC$(62)="D" THEN 40
50 N=N+1:X$(N*62-61,N*62)=REC$:GOTO 40
60 CLOSE #3:? " RECORDS LOADED= ";N;", BEGIN SORT"
70 IF N>1 THEN A=USR(1536,ADR(X$),N)
80 ? "[E] SORT FINISHED, SAVING FILE"
90 XIO 36,#3,0,0,AP$:OPEN #3,8,0,AP$
100 FOR I=1 TO N:REC$=X$(I*62-61,I*62):? #3;REC$:NEXT I
110 CLOSE #3:XIO 35,#3,0,0,AP$
120 ? "[E] LOADING PROGLIB":RUN "D:PROGLIB"
130 STATUS #3,ST:CLOSE #3:? " CHECK DISK, ERROR ";ST
140 ? " PRESS RETURN TO CONTINUE":INPUT REC$:GOTO 30
```

Program 3.

```
10 REM ** PROGLIB MACHINE LANG SORT  **
11 REM **      R MARCUSE             **
12 REM BINARY SAVE WITH FILESPEC=AUTORUN.SYS
20 FOR I=0 TO 109:READ A:POKE 1536+I,A:NEXT I
100 DATA 104,104,133,216,104,133,215,104,133,213,104,133,212,169,0,133,209,133
110 DATA 214,162,1,165,215,133,205,165,216,133,206,24,165,205,133,203,105,62
120 DATA 133,205,165,206,133,204,105,0,133,206,164,207,177,205,209,203,144,11
130 DATA 240,2,176,28,196,208,176,24,200,144,239,169,1,133,209,160,62,136,177
140 DATA 205,72,177,203,145,205,104,145,203,192,0,208,241,232,224,0,208,2,230
150 DATA 214,228,212,208,188,165,213,197,214,208,182,165,209,201,0,208,160,96
```

heading DATA statements. Finally, the error routines for the disk drive and printer are found at lines 9000 and 9020. The variable R is the return line number.

As there are several unprintable characters in the programs, we have substituted others in their place in the BASIC listings. Take a peek at the conversion table before typing. For a better visual effect, the characters enclosed by a box should be typed in reverse video. The two programs call each other by name, so please save them by the names "D:PROGLIB" and "D:PROGSORT". Some last thoughts: The machine language sort routine will not be in storage unless you put it there by either powering up (AUTORUN.SYS will boot in) or doing a DOS binary load. Create the PROGLIB file by typing (in direct mode):

OPEN #3,8,0, "D:PROGLIB.DB": CLOSE #3:XIO 35,#3,0,0, "D:PROGLIB.DB".

Good luck. ©

MATCH — A Game Of Memory And Timing

Ron Walker
Smithville, Ontario

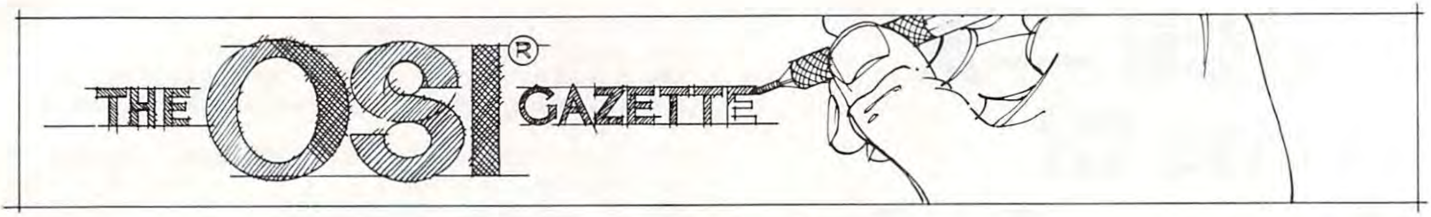
There are nine different skill levels to this game, with level one being the most difficult and nine being the easiest.

The object of the game is to repeat a pattern of musical notes and cursor positions that are randomly selected by the computer. Sounds simple, but at the highest level, the notes occur so rapidly that only the sharpest of minds can cope with it. Try it.

NOTE: Any numbers in brackets in quotes in the program listing are the number of blank spaces to leave. The "rvs" means leave one reverse space. Also, any words on brackets are what they say they are. e.g. PRINT "(clear" means to clear the screen (esc-shift-clr) etc. Good luck.

```
100 OPEN#1,4,Ø,"K:"
110 GRAPHICS Ø:POKE752,1:DIM NT(4),HOR(4)
    ,VIR(4),CHIME(5Ø)
120 DIMA$(4Ø):A$="    (39)      "
130 POSITION15,6:PRINT"MATCH"
140 POSITION1Ø,16:PRINT"ENTER LEVEL (1-9)
    "
150 GET#1,VAR:TRAP51Ø:LEVEL=VAL(CHR$(VAR)
    )*1Ø
160 FORX=1TO4:READ A,B,C:NT(X)=A:HOR(X)=B
    :VIR(X)=C:NEXTX
170 DATA 1Ø,19,4,5Ø,9,12,1ØØ,29,12,15Ø,19
    ,21
180 PRINT " (CLEAR)": FORX=5TO2Ø:POSITION
     19,X:PRINT"|":NEXTX
190 FORX=1ØTO2Ø:POSITIONX,12:PRINT"-":NEX
    TX:POSITION 19,12:PRINT"+"
200 POSITION2,Ø:PRINT"HERE ARE THE NOTES:
    ":FORDELAY=1TO5ØØ:NEXTDELAY:POSI
    TION2,Ø:PRINTA$
210 FORX=1TO4:POSITION HOR(X),VIR(X):PRIN
    T"RVS":GOSUB47Ø:POSITIONHOR(X),V
    IR(X):PRINT" "
215 NEXT X
220 POSITION2,Ø:PRINT"PRESS BUTTON TO STA
    RT WITH YOUR FIRST NOTE":SOUNDØ,
    Ø,Ø,Ø
230 IFSTRIG(Ø)<>ØTHEN23Ø
240 POSITION Ø,Ø:PRINTA$:PRINTA$
250 GUESS=1:CHIME(GUESS)=INT(RND(1)*4)+1
260 FORX=1TOGUESS:GOSUB44Ø:NEXTX:SOUNDØ,Ø
    ,Ø,Ø
270 POSITION1,22:PRINT"YOUR TURN.  PRESS
    BUTTON TO START"
280 IF STRIG(Ø) <>ØTHEN28Ø
290 POSITION 1,22:PRINT"(DELETE LINE) ENT
    ER NOTES"
300 FORX=1TOGUESS:POKE77,Ø
310 TONE=STICK(Ø):SOUNDØ,Ø,Ø,Ø
320 IFTONE=14THENTONE=1Ø:GOTO37Ø
330 IFTONE=13THENTONE=15Ø:GOTO37Ø
340 IFTONE=7THENTONE=1ØØ:GOTO37Ø
350 IFTONE=11THENTONE=5Ø:GOTO37Ø
360 GOTO31Ø
370 IFNT(CHIME(X))=TONE THEN GOSUB 44Ø:GO
    TO4ØØ
380 GOTO41Ø
400 NEXTX:GOTO48Ø
410 PRINT"(CLEAR)NOT QUITE RIGHT.  YOU GO
    T";GUESS;" NOTES IN A ROW":PRINT
    "TRY AGAIN?"
420 GET#1,VAR:IFVAR=89 THEN GRAPHICS Ø:EN
    D
430 GOTO18Ø
440 SOUNDØ,NT(CHIME(X)),1Ø,1Ø
450 POSITIONHOR(CHIME(X)),VIR(CHIME(X)):P
    RINT"RVS":FORDELAY=1TOLEVEL:NEXT
    DELAY
460 POSITIONHOR(CHIME(X)),VIR(CHIME(X)):P
    RINT" ":RETURN
470 SOUNDØ,NT(X),1Ø,1Ø:FORDELAY=1TO5ØØ:NE
    XTDELAY:RETURN
480 GUESS=GUESS+1:SOUNDØ,Ø,Ø,Ø:POSITION1,
    22:PRINTA$
490 POSITION1,Ø:PRINT"OK SO FAR.  NOW I A
    DD ANOTHER ONE":CHIME(GUESS)=INT
    (RND(1)*4)+1
500 FORDELAY=1TO5ØØ:NEXTDELAY:POSITIONØ,Ø
    :PRINTA$:GOTO26Ø
510 PRINT"A NUMBER!":FORDE=1TO5ØØ:NEXTDE:
    PRINT"(UP) (14)":GOTO14Ø
```

©



Calling BASIC Commands From Machine Language Routines

William Taylor
Leavittsburg, OH

While working on a tape operating system (TOS) for my OSI C1P and a Stringy Floppy tape drive, many unknown, but desired, features were needed to interface ROM BASIC and the TOS. First, I wanted the TOS to always have command of BASIC's LOAD and SAVE routines. Second, I wanted always to return to the TOS whenever a BASIC program had been loaded into the BASIC workspace. Third, I wanted to go directly from the TOS and RUN a BASIC program that was in the BASIC workspace. In addition, I wished to exit the TOS to the ML Monitor; write a file directory; store the directory on tape; retrieve the directory; and write or load language tapes into the C1P using file marks.

Since the Stringy Floppy tape drives require that all programs stored on tape have file marks or numbers, I needed to free the C1P from ROM BASIC in order to create files on the tape for all programs stored on the tape. The TOS could be written in machine language. The TOS would generate the file numbers under the control of the user, but interfacing the TOS to ROM BASIC was the problem that I faced and pondered for several weeks. How the OSI ROM BASIC and the TOS were interfaced brought several interesting points to light that could be useful in other programming tasks.

Let me summarize. First, calling BASIC commands and executing BASIC programs can be handled from machine language routines. Also, we may LIST, SAVE, LOAD, and exit BASIC to our machine language routines without any USR function call. How these commands can be executed from a machine language routine will become clear with some new knowledge of how BASIC's interpreter works. Let's start with some facts about the BASIC interpreter and how BASIC commands are executed.

Let's look at BASIC's LOAD and SAVE flags and see how they are used to determine if BASIC programs are to be listed to the CRT or to the Cassette port and if the keyboard or the Cassette input port will be the input device.

BASIC's Immediate Mode Commands

BASIC commands are usually executed when input from the keyboard is entered. For example, when you type RUN followed by a carriage return any BASIC program in the workspace will be executed or start to run, starting at the first line of the program. Notice that I said type RUN! This type of command is known as an immediate mode command. If you had typed a number before the RUN command the C1P would have responded with OK. The program would not run but the line of text would have been saved or entered into the program memory. To understand what happens in either the programming mode or the immediate mode we must know how BASIC interprets the code input by the operator. To do this let's look inside BASIC and examine some of what happens during the course of any type of code execution.

At the beginning of system memory is what has become known as zero page. This memory area consists of the first 256 locations of low memory. OSI BASIC uses this area of memory as a scratch pad. OSI BASIC uses page locations $0013 through $005A as what is known as the BASIC Input Line Buffer. What is the Input Line Buffer? This area of low memory is used by BASIC to temporarily store any input code from the user. The code input by the user in the Input Line Buffer will be examined by BASIC to determine what the code's destiny will be. When the user terminates a line of code with a carriage return, the destination of the code input by the operator depends on two factors. First, if the code began with a line number

OSI AARDVARK OSI
NOW MEANS BUSINESS!

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

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

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

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

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

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

NEW-NEW—NEW TINY COMPILER

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry – 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic – about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -,=,*,/, (,), <,>,Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.
TINY COMPILER – $19.95 on tape or disk

THE AARDVARK JOURNAL

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

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

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

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

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

* * * SPECIAL DEAL – NO LESS! * * *

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

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

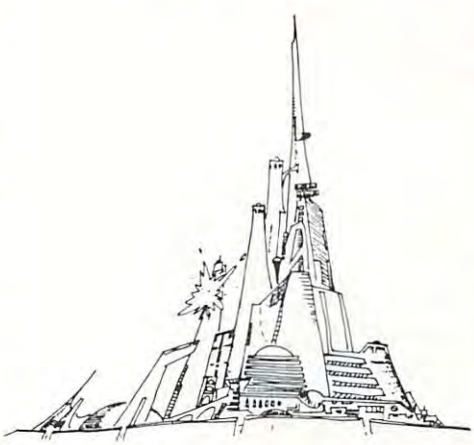
SUPERDISK II

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

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

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

AND FUN, TOO!



BOOKKEEPING THE EASY WAY – WITH BUSINESS I

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

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

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

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

GAMES FOR ALL SYSTEMS

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

NEW – NEW – NEW

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

NIGHT RIDER - You've seen similar games in the arcades. You see a winding twisting road ahead as you try to make time and stay on the road. NIGHT RIDER uses machine code to generate excellent high speed graphics - by the same author as MINOS.
NIGHT RIDER – $12.95 cassette only

THIEF - Another machine code goody for the C1P cassette only. You must use mobile cannon to protect the valuable jewels in the middle of the screen from increasingly nasty and trigger happy thiefs. Fast action and fun for one or two players. THIEF $13.95 on C1 cassette only!

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

Please specify system on all orders

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

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

OSI OSI

this signals BASIC that the code must be saved as a BASIC program line. Second, if the code in the Input Line Buffer does not start with a numeral, then the code represents a BASIC immediate command or some error that the user made while typing at the keyboard. In either of the latter cases, the code will be immediately executed. If the code was a valid command, the command will be executed. If the input was an error, BASIC will respond with Syntax Error.

To demonstrate and reveal the format of the code placed in the Input Line Buffer, please examine the following example of an input line which will be considered a BASIC line of program text: 10 LIST. On examination of the Input Line Buffer, it would reveal the following code if no carriage return were typed after the line of text. Type in the line of code: 10 LIST. Do not enter a carriage return. BREAK the C1P. Call up Monitor Mode by typing M. Call address mode. Call memory location $0013. You will find that the code listed in the next example will reside at memory locations starting at $0013.

```
0013 31 = ASCII 1
0014 30 = ASCII 0
0015 20 = ASCII space
0016 4C = ASCII L
0017 49 = ASCII I
0018 53 = ASCII S
0019 54 = ASCII T
```

On examination of the code in the Input Line Buffer you will find that all the code will be the hexadecimal ASCII equivalent of the text entered at the keyboard.

The code stored in the Input Line Buffer will have a different appearance if you terminate the line with a carriage return. The code will appear in the Input Line Buffer as in the next example.

```
0013 99
0014 00
0015 20
0016 00
0017 49
0018 53
0019 54
```

Try entering the line 10 LIST (CR). Break the computer. Call $0013 and examine the code in the Input Line Buffer. As you can see BASIC has converted its contents.

Now let's try an Immediate Mode operation and examine the Input Line Buffer. First, clear BASIC workspace. Type NEW (CR). Next type LIST (CR). Break the computer and call Monitor Mode. As before, call $0013 and examine the code stored in the buffer. On examination you should find the following code:

```
0013 99
0014 00
0015 53
0016 00
0017 00
```

This data spells out the LIST command. The byte $99 is a Token for the keyword LIST. What is a Token? It is a single byte that represents a command or keyword. OSI BASIC has a Token for all BASIC keywords. Tokens are used by BASIC in immediate Mode or they are stored in all BASIC programs stored in the BASIC program workspace or BASIC source code table. For the sake of this article let's say that a Token describes to BASIC a keyword. A keyword is an indicator to BASIC as to what function BASIC must perform in the case of $99 (LIST) BASIC is told to LIST all the source code in the BASIC workspace.

The point that we have made with the examples indicates that, for BASIC to know what is expected, the proper code must be in the Input Line Buffer starting at $0013. We can use the facts just presented to make BASIC think an operator has entered an Immediate Mode command, but the command can be initiated from a machine language routine as you will see. We are not ready yet to use our new knowledge about the Input Line Buffer and Tokens as commands called from machine language routines. First we must learn some more facts about BASIC.

How does BASIC execute the code for commands in the Input Line Buffer? The code must be read by the BASIC Interpreter. On examination of a Zero page memory map, you will find a machine language routine which starts at $ 00BC. This routine is called a "PARSER." It is used to read a line of code, character by character, stored in the line buffer or code stored in a program line in the BASIC workspace. The Parser routine at $00BC looks at the first character of code in the buffer to see if the character is an ASCII numeral or not. If the first character were a numeral, the Parser tests each character until a non-numeral is found. If the first character is a numeral, the line of code in the buffer is recognized as a line of source code and will be stored in the source code table. When the Parser detects a non-numeral, the Parser routine hands the code to a routine that "Tokenizes" the line before the line is placed in the source code table or back into the input line buffer. If the first character in the buffer is a non-numeral, the parser determines that the input code must be an immediate mode command. If you recall the earlier examples, we demonstrated the keyword LIST entered as a program source line. First we examined the buffer without a carriage return. It was evident that the code was ASCII. Next, we entered a line of text ending with a carriage return and examined the data in the buffer. At this point, we found that the data was in a Tokenized form. As you can see, the BASIC interpreter had, in fact, converted the ASCII to a condensed (or Tokenized) line of code.

To understand how the parser routine interprets the source code (or the code in the Input

Line Buffer) please refer to Listing 1. The machine language parser routine shown in Listing 1 shows that memory locations $00C2, 00C3, and 00C4 contain an LDA direct instruction or AD 13 00. This instruction causes the 6502 accumulator to be loaded with the code at the first address of the Input Line Buffer. On initialization, (BASIC Cold Start) address $00C2, 00C3, and 00C4 will point to $0013 (the beginning of the Input Line Buffer). If you type RUN in Immediate Mode without a program in the BASIC workspace, address $00C2, 00C3, and 00C4 will contain AD 00 03. As you can see, the Parser now points to the beginning of the BASIC workspace.

At this point, enough knowledge about the Input Line Buffer and the parser routine has been presented to allow us to explore the possibility of implementing and executing BASIC Immediate Mode commands called from outside ROM BASIC using machine language routines.

Let us now experiment with the Input Line Buffer and the parser routine to see if we can actually call a BASIC Immediate Mode command from a machine language program. As I mentioned at the beginning of this article, I needed to call BASIC's LOAD and SAVE commands. Let's begin with these. First, let's try the SAVE command to demonstrate how it can be called from a machine language routine.

To use the SAVE command we must learn yet more facts about how BASIC functions. When the user wishes to save a program that is stored in the BASIC workspace, the SAVE command must be used. What happens when you type SAVE? When the command, SAVE, is entered at the keyboard and ended with a carriage return, the code will, of course, be placed in the Input Line Buffer as ASCII. When the carriage return is entered, BASIC examines the code and recognizes that this is an Immediate Mode command. The code in the Input Line Buffer will be Tokenized and placed back in the Input Line Buffer. The Input Line Buffer would not contain:

$0013 94 = TOKEN FOR SAVE
$0014 00 = NULL
$0015 53
$0016 00 = NULL

Now, on examination of the Parser routine at address $00C2, 00C3, and 00C4, you will find that the Parser has read the code located at address $0013 and found a Token for the keyword SAVE, and that BASIC has executed the comand. When the SAVE command was executed, BASIC performed the task of setting what is called the SAVE flag. This flag tells the computer that any data sent from BASIC will be sent to the cassette port and to the screen. The SAVE flag is located at $0205. If the contents of $0205 are set to $00, then output from BASIC will be listed to the screen.

If the SAVE flag contains $01 then the cassette port along with the screen will be activated.

We may use these facts to call the BASIC SAVE command from a machine language routine. Let me demonstrate with an example. Enter the machine language routine (Listing 2) into the computer. Now write a BASIC program into the computer. This program can be any program that you may have on hand, but a single program line will do for the demonstration. Exit BASIC and call the address of the machine language routine of Listing 2. Run the machine language routine. As you can see, the BASIC program that you entered into the computer was LISTed out to the screen of your monitor. Also, the program will be sent to the cassette port.

On examination of the Assembly Listing, notice that we have loaded the Input Line Buffer at $0013 with the Token for LIST ($94). Also notice that, in the Listing, we are setting the SAVE flag at $0205 to the the value of $01. We have set address $00C3 and $00C4 in the Parser routine to point to the beginning of the input line buffer. Finally, we call a routine in the BASIC interpreter located at $A4B5. This routine is called the LIST routine and will execute the LIST command when called by a BASIC program, Immediate Mode, or by a machine language calling routine. As you can see, we have programmed a SAVE and a LIST

Program 1.

```
 10 0000          ;
 20 0000          ;
 25 0000          ;
 30 0000          ;
 40 0000          ;      PARSER CODE
 50 00BC          *=$BC
 60 00BC E6C3     S0 INC $C3   INCREMENT LOW ADDR. BYTE
 70 00BE D002     BNE S1
 80 00C0 E6C4     INC $C4   INCREMENT HIGH ADDR. BYTE
 90 00C2 ADFFFF   S1 LDA $FFFF   LOAD WITH CODE CHARACTER
100 00C5 C93A     CMP #': CHECK FOR COLON (STATEMENT END)
110 00C7 B00A     BCS S2  IF YES BRANCH TO START NEW LINE
120 00C9 C920     CMP #'  ISIT A SPACE
130 00CB F0EF     BEQ S0  IF YES GET NEW CHARACTER
140 00CD 38       SEC    SET CARRY FLAG
150 00CE E930     SBC #$30  SUBTRACT $30
160 00D0 38       SEC    SET CARRY FLAG
170 00D1 E9D0     SBC #$D0  SET C FLAG FOR ASCII NUMBERS
180 00D3 60       S2 RTS  END ROUTINE.  CHARACTER NOW IN A
```

Program 2.

```
 10 0000          ;
 20 0000          ;
 30 0000          ;
 40 0000          ;
 50 0000          ;  BASIC SAVE COMMAND CALL
 60 0000          ;
 70 0000          ;
 80 0000          ;
 90 0000          ;
100 1000          *=$1000
110 1000 A901     START LDA #$01  VALUE SAVE FLAG=ON
120 1002 8D0502   STA $0205     ST0RE IN SAVE FLAG
130 1005 A999     LDA #$99    TOKEN LIST
140 1007 8513     STA $13    PUT IN LINE BUFFER
150 1009 A900     LDA #$00    NULL
160 100B 8514     STA $14    PUT BUFFER+1
170 100D 8516     STA $16    PUT BUFFER +3
180 100F A953     LDA #$53
190 1011 8515     STA $15    PUT BUFFER+2
200 1013 A914     LDA #$14    PARSER SCAN START LOW BYTE
210 1015 85C3     STA $C3    PUT IN PARSER
230 1017 A900     LDA #$00    PARSER SCAN HIGH BYTE
240 1019 85C4     STA $C4    PUT IN PARSER
250 101B 4CB5A4   JMP $A4B5    GOTO BASIC LIST ROUTINE
```

command into BASIC from outside ROM BASIC and caused its execution.

In a similar manner, let's call and execute a LOAD command from a machine language routine. Enter Listing 3 into the computer. Next bring up BASIC in Warm Start. (Type NEW (CR).) Exit BASIC. Call up the machine language routine for the LOAD command. Place a BASIC program tape into your cassette recorder, execute the machine language routine, and start your recorder on play. Your BASIC program will load into the computer as if called directly under BASIC.

On examination of Listing 3, you will find that the implementation of the LOAD command was very simple. We only need to set the LOAD flag to turn the system on for a BASIC load and jump to the Warm Start of BASIC.

Listing 4 will be used to implement the BASIC RUN command from a machine language program. As before, enter the machine language program into memory and then load a BASIC program into the BASIC workspace. Exit BASIC and call the machine language routine. Start the machine language program. The computer will jump to the BASIC program and run.

On close examination of Listing 4, you will see that we have used the same procedure to force a BASIC RUN command that we used in the SAVE and LOAD routines. We loaded the input line buffer with the Token for RUN, set the Parser scanner to start reading the code in the Input Line Buffer at $0013. With the RUN command it was found that two BASIC interpreter routines were needed to force the computer to execute the RUN command. These were the conversion routine at $A3A6 and the execution routine located at $A5F6.

At the beginning of this article, I said that an executive TOS could be written in machine language that could call BASIC commands. Also, it was mentioned that in order for the TOS to be truly an executive, we must devise some means of exiting BASIC and returning to the TOS. I have shown how BASIC commands could be executed from machine language routines. But, how do we exit BASIC to our machine language routines? At first, it appears that ROM BASIC can only be exited with a BREAK or through a USR function call. This is true unless we can devise some means of patching into BASIC at some point and make BASIC think there is some new form of keyword present in the interpreter. Well, implementing new Keywords is not possible with ROM BASIC, so some other method must be devised.

An article which appeared in *Micro* described interception of BASIC Syntax error codes when printed on the monitor screen. A patch devised to intercept a Syntax error can be utilized to direct an exit from BASIC and force a return to a calling machine language program. The machine language patch routine shown in Listing 5 can be used to force an exit from BASIC during a running BASIC program, and in Immediate Mode or when a BASIC program has finished loading from cassette into the BASIC workspace. Listing 5 is a routine that has been revised for the purpose of exiting BASIC. The routine appeared in an article titled "Stop Those S' Errors" published in the November 1980 issue of Micro Magazine (*Micro*, 30:37).

The patch code for the BASIC exit routine utilizes a vector location in zero page. The vector is located at $03 and $04. Normally, this vector points to the string output routine of the BASIC interpreter at $A8C3. If we replace this jump with a call to our patch routine, we may use the pointer and our patch routine to exit BASIC on command. Listing 5, shows the Exit patch routine that is loaded into memory starting at $0240. To use the patch routine, replace the jump at $03 and $04 with the start of the exit patch routine. That is, load $40 into memory location $03 and $02 into location $04. This can be done in BASIC using the POKE command: POKE 3, 64 : POKE 4, 2. Once the address for the patch code has been loaded into the pointer at $03 and $04 the pointer will not have to be changed unless the computer has been reset.

Program 3.

```
0000          ;
0000          ;
0000          ;
0000          ;
0000          ;           BASIC LOAD COMMAND CALL
0000          ;
0000          ;
0000          ;
0000          ;
1100          *=$1100
1100 A9FF     START LDA #$FF    VALUE LOAD FLAG =ON
1102 8D0302   STA $0203     PUT IN LOAD FLAG
1105 4C74A2   JMP $A274     GOTO BASIC WARM START
```

Program 4.

```
0000          ;
0000          ;
0000          ;
0000          ;
0000          ;          ** BASIC RUN COMMAND CALL **
0000          ;
0000          ;
1150          *=$1150
1150          ;
1150 A952     LDA #$52        GET RUN TOKEN
1152 8513     STA $13        PUT IN LINE BUFFER
1154 A900     LDA #$00        NULL
1156 8514     STA $14        PUT BUFF+1
1158 8516     STA $16        PUT BUFF+3
115A 85C4     STA $C4        PUT PARSER HIGH BYTE
115C A94E     LDA #$4E
115E 8515     STA $15        PUT BUFFER+2
1160 A913     LDA #$13        GET PARSER START LOW
1162 85C3     STA $C3        PUT PARSER LOW
1164 20A6A3   JSR $A3A6         GO BASIC CONVERSION RTN.
1167 4CF6A5   JMP $A5F6         GO TO BASIC EXECUTION (RUN)
```

Program 5.

```
0000          ;
0000          ;
0000          ;
0000          ;
0000          ;
0000          ;        BASIC EXIT PATCH ROUTINE
0000          ;
0000          ;
00F0                *=0240
00F0          ;
00F0 48       PHA            SAVE PRT CHRACTER IN ACC.
00F1 AD65D3   LDA $D365         GET CHRACTER FROM SCREEN
00F4 C93F     CMP #$3F         TEST FOR ERROR(?)
00F6 D008     BNE OUT        NO NOT ERROR GO PRINT CHR.
00F8 A900     LDA #$00         YES ERROR GET READY TO EXIT
00FA 8D0302   STA $0203         RESET LOAD FLAG
00FD 4CFFFF   JMP $FFFF         RETURN TO CALLER($FFFF DUMMY
0100 68       OUT PLA           ADDRERESTORE CHARACTER TO ACC
0101 4CC3A8   JMP $A8C3         GO PRINT CHR. RETURN TO BASIC
```

The patch routine at $0240 tests memory location $D365 for a question mark ($3F) for each character printed out to the monitor screen. In the event of an error, such as ? Sn Error, the question mark will be loaded into video RAM at $D365. The routine tests $D365 for $3F. If there should be any type of error, the question mark code will appear at $D365. On detection of the error code, the patch routine will cause an exit to your machine language routine. Under normal program execution, the data to be printed is passed to the string printing routine of BASIC as if the patch routine did not exist.

The exit patch code routine was implemented into my TOS to detect an error at the end of a program loading from tape. My Stringy Floppy tape unit sends $8F when all the program on tape has been sent to the C1P. This hex byte, when seen by BASIC, will send back a Syntax error which will be detected by the patch routine causing an automatic exit to the TOS. While in BASIC, if the user types any key followed by a carriage return. It will cause a Syntax error and force a return to any calling routine. In addition, programming a line of illegal code at the exit point of the BASIC program will force a return to the calling machine language routine. An example line of illegal code could be: 10/ or 10 EXIT etc...

This article has presented some ways of implementing BASIC commands and calling these commands from machine language programs. Through these efforts, I have further expanded the ways in which we may use OSI BASIC and machine language programs as a means of system development. In my case, I have a TOS that functions like a disk operating system (DOS). With the information presented in this article, you may also be inspired to develop new programming techniques. Although this article was developed around OSI 6502 BASIC, the concepts should apply to other systems using similar BASIC such as, PET, and APPLE. Of course, tokens and interpreter routine addresses may need changing but the basic principles still apply.

References:

OSI BASIC In ROM, Edward H. Carlson
"Stop Those S' Errors," *Micro* Magazine, November 1980.

THE PET® GAZETTE

Practical Pet Printing Primer For Perplexed Programmers

Ron Gunn
Livermore, CA

When you first connect a printer to your PET, it is for one primary purpose: to obtain program listings so you can see enough code at one time to rescue yourself from the latest paradox in your programming. As eminently useful as this is, it soon becomes but a step in the utilization of the printer for the output of data from your programs in an organized and permanent form.

Organizing data on a printed sheet implies just that; organizing. You don't want it scattered randomly and capriciously all over the page. Columns with labels are often desired, and so you find that the convenient TAB functions for the PET screen don't work on the printer. It is then that you perhaps start to reduce your data to strings, add blanks, and then partition these longer strings with the LEFT$, RIGHT$, and MID$ functions to make neatly justified columns.

Universal Printing

We are assuming that you are interested in programming that will allow you to print on any printer that will give a listing, a PET, Epson, or whatever. This kind of coding will be the subject of this article. Many printers, including Commodore's, have proprietary ways of handling this problem and I say more about the PET printer's formatting later. These proprietary methods will normally work only on that one make of printer.

You will be able to take the kind of code we are discussing here over to Joe's house and it will work on his PET, Anadex or Kung-Fu papergobbler.

The Answer That Creates A Problem

Here is a sample of code that outputs 3 variables in neat columns on any printer that will LIST. It is intended to illustrate how the string technique mentioned in paragraph two works in practice. There is a problem with this, especially in PET ROM sets earlier than 4.0, and it is not recommended for large data bases or more than a page of print for reasons we will get into.

```
2200 FOR I=1 TO 30
2210P$=""
2220Q$="        "+STR$(K%(I))
2230P$=RIGHT$(Q$,8)
2240Q$="        "+STR$(L(I))
2250P$=P$+RIGHT$(Q$,8)
2260Q$="        "+STR$(M%(I))
2270P$=P$+RIGHT$(Q$,8)
2280PRINT#4,P$
2290NEXT
```

This code is reasonably concise and straightforward and will produce neat columns of numbers of varying length, all nicely right-justified. What, then, is the problem? It is a lot like watching a centipede on a treadmill. There is an awful lot going on, but not much progress. Concantenating strings and then printing them out is potentially very slow. As you increase the number of columns, the string handling becomes appreciably slower than the printer.

When you add to this the fact that enormous numbers of throwaway strings are created, and that pre-4.0 PETs can take many seconds to collect garbage, you have a serious problem. (See Butterfield, "Learning About Garbage Collection," **COMPUTE!** #10.) If you have not experienced this yet, try the experimental listing at the end of this article for an eye opener.

This code example is print test #1 in the sample listing, and 100 seconds and more were required on my 32N to run it. This is admittedly an extreme example, but points up the trouble you can get into with a program that is large for your machine when you get heavily into string handling.

A Partial Answer

As the referenced article discussed, you can reduce the garbage collection, and you can get the printer to print while the computer is computing, with the following changes. If you have existing print routines that are structured like the above, and which could use some speeding up, then the following substitute coding will help:

```
2300 FOR I=1 TO 30
2310Q$="        "+STR$(K%(I))
2320PRINT#4,RIGHT$(Q$,8);
2330Q$="        "+STR$(L(I))
2340PRINT#4,RIGHT$(Q$,8);
2350Q$="        "+STR$(M%(I))
2360PRINT#4,RIGHT$(Q$,8)
2370NEXT
```

WATCH

THIS

SPACE

WATCH

THIS

SPACE

The change, of course, is that an intermediate variable is not created and then concantenated. The variable is created as a string and is packed with blanks to the maximum needed length on the same line. It is then immediately printed using the appropriate string handling function to make it the correct number of spaces to fit the column you are creating.

You have saved in two ways. The forever concantenated P$ is not created; an advantage as pointed out by the redoubtable Mr. Butterfield. In addition, the printer can process each output while the next gets a head start.

This is print test #2 in the sample listing. Run times on my machine ran some 50 seconds, about half the total time required for test #1.

Columns Of Names

These examples have been right justified. If you need to print a column of names, you will want to left justify it. This is not necessarily because it is better, it is just that it will be considered a bug or a product of indifference if you do not. Sample coding to do this is:

```
2800P$=N$+"                    "
2810PRINT#4,LEFT$(P$,20)
```

In this case, the necessary blanks have been added after the name, and the resulting oversized string has been printed out starting with the left-most character using the LEFT$ function. Always add enough blanks to insure that the space is filled even if N$ turns out to be a null string. This insures that other columns will not be disturbed if that happens.

When you have strings to print, you have to print strings. You can avoid extra string creation as shown by tacking on the necessary blanks at once. Then print immediately using the LEFT$ function in the print statement.

Hints And Kinks

If you have a lot of fill blanks to add, it saves memory to create space variables as a substitue for the blanks-within-quotes as shown on line 2220. Create these handy variables when the program initializes:

```
6000S4$="    ":S5$="     ":S6$="      "
6010S7$="       ":S8$="        "
6020RETURN
```

Use a gosub to do this when the program runs. GOSUB6000 in the first line of a program would set up the space variables shown above for use later.

The eight blanks in a line like 2310 could then be added to each string as:

```
2310P$=S8$+STR$(K%(I))
```

Another point; you already know that the TAB function doesn't work on the printer. Well, the SPC function does. You may have wondered what the differences between these two seemingly similar functions are. This is one of those differences. We will see an example of use later.

WHAT #0$*! APPROACH WORKS?

That second word was copied verbatim from the cover of **COMPUTE!** #7. Anyhow, there is a way to get whole numbers into printed columns without having to cope with slow string handling. It may sound silly, but the way to do it is to keep it as data (numerical data that is). We will now examine the techniques that may allow you to avoid string renderings of data altogether. You will then put the data out as fast as the printer can handle it; all day if necessary and always in neat columns.

The Basic Idea

Let's say you want to print out a number that will range between 0 and 999 in a column eight spaces wide. Using the integer variable K%, we might write:

```
2900IFK%=0THENPRINT#4,S8$;:GOTO2950
2910IFK%<10THENPRINT#4,S5$K%;:GOTO2950
2910IFK%<100THENPRINT#4,S4$K%;:GOTO2950
2930 PRINT#4,S3$K%;
2950REM NEXT COLUMN
```

Here we have covered the full range from 0 to 999. If the number goes over 999 it will disturb the columnation unless another line of code is added to catch numbers less than 9999 to realign it. Only one of these print statements will actually execute. If you expect a lot of larger numbers then this routine would go faster with the larger numbers first, as fewer comparisons would take place before the correct line was found and executed. This code will place one column on the sheet. The next column would require a repeat of the whole routine, so coding this way could get long. It is fast, however, and it is effective and straightforward if only a few columns are needed.

In this example, zero is not printed, eight spaces are printed instead. A single digit, including zero if you want it to print, takes three spaces. The <10 or single digit line (line 2910) has 5 spaces placed first, then the sign space, the numeral itself, then a space after. If you want 0 to print, then put in REM at the start of line 2900 to neutralize it. (Don't eliminate it as it is a GOSUB target line).

Note that the carriage return suppress semi-colon is used on all of these lines. This leaves the printer poised on the start of the next column on the same line. A PRINT# statement must be added at the end of the line to go smoothly to the next line, as we will see in the next example. It is like bowling, where the tenth frame is handled differently to wrap things up.

The GOSUB Variable Width Column Maker

Now that the principle has been covered, let's go to some code that will produce an unlimited number of columns of varying width using a reusable subroutine for each column desired. This routine will compete in size with string handling code and is as fast as you'll want.

NEW! IBM COMPATIBLE 8" FLOPPY DISK for CBM/PET,* Aim, Sym

PEDISK II

LOW COST — HIGH PERFORMANCE FLOPPY DISK

5¼" MINI FLOPPY DISK SYSTEMS:

Model 540-1 Single Drive, Double Density (143K) $595.00
Model 540-2 Dual Drive, Double Density (286K) $895.00
Model 580-1 Single Drive, Quad Density (286K) $795.00
Model 580-2 Dual Drive, Quad Density (572K) $1,195.00

8" MINI FLOPPY DISK SYSTEMS:

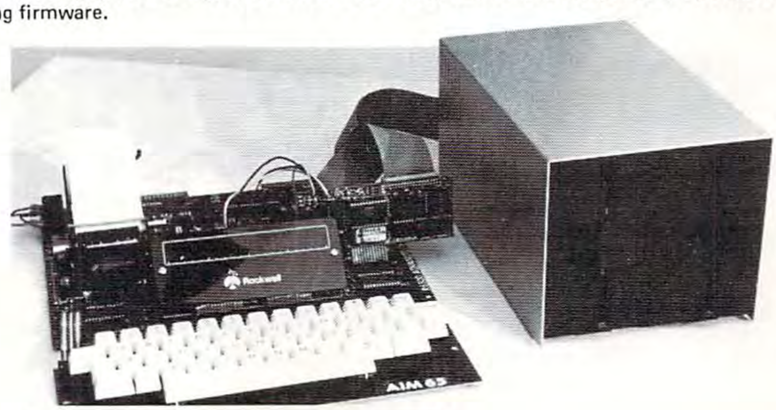
Model 877-1 Single Drive, IBM standard (295K) $1,095.00
Model 877-2 Dual Drive, IBM standard (590K) $1,695.00

Payable in U.S. Dollars

PEDISK II is a high performance mass storage peripheral to enhance your computer's storage capability. Total storage to 850K bytes is available. The PEDISK II system consists of a small disk controller electronic board that mounts inside the computer and an external disk drive assembly. PEDISK II offers the fastest mass storage system available for the Commodore PET. With a data transfer rate of 250000K bits per second, the PEDISK II loads data directly to memory. This is up to four times faster than any 488 bus-type mass storage device.

One, two, or three drives connect to the PEDISK controller board. The user can choose 5¼" or 8" disk drives. An important feature of the PEDISK II system is its ability to exchange data with other computers. The standard 8" IBM 3740 format allows data exchange with large computers. Data files, mail lists, etc. available on large computers can be processed by the PET/PEDISK system. Remote PET/PEDISK computers can provide data entry for large machines.

PDOS II software links directly to the standard BASIC and operates with BASIC-type commands for easy interfacing. A full set of disk utility commands completes the powerful disk operating firmware.



DISK COMMANDS:

BASIC

!LOAD — reads a program file to the computer
!SAVE — stores a BASIC program file to the disk
!OPEN — forms a sequential or relative data file
!INPUT — reads a data record from a file on the disk
!PRINT — stores a data record to a file on the disk
!CLOSE — ends a sequential or relative data file
!LIST — displays a directory of all files on the disk
!RUN — reads a program file and executes

MONITOR-DOS

D — displays contents of memory or diskette.
G — go to program and execute.
H — help user with listing of all commands.
K — kill a file on the diskette (erase file).
L — read program to the computer memory.
M — memory examine and change monitor.
N — name a file differently (rename).
P — print directory of all files on the disk.
R — return to BASIC mode.
S — save program or data from memory to the disk.
U — utility: format, copy, compress, patch diskette.
X — execute program after loading.

fullFORTH +

FULL FEATURE "FORTH" FOR 6502 SYSTEMS

INTERPRETER — can be executed directly in an interpretive mode to speed testing and debugging.

CROSS-COMPILER — words can be individually compiled and tested, the entire program can also be cross-compiled for maximum efficiency.

COND. ASSEMBLER — Machine language modules can be intermixed and conditionally assembled to fullFORTH.

STRING HANDLING — variable length constants and variables are allowed. Processes compare, move, concatenate and sub-string words.

FLOATING POINT — process 5 or 9 digit integer and floating point numbers for arithmetic operations.

SCREEN EDITOR — contains a unique full cursor visible screen editor.

SPECIFY PEDISK II, PET 2040 or 4040 DISK, OR APPLE $75.00

FOR INFORMATION, SEE YOUR DEALER OR:

cgrs MICROTECH

P.O. BOX 102 • LANGHORNE, PA 19047 • (215) 757-0284
U.S.A.

DEALER INQUIRIES INVITED

*PET IS A TRADEMARK OF COMMODORE

```
2400FORI=1 TO 30
2410P=K%(I):S=6:GOSUB2460
2420P=L(I):GOSUB2460
2430P=M%(I):S=8:GOSUB2460
2440PRINT#4
2450NEXT
2460IFP=0THENPRINT#4,SPC(S);:GOTO2510:
REM BLANK WHEN P=0
2470IFP<10THENPRINT#4,SPC(S-3)P;:GOTO2510
2480IFP<100THENPRINT#4,SPC(S-4)P;:GOTO2510
2490IFP<1000THENPRINT#4,SPC(S-5)P;:GOTO2510
2500PRINT#4,SPC(S-6)P;
2510RETURN
```

Any practical number of columns can be created by reusing the GOSUB at 2460 over and over. The width of each separate column is determined by the variable S, set at line 2410 for six spaces for K%(I) and L(I), and then set again for eight spaces for the variable M%(I).

When the last column has been printed out, the carriage return suppress is cleared with a final PRINT# statement on line 2440. Any additional columns would be added after line 2430 and before 2440, so the PRINT# statement would still be last.

This is print test #3. It prints in about 16 seconds with no pauses. My 80 cps printer is apparently running at its full speed through the whole test.

Check the data that you are printing to make sure that there is a printing line for each possible number that will be sent to the printing subroutine. Any number that is out of range will skew the columnation. You may have to exclude numbers that are too small or too large, or which contain negative or decimal quantities.

Lines, sometimes a number of them, may have to be carefully added to care for everything that you want to include in your printout. Remember the words of Gerald Weinberg in *The Psychology of Computer Programming*: "To detect errors, the programmer must have a conniving mind, one that delights in uncovering flaws where beauty and perfection were once thought to lie."

The Commodore Answer

I have not seen an in-depth review on the PET 2022 printer, but I have worked with a number of them. Commodore has provided a neat way to format printed information using a format file which you instruct as needed to produce columns, with many useful options. This works only with PET printers and then with reservations, as explained below.

There are two ROM sets for the 2022. The early set is noteworthy for the fact that it supplies a carriage return for every linefeed. When it pages or passes blank lines, the print head moves clear across for each line. That is the bad news. The good news is that the formatting works, and works well.

The newer ROM set will give linefeeds without the time-consuming full carriage scan. There is a bug, however, that causes the machine to go into permanent lower-case mode when more than a few columns are sent to the format file and then printed to. This is a fatal error, as nothing short of a power down will restore normal operation. Complex formatted output that prints perfectly on the older ROM set cannot be made to run on the new set.

I have not been able to locate a corrected ROM, although it has been on the horizon for a year. I have not yet had a chance to try the new PET/MX-80 printer.

Epson

The standard MX-80 was built for use with the TRS-80 and contains the Radio Shack character set to prove it. It does not have the same logic regarding carriage return/linefeed that the PET printer does. With the switch set for listings, KEYPRINT (**COMPUTE!** #7) will not work. Wordpro 3 required an added lf1 (linefeed) in every file header, and so on. Hoo Boy!

There is a switch that adds linefeeds, but it is buried inside the machine. I almost had my cover trained to jump off when I snapped my finger, when I finally added an external SPST switch in parallel with switch 2-3. When that SPST wears out I will have to replace it.

David Lein's late 1980 manual does a good job of explaining and demonstrating printer features. Appendix D on using the unit with other than the TRS-80 is very brief and touches on the Apple only.

The important features like double width letters, variable line spacing, and double strike work from the keyboard and under program control if you leave those pernicious JPET controls on the IEEE interface board OFF.

I hope that these samples and the discussion will help you to avoid some of the long hours otherwise required to find out how to efficiently use your printer. Some of these problems had me almost carrying my head around under my arm a few times. Remember Weinberg's words; "Any fool without the ability to share a laugh on himself will be unable to tolerate programming for long."

The Demonstration Program Listing.

```
1000 M=PEEK(51000):J=140
1450 IFM=0THENL=PEEK(135):POKE135,26:GOTO2000
1500 L=PEEK(53)
1510 POKE53,26
2000 DIMA$(J),B$(J),C$(J),D$(J),K%(30),L(30),M%(30)
2020 FORI=1TOJ
2030 A$(I)="TRY":B$(I)=A$(I)+"THIS":C$(I)="OUT":D$
     (I)=C$(I)+"TO SEE"
2040 NEXT
2050 FORI=1TO30
2060 K%(I)=RND(1)*9
2070 L(I)=INT((K%(I))↑2)
2080 M%(I)=(L(I))↑2
2090 NEXT
2100 OPEN4,4
2110 T1=TI
```

SM-KIT

for Commodore Computers

A Programming Productivity Tool

ONLY $40

A 4K ROM with both programming and disk handling aids.

The SM-KIT is a collection of machine language firmware programming and test aids for BASIC programmers. SM-KIT is a 4K ROM (twice the normal capacity) which you simply insert in a single ROM socket on any BASIC 4 CBM/PET—either 80 column or 40 column. Includes both programming aids and disk handling commands.

ERROR DETECTION: the SM-KIT automatically indicates the erroneous line and statement for any BASIC program error.
LINE NUMBERING: the SM-KIT automatically numbers BASIC statements until you turn the function off.
SCREEN OUTPUT: the commands FIND, DUMP, TRACE and DIRECTORY display on the CRT while you hold the RETURN key (display pauses when the key is released). Continuous output is selected with shift-lock.
OUTPUT CONTROL to DISK or PRINTER: in addition to displaying on the CRT, you can direct output to either disk or printer.
HARDCOPY: allows screen displays to be either printed or stored on disk.
FIND: searches all or any part of a program for text or command strings or variable names. Either exact search or wild card search supported.
RENUMBER: the SM-KIT can renumber all or any part of a program. The selective renumbering allows you to move blocks of code within your program.
VARIABLE DUMP: displays the contents of floating point, integer, and string variables (both simple and array). Can display all variables or any selected variables.
TRACE: SM-KIT can trace program execution either continuously or step by step starting with any line number. Selected program variables can be displayed while tracing.
DISK COMMANDS: as in DOS Support (Universal Wedge), the "shorthand" versions of disk commands may be used for displaying disk directory, initializing, copying, scratching files, load and run, etc.
LOAD: SM-KIT can load all or part of BASIC or machine language programs. It can append to a program in memory, overwrite any part of a program, load starting with any absolute memory location, and load without changing variable pointers.
MERGE: allows merging all or any part of a program on disk with a program in memory.
SAVE and VERIFY: SM-KIT provides one step program save and verification. It also allows you to save any part of a program, or any address range.

Developed by (and available in Europe from) SM Softwareverbund-Microcomputer GmbH, Scherbaumstrasse 29, 8000 Munchen 83, Germany

252 Bethlehem Pike
Colmar, PA 18915
215-822-7727 **A B Computers**

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

NEW PRODUCT

SWARM-100

For 24 Pin ROM Machines Only.

JUST PLUG IT IN

• No soldering • No messy wires

SOFTWARE SELECTABLE

1. Software select one of two operating systems. (BASIC 2.0 / BASIC 4.0)
2. Software select utility ROMs at conflicting addresses.

$125.00 (U.S.)
($150.00 Canadian)
Add $3.00 shipping to all points outside Canada.

Master Charge and VISA accepted.

BATTERIES INCLUDED

Village by the Grange
71 McCaul Street
Toronto, Ontario
Canada M5T 2X1
(416) 596-1405

```
2120 PRINT"PRINT TEST NUMBER (1, 2, OR 3)"
2130 INPUTB:ONBGOSUB2200,2300,2400
2135 T=(TI-T1)/60
2140 PRINT#4,T
2180 CLOSE4
2190 PRINT"THAT TOOK"T"SECONDS TO PRINT"
2192 PRINTFRE(0)
2193 IFM=0THENPOKE135,L:END
2195 POKE53,L:END
2200 FORI=1TO30
2210 P$=""
2220 Q$="          "+STR$(K%(I))
2230 P$=RIGHT$(Q$,8)
2240 Q$="          "+STR$(L(I))
2250 P$=P$+RIGHT$(Q$,8)
2260 Q$="          "+STR$(M%(I))
2270 P$=P$+RIGHT$(Q$,8)
2280 PRINT#4,P$
2290 NEXT:RETURN
2300 FORI=1TO30
2310 Q$="          "+STR$(K%(I))
2320 PRINT#4,RIGHT$(Q$,8);
2330 Q$="          "+STR$(L(I))
2340 PRINT#4,RIGHT$(Q$,8);
2350 Q$="          "+STR$(M%(I))
2360 PRINT#4,RIGHT$(Q$,8)
2370 NEXT:RETURN
2400 FORI=1TO30                                   13
2410 P=K%(I):S=6:GOSUB2460
2420 P=L(I):GOSUB2460
2430 P=M%(I):S=8:GOSUB2460
2440 PRINT#4
2450 NEXT:RETURN
2460 IFP=0THENPRINT#4,SPC(S);:GOTO2510:REM THIS
     LINE LEAVES BLANK SPACE ON 0
2470 IFP<10THENPRINT#4,SPC(S-3)P;:GOTO2510
2480 IFP<100THENPRINT#4,SPC(S-4)P;:GOTO2510
2490 IFP<1000THENPRINT#4,SPC(S-5)P;:GOTO2510
2500 PRINT#4,SPC(S-6)P;
2510 RETURN
READY.
```

©

Odds And Ends:

A Fat Forty Bug

Gordon Campbell, Willowdale, Ont.

Some machine language programs which work just fine in PETs with BASIC 4.0 and 9-inch screens will yield odd results on the new 12-inch, 40-column machines. Occasionally, it will appear as if the program has responded inaccurately to the key which was pressed.

This is due to a very subtle difference in the ROM. If the machine language program happens to be in decimal mode when an interrupt occurs, the keyboard decode is completely inaccurate. Even if a key is still pressed from a previous character, it will be decoded as a different character, so, instead of getting one character, you get three (the original correct one, one that is wrong, and probably the original one again).

The problem is well illustrated by the program listing attached. If you assemble the program as shown, then from the monitor type: G 0800 you can type on the keyboard and see what character the PET thought was pressed. By the way, to get out of the program, press the RVS key. Now change the SED (set decimal mode) to a NOP. The program gives accurate keyboard decode. Now the STOP key will get you out.

The way to get around this bug, if you must use decimal mode, is to precede the routine with an SEI (disable interrupts) instruction, and follow it with a CLI (enable interrupts).

```
                                  .BA $0800
                                  .OS
                ; TEST KEYBOARD DECODE
                ; WITH DECIMAL MODE SET!
                GET               .DE $FFE4
                PRINT             .DE $FFD2
                ;
0800- F8        STRT              SED
                ;
0801- CA        DLLOOP            DEX
0802- D0 FD                       BNE DLLOOP
0804- 88                          DEY
0805- D0 FA                       BNE DLLOOP
                ;
0807- D8                          CLD
0808- 20 E4 FF                    JSR GET
080B- C9 00                       CMP #0
080D- F0 F1                       BEQ STRT
080F- C9 03                       CMP #3
0811- F0 06                       BEQ STOP
0813- 20 D2 FF                    JSR PRINT
0816- 4C 00 08                    JMP STRT
0819- 00        STOP              BRK
                                  .EN

LABEL FILE:   [ / = EXTERNAL ]

/GET=FFE4           /PRINT=FFD2    STRT=0800
DLLOOP=0801         STOP=0819
//0000,081A,081A
```

©

Machine Language: What's Your Sign?

Jim Butterfield
Toronto, Canada

Beginning programmers learn very quickly that a memory location has eight bits capacity, so that it may hold a number from zero to 255, or 00 to FF in hexadecimal. That's the range of values that you are allowed to POKE and that you will see with a PEEK. These numbers all seem to be positive. Why, then, do some of them set the N (negative) flag when loaded? More generally — how do you handle signed numbers?

As Humpty Dumpty almost said: "When I use a number, it means just what I choose it to mean — neither more nor less". As programmers, we can choose to treat a memory location value as if it were unsigned, a number from 0 to 255; or signed, a number from -128 to +127. We can view the value in many other ways, too: as part of a bigger number, as an ASCII character, as two or more small numbers packed together, and so on. For the moment, let's concentrate on the signed number aspect.

Sign Posts

The N (negative) flag is tied to the most significant bit of the value under consideration. For example, if we load into a register a value of hexadecimal C8 which would PEEK as 200 decimal, the N flag is turned on. Why? If we write C8 in binary we get 11001000. The first bit (called the high-order bit or the most significant bit) is one, and that's what kicks the N flag on. If we mean the number to be unsigned, we may ignore the N flag; but if we mean the number to be signed, the N flag tells us that it is negative. We can tell what negative value is represented in an 8-bit location by subtracting its unsigned value from 256, so that C8 (unsigned 200) has a signed value of -(256-200) or -56.

This method of representing signed numbers is called twos-complement and it works well once you get used to it. There are a few special rules to keep in mind when you add, subtract, multiply, and compare, but most things are quite straightforward. You'll quickly learn that FF or decimal 255 has a signed value of -1; that the highest 8 bit signed value is hex 7F or +127 and the lowest is hex 80, or -128.

Overflow: My Byte Runneth Over...

When we add unsigned numbers, we need to watch for a leftover Carry (C flag) after the addition is complete. If the C flag is on, it means that the addition has generated a result that is too large to fit the space available. Similarly, when we subtract unsigned numbers, we look for the inverse: the C flag being off means that we have tried to subtract a bigger value from a smaller one — and that's illegal if we want unsigned results.

The rules are different when we add and subtract signed numbers. The problem we must look for here is a "sign switch": for example, adding +100 to +100, two positive numbers, will generate a value of 200 — which is a negative value if placed in a single byte. This type of error is called "overflow," and the 6502 conveniently provides us with an overflow indication (the V flag) to warn us of difficulty in signed addition or subtraction. A BVS (Branch Overflow Set) will detect the fault and allow us to code an appropriate error or warning routine.

Remember that both Carry and Overflow are set with each Add and Subtract command you execute. It's up to you to choose which flags are important: you know which numbers are signed and which are not.

Multiplication: Sign Of The Times?

General multiplication of signed numbers calls for careful testing of both signs and quite a bit of work. For the moment, we'll concentrate on simple multiplication routines: multiplying by a fixed value of say four or five.

We multiply a number by two by using an ASL command. If we were doubling an unsigned number, we once again test the C flag to make sure that the new value fits into the space provided. For signed numbers, it's a little more work: we must make sure that doubling the number hasn't caused the sign to change. The overflow flag won't help us here (I wish it did) since it is unaffected by Rotates and Shifts. The usual coding method is to check that the C flag, which holds the previous sign, matches the N flag, which holds the current sign.

To multiply by four, we use two ASL commands, and we must carefully check for errors after each one. If we wish to multiply by five, we multiply by four and then add the original value — hopefully stored somewhere — and make the final overflow check on the addition.

Comparison: Getting the High Sign...

Comparing signed numbers can't be done with a single flag or a single test. The C flag gives you a valid comparison if the two signs are the same, but not if they differ. You could pre-check the signs: for example LDA VALUE1 : EOR VALUE2 would

turn the N flag off if the signs were the same, or on if they were different. If the signs were identical, a normal Compare would settle the matter; if not, the positive number must be the biggest one.

Frank Covitz offers the following elegant signed compare:

```
TEST1:  SEC
        LDA  N1      ;Get 1st number
        SBC  N2      ;Compare vs. 2nd number
        BVC  TEST2   ;Branch on overflow clear
        BIT  N1      ;Else test sign of 1st number
TEST2:  BPL  GTE     ;If plus, branch to Greater Than
                          or Equal
        ..           (code for less than)
        ..
GTE:    ..           (code for greater than or equal)
```

Note that two tests are performed: the SBC (which is used to set flags rather than calculate results) and the BIT test. The N flag is brought into play here — it's unusual to see it doing a useful job in a comparison situation.

Odd Signs

When signs are used as part of large numbers, the sign bit appears only in the most significant byte. So if you allocate 32 bits (four bytes) to a value, only one — the highest order — gives the sign.

It's possible to sign decimal numbers. If the numbers are held in BCD for decimal addition and subtraction, the sign works out rather oddly. The high bit is still the most convenient to use — but this causes positive values to be those starting with the digits 0 through 7, and negative values to begin with digits 8 and 9 only. This "unbalanced" arrangement of numbers is often satisfactory and allows the N and V flags to perform their proper roles. If, on the other hand, you need to balance the range of positive and negative decimal values, you'll want positive numbers to start with digits 0 through 4 and negative values with 5 through 9. In this case, you have to do most of your own sign work. As a last resort, you can keep the sign as a completely separate flag — but beware of additions and subtractions that cross the positive-negative boundary.

Signing Off...

Most machine language work is in unsigned integers: you'll need to deal with signed numbers only rarely. But when you do, it's essential to know how to handle them ... you might say that it's one of the signs of good programming. ©

BackPack

BATTERY BACKUP SYSTEM

FOR COMMODORE PET/CBM COMPUTERS

Standard Features:

- Full power to PET/CBM for a minimum of 15 minutes
- Installs within PET/CBM cabinet
- No wiring changes necessary
- Batteries recharged from PET/CBM integral power supply

Specifications:

- **Physical Size:** 5.5″ x 3.6″ x 2.4″
- **Weight:** 4.5 lbs.
- **Time to reach full charge:** 16 hours
- **Duration of outputs:** Minimum of 15 min.
- **Voltages:** +16, +9, −12, −9
- **Battery Life Expectancy:** 3 to 5 years
- **Battery On-Off Switch**

For Use With:

- Commodore PET/CBM 2001 and 4000 series computer
- Commodore PET/CBM 8000 series computer (screen size will not be normal on battery back-up)
- Commodore C2N Cassette Drive

Never again lose valuable data because of power shortages or line surges. **BackPack** supplies a minimum of 15 minutes reserve power to 32K of memory, the video screen and tape drive. **BackPack** fits inside the PET/CBM cabinet and can be installed easily by even the novice user. **BackPack** is recharged during normal operation and has an integral on-off switch.

BackPack comes fully assembled and tested. Instructions included.

BackPack is a trademark of ETC Corporation
CBM/PET are trademarks of Commodore Business Machines

Designed and manufactured by:

ELECTRONIC TECHNOLOGY CORPORATION
P.O. Box G, Old N.C. 42
Apex, North Carolina 27502
Phone: (919)362-4200 or (919)362-5671

Electronic Manufacturing
Technical Design and Development
Computer System Technology

ELECTRONIC TECHNOLOGY CORPORATION

Train Your Pet To Run VIC Programs

Lyle Jordan
Maple Grove, MN

Have you already wished for the capability to renumber your VIC program, or to get a print-out, or to save it on a disk?

This could be especially frustrating if you have a PET, a printer, and a disk sitting idly nearby.

Or how about satisfying the desire to "upload" your VIC program into a PET? This article will give you a couple of quick and easy ways to do just that.

The PET BASIC programs start to occupy memory at location 1025 decimal or $0401 hex. For the VIC, programs will start at 4097 decimal or $1001 hex. To make things compatible, start by putting a one line program into your PET (example: 1 REM). Now load your VIC program by typing "load". The VIC program will load just fine, but will have a starting location of $1001 hex and if you do a LIST, it won't show up at all. You will see only your one line program, 1 REM.

To get to the VIC program, you will need to change the forward linking pointers. This can be accomplished by doing a SYS 54386 (to get into the machine language monitor) and then by changing two memory locations.

First look at memory locations $0400 to $0407, by typing:

```
M 0400,0407
```

The PET will display the following:

```
.M 0400,0407
.: 0400 00 07 04 01 00 8F 00 00
```

Next list memory $1000 to $1007 and see:

```
.M 1000,1007
.: 1000 AA 18 10 0A 00 99 22 56
```

This display will vary depending on the first line of your VIC program. My first line was 10 PRINT "VIC-20".

Now you can change the '07' and the '04' at locations $0401 and $0402. You want this to point to the location of the first forward pointer of the VIC program, so the '07' becomes '01' and the '04' is changed to '10'. Make the changes, press RETURN, and cursor down to the last line displayed, type 'x', and then press return again. When the PET gives the "READY", you are back in BASIC and can do a LIST. What appears is the one line, 1 REM, followed by the VIC program.

Having served its purpose, line one can now be removed, and the VIC program will be copied into the normal start of PET BASIC at location 1025 decimal or $0401 hex.

If you have The BASIC Programmer's Toolkit from Palo Alto IC's, this entire procedure can be replaced by simply activating the Toolkit, and typing "APPEND".

I will have a lot of use for both of these procedures. Some that come to mind immediately are such things as getting a VIC program listing on a PET printer, renumbering a VIC program, and compacting a program so as to make the best possible use of the VIC's 3.5K of memory. I hope that this simple procedure will prove useful to others. ©

PETTERM™

Terminal Program

Intelligent Communication

Uploads and Downloads

In Machine Language

Use your PET as an intelligent terminal. Access timeshare systems and networks such as the Source or Telenet. This is an interrupt driven system with buffered input/output. Unlike dumb terminals, the PET can do other tasks instead of having to wait for incoming data. This flexibility allows many advanced features.

These are complete assembled hardware and software packages. All include line editing/resend, auto-repeat, shift lock, output to IEEE printer and much more. Delivered on cassette or disk. Also, available for Commodore 8010 modem or Livermore Star 488 modem.

PETTERM I All features above .... **$ 80.00**
PETTERM II All features of I. plus local text editor with down-loading capability **90.00**
PETTERM III All features of II. plus 80 132 column scrolling window for viewing formatted outputs wider than 40 columns **100.00**

FORTH

A Programming Revolution

FORTH Interactive high level compiler and operating system 5–10 times faster than PET BASIC. High level block structured language. This is a true fig-FORTH implementation for 16 and 32K PETs.

fig-FORTH, with editor, assembler $50.00
with floating point and strings $60.00
with floating point, strings, macro-assembler .. $70.00
cassette and disk interface available.

DEALERS INQUIRE

F S S

software for small computers

1903 Rio Grande
Austin, Texas
78705

1-512-477-2207

P.O. Box 8403
Austin, Texas
78712

Source: TCE995

Converting To Fat-40

Joe Ferrari
Commodore Canada

The addition of some new features to the 40 column PET has brought about some problems with program compatibility between the 4032 ten inch and 4032 twelve inch CRT display machines. In some cases, the program changes required for proper operation on the "Fat Forty" may be trivial and in other cases the conversion may be next to impossible. I will attempt to cover areas where failure can occur and what changes need implementing.

Level 1
Programs Loading Below BASIC ($0400)

Standard BASIC programs should work without any modification, unless they employ PEEKs and POKEs or if the program loads into memory below BASIC. The latter problem can be a bit tricky to spot unless you know specifically what to look for. If the program does load below BASIC (say $033A), but does not use locations $03E9-03f9, one method that will correct the problem is:

1) get the program into memory (don't execute)
2) SYS to the monitor (SYS 4)
3) display hex $03E9-03F9
4) modify these registers with the following values

[$03E9: 10,10,09,10,00,00,00,00,00,00,00,00,00,00,00, 00,00]

5) resave the program via the monitor

Tape Unit #2

Another area where the standard BASIC program can fail is in the utilization of the second cassette unit for sequential file access. If any program calls files from tape unit #2, unpredictable effects can result depending on the data coming into the buffer. In this case, nothing can be done to resolve the problem. The 12 inch 4032 has rendered the operation of the second tape unit virtually useless.

PEEKs And POKEs

Decimal location 151, often used to check if a particular key has been pressed, is still the same on the 12 inch, but the value of the keys has changed and therefore expected values for certain keys will return false information. The following table will assist in the conversion of a program with this problem.

KEY	OLDV	NEWV	KEY	OLDV	NEWV
@	15	64	A	48	65
C	31	67	D	47	68
F	39	70	G	46	71
I	53	73	J	45	74
L	44	76	M	29	77
O	60	79	P	52	80
R	55	82	S	40	83
U	61	85	V	23	86
X	24	88	Y	54	89
1	26	49	2	18	50
4	42	52	5	34	53
7	58	55	8	50	56
0	10	48			

KEY	OLDV	NEWV
B	30	66
E	63	69
H	38	72
K	37	75
N	22	78
Q	64	81
T	62	84
W	56	87
Z	32	90
3	25	51
6	41	54
9	57	57

When POKEs to this problem area are used for saving byte variables (whatever purpose), they must be moved to a free spot elsewhere in memory. If a space is free just below $03E9, this could be a good area for relocating the byte variables.

Level 2
BASIC Programs With Machine Language Utilities

BASIC programs using machine language utilities that reside in the second cassette buffer can work properly provided they don't use the taboo area of the buffer (namely decimal 1001-1017). Again, if the utility uses this area, the space must be relinquished to the PET operating system in order to obtain successful execution of the program. Usually, in the case of small machine language utilities, it shouldn't be too difficult to understand and relocate it to an area of memory that is free.

Level 3
Machine Language Programs

This will be the most difficult area to troubleshoot. If you are going to attempt to modify this type of program, be prepared to spend a good deal of time. Making the necessary changes to get the program working will most likely require a considerable amount of effort — which I personally don't recommend.

If you are really desperate, here are a few helpful hints that may assist you:

CBM/PET INTERFACES

RS-232 SERIAL PRINTER INTERFACE – addressable – baud rates to 9600 – switch selectable upper/lower, lower/upper case – works with WORDPRO, BASIC and other software – includes case and power supply.

MODEL – ADA1450 149.00

CENTRONICS/NEC PARALLEL INTERFACE - addressable – high speed – switch selectable upper/lower, lower/upper case – works with WORDPRO, BASIC and other software – has Centronics 36 pin ribbon connector at end of cable.

MODEL – ADA1600 129.00

CENTRONICS 730/737 PARALLEL INTERFACE – as above but with Centronics card edge connector at end of cable.

MODEL – ADA730 129.00

COMMUNICATIONS INTERFACE WITH SERIAL AND PARALLEL PORTS – addressable – software driven – true ASCII conversion – selectable reversal of upper-lower case – baud rates to 9600 – half or full duplex – X-ON, X-OFF – selectable carriage return delay – 32 character buffer – centronics compatible — much more.

MODEL – SADI 295.00

ANALOG TO DIGITAL CONVERTER – 16 channels – 0 to 5.12 volt input voltage range – resolution is 20 millivolts per count – conversion time is less than 100 microseconds per channel.

MODEL – PETSET1 295.00

REMOTE CONTROLLER WITH CLOCK/CALENDAR – controls up to 256 devices using the BSR X10 remote control receivers – 8 digital inputs, TTL levels or switch closure — 8 digital outputs, TTL levels.

MODEL – PETSET2 295.00

All prices are in US dollars for 120VAC.
Prices on 220 VAC slightly higher.
Allow $5.00 shipping & handling, foreign orders add 10% for AIR postage.
Connecticut residents add 7½% sales tax.
All prices and specifications subject to change without notice.
Our 30 day money back trial period applies.
MASTER CHARGE/VISA accepted.

MENTION THIS MAGAZINE WITH YOUR ORDER AND DEDUCT 5% FROM TOTAL.

IN CANADA order from: Batteries Included, Ltd., 71 McCaul Street, F6 Toronto, Canada M5T2X1, (416)596-1405.
IN THE USA order from your local dealer or direct: Connecticut microComputer, Inc., 34 Del Mar Drive, Brookfield, CT 06804, (203)775-4595.

Dealer inquiries invited.

CmC

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

Every PET Needs a Friend.

CURSOR is the best friend your Commodore PET will ever have. Since July, 1978 we have published 150 of the most user-friendly programs for the PET available anywhere. When we write or edit a program, we spend lots of time fussing about how it will treat you. We pay attention to lots of little things that help make using a computer a pleasure instead of a pain.

Naturally, **CURSOR** programs are technically excellent. Each program that we purchase is extensively edited or rewritten by a professional programmer. But *imagination* is just as important as being user-friendly and technically good! We delight in bringing you off-beat, unusual programs that "show off" the abilities of your PET or CBM.

CURSOR is user-friendly, technically great and full of imaginative programs. And every issue of **CURSOR** is still available! We continue to upgrade previously published programs so that they'll work on the three varieties of Commodore ROM's (Old, New, and 4.0). New issues also work on the 80 column CBM.

For only $4.95 you can buy a sample issue and judge for yourself. Or send $18 for a four-issue subscription. Each **CURSOR** comes to you as a C-30 cassette with five programs and a graphic Front Cover, ready to LOAD and RUN on your PET.

Who knows? After your PET meets **CURSOR**, things may never be the same!

AUTHORIZED DISTRIBUTORS:

Great Britain
AUDIOGENIC, Ltd.
P.O. Box 88
Reading, Berkshire

Holland
COPYTRONICS
Bergemeester
Van Suchtelenstraat 46
7413 XP Deventer

Japan
SYSTEMS FORMULATE CORP.
Shin-Makicho Bldg. 1-8-17
Yaesu, Chuo-ku, Tokyo 103

Australasia
MICROCOMPUTER HOUSE, LTD.
133 Regent Street
Chippendale, Sydney
N.S.W. Australia 2008

THE CODE WORKS
Box 550
Goleta, CA 93116
805-683-1585

1) Use Supermon or Extramon to locate any absolute occurrences of memory addresses from \$03E9-\$03f9 and reassign new values.

2) Check hi-low tables for references to the same address locations and, if any, reassign new values.

3) Seek all immediate operations involving hex \$03 and \$e9-f9...if any, look at code where occurrence takes place and evaluate.

4) Check all JSR & JMP occurrences into the E block ROM. All other ROMs can be ignored since they are identical.

Factory CRT Setup (Size)

One other problem that may or may not be encountered is screen setup. If the user decides on entering screen text mode via "PRINT CHR\$(14)", the top line of screen may run off the upper edge and not be visible. To restore screen to normal graphics mode enter "PRINT CHR\$(142)". One easy solution to this problem is to use "POKE 59468,14". This will turn the PET to text mode without opening up pixel lines between text.

The changes required to existing software may be a problem now, but, at the same time, these changes bring the 4032 to a closer compatibility with the 8032 model. Features such as repeat keys, scroll up and down, bell, and more are available. ©

High Resolution Bar Graphs For The Pet

David C. Swaim
Atlanta, GA

One of the reasons I chose the PET when I bought my computer was its excellent graphics capabilities. The PET/CBM graphics character set can be used to make "double density" or "high resolution" graphs without resorting to additional hardware. The highest resolution can be achieved when using bar graphs. Bar graphs can be found regularly in the financial section of the newspaper and are used to illustrate everything from the yearly rise in prices to the yearly production of wheat. You can draw bar graphs on your PET easily. What I want to describe in this article is a method I use.

How It Was Done

The first program I wrote to draw a bar graph used the reversed space to draw the bar. These bars looked fine but there were only about 20 steps vertically or 35 horizontally (allowing some space for labels). I wanted higher resolution than that for my graphs. It is possible to increase the resolution by a factor of eight either horizontally or vertically. All I had to do was use the other characters needed along with my reversed spaces. I put the needed graphics characters in an array. First I needed to plot the proper number of whole spaces for the bar. This was done by dividing the value to be plotted by the value I chose to be the maximum allowable value (the value that would give me a complete solid bar). This gave me a fraction that I multiplied by 20 (my vertical bar is 20 lines tall) and the result was truncated to an integer value, i.e., the fractional part was dropped (in my "low resolution" program the result was rounded to the nearest integer). This is an important difference. This integer corresponded to the number of times I would print a reversed space. The graphics character to be used for the last space was found by taking the fractional part from the original calculation of length and multiplying it by eight. This number was rounded to the nearest integer value and used as the index of my character array. Thus the proper character was printed to finish the bar.

The Program

The program listed here is a general purpose bar graphing program illustrating the high resolution bars. The bar graph is loaded into an array first before it is plotted on the screen. This is done for flexibility. To switch from vertical to horizontal graphs requires rewriting only the print routine (lines 1000 and up) and putting the appropriate graphics characters in the array X$ (lines 110 and 115). The rest of the program remains the same. You can change the data plotted by retyping the DATA statement (line 150) and changing the heading in K$ (line 160). You do not need to worry about the maximum value of the data. The program calculates (on line 280) a maximum bar length so that the largest data value is plotted nine-tenths of the maximum allowable bar length. If you prefer line 280 can be changed to set a definite ceiling. Be careful though. The program is not protected against data larger than the maximum bar length.

Figure 1 shows the bar graph displayed on the screen by the program. This particular graph shows the seasonal or monthly variation of my natural gas bills last year. During September and October my securing deposit was refunded as a credit on my bills so I paid no bills for those months. I put a dollar in September and a dollar and a half in October to illustrate the high resolution available with this program. This is a resolution of about half of a percent (maximum bar length would represent around $100). Figure 2 illustrates the same data printed with a horizontal format. Try it and I'm sure you'll never be satisfied with "low resolution" graphs again.

```
10 REM ************************
60 REM ************************
100 DIM M$(11,20),Y(12),X$(8)
105 REM X$ CONTAINS GRAPHICS CHARACTERS
106 REM FOR MAKING THE BAR
110 X$(1)="$":X$(2)="o":X$(3)="y":X$(
    4)="b":X$(5)="{REV}x{OFF}":X$(6)=
    "{REV}w{OFF}"
115 X$(7)="{REV}c{OFF}":X$(8)="{REV}{OFF}"
119 REM MOVE CURSOR DOWN CHARACTERS
120 P$="{24 DOWN}"
125 REM HOME AND UP, BACK CHARACTERS
130 H$="{HOME}":U$="{UP}{LEFT}"
140 REM DATA TO BE PLOTTED ON THE BAR
150 DATA 75.36,91.53,61.29,39.56,21.7
    8,11.4,11.39,10,1,1.5,24.69,35.67
155 REM K$ IS THE TITLE OF THE GRAPH
160 K$="NATURAL GAS BILLS"
190 REM READ THE VALUES TO BE PLOTTED
200 FOR X=1 TO 12
210 READ Y(X)
220 NEXT X
230 Y(0)=Y(1)
250 FOR X=2 TO 12
260 IF Y(X)>Y(0) THEN Y(0)=Y(X)
270 NEXT X
275 REM SET MAXIMUM BAR LENGTH, Y(0),
276 REM TO BE 10/9 TIMES LARGEST VALUE
280 Y(0)=10*Y(0)/9
300 FOR M=1 TO 12
301 N=M-1
305 REM CALCULATE THE NUMBER OF PRINT
306 REM LINES LONG THE BAR  WILL BE
310 Z=Y(M)*20/Y(0)
315 REM Z% IS NUMBER OF WHOLE LINES
316 REM TALL THEN BAR IS
320 Z%=INT(Z)
325 REM R% IS NUMBER OF RASTER LINES
326 REM TO BE PRINTED ON THE Z%+1
```

```
330 R%=INT((Z-Z%)*8)
336 IF Z%=0 THEN 530
500 FOR P=1 TO Z%
510 M$(N,P)=X$(8)
520 NEXT P
530 IF R%=0 THEN 550
540 M$(N,Z%+1)=X$(R%)
550 NEXT M
1000 REM PRINT THE BAR GRAPH
1004 PRINT"{CLEAR}"
1005 REM WRITE THE GRAPH TITLE
1006 PRINT SPC(20-LEN(K$)/2);K$
1009 REM PUT BAR IDENT. AT BOTTOME OF SCREEN
1010 PRINT "{HOME}";P$;"   J  F  M  A~
      M  J  J  A  S  O  N  D";
1020 FOR M=1 TO 12
1021 N=M-1
1025 REM POSITION CURSOR READY TO DRAW
1026 REM THE NEXT BAR
1030 PRINT H$;LEFT$(P$,21);SPC(3*M);
1035 REM DRAW THE ENTIRE BAR INCLUDING
     BLANKS
1040 FOR B=1 TO 20
1050 PRINT M$(N,B);U$;
1060 NEXT B
1070 NEXT M
1080 GETA$:IFA$=""THEN1080
1100 END
READY.
```

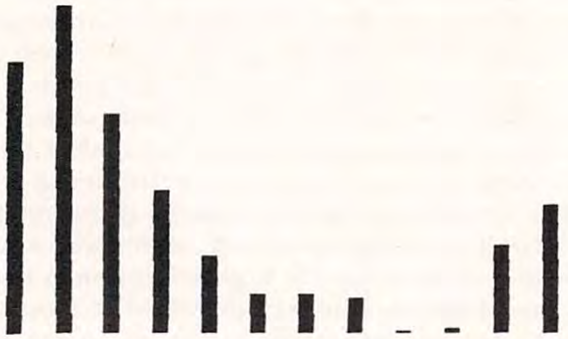


Figure 1: Natural gas bills are plotted to illustrate vertical bars. Note the fine resolution between September and October.

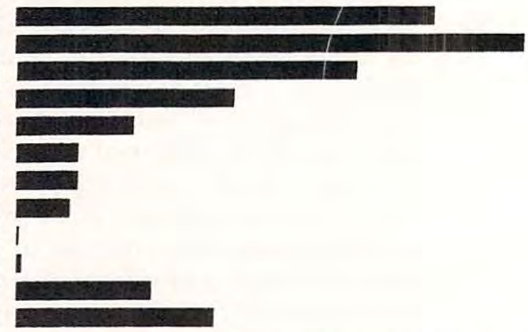


Figure 2: The same data as in Figure 1 is plotted horizontally. The only program changes made were to lines 110 and 115 and the print routine beginning on line 1000. ©

PIE-C

PET/CBM* IEEE-488 TO PARALLEL PRINTERS By LemData Products

P.I.E.-C MEANS—Professional design, Indispensible features, Excellent quality and Cost effectiveness. You can't buy a better parallel interface for your PET/CBM.

Our P.I.E.-C will interface your PET/CBM through the IEEE-488 bus to . . . . . .

the NEC Spinwriter, the C. Itoh Starwriter, printers by Centronics, Epson, Anadex, Escon Products, the Paper Tigers by IDS, the MIPLOT by Watanabe, the DIP printers, the AJ-841, the OKIDATA printers, plus ALL OTHER parallel ASCII printers.

Assembled with custom case, CBM-TO-ASCII code converter and appropriate cable, the **P.I.E.-C** is only $119.95 (+ $5 S&H). Md. Res. +5% tax. Specify printer and CBM models.

LemData Products, P.O. Box 1080, Columbia, Md. 21044 Phone (301) 730-3257
*PET/CBM are trademarks of Commodore Business Machines

IMMEDIATE DELIVERY

EDUCATORS....

CAN YOUR SOFTWARE PASS THIS TEST?

DOES IT OFFER:

1. PROGRAMMABILITY via Educator Interactivity for ALL Subjects?
2. MULTIPLE CHOICE, FILL-IN or TRUE & FALSE at Educator's request?
3. STACKED or SIDE-BY-SIDE answers at Educator's request?
4. Choice of displaying incorrectly answered Questions again?
5. Choice of displaying correct answer if students reply is wrong?
6. AUTOMATIC creation of Data Statements into program from Screen?
7. FREE MEMORY status on Screen as tests are created?
8. Auto-Erase of Data Creation portion to conserve memory?
9. Capability of SAVING created tests to Disk or Cassette for later use?
10. Individual Student Summary to Screen or Printer of Date, Name & Score?
11. Automatic printer commands, no opening or closing ports manually?
12. Incorrect Entry protection? A RETURN alone defaults to a WRONG answer?
13. Updated progress report to student as test is running?
14. Upward compatibility with ALL ROM SETS including latest 4.0?
15. Two safeguards to Insure program Integrity? Teachers own personal SECURITY ACCESS CODE, changeable at each test-session? Protection from STUDENT STOPPING or LISTING program?
16. Copies for BOTH 40 & 80 Character Screens? Operation in 8 to 32K?
17. Full-featured MEMORY TEST to allow you to check your PET/CBM out?
18. BACK-UP copies provided FREE with order? White POLY Notebook with 30+ Pages of Documentation & Program Listings?

OUR SCHOOLPAC-1 PACKAGE DOES ALL THIS AND MORE!

CBM/PET?

SEND $1 FOR CATALOG AND $5 OR $10 OFF OF YOUR NEXT ORDER!

SCHOOL PAC-1 **Disc or Tape—ONLY $34.95**

ADD $2 FOR S/H to ALL ORDERS! WE ACCEPT VISA & MC Orders!

COMPETITIVE SOFTWARE
21650 Maple Glen Drive
Edwardsburg, MI 49112

Waking Up The PET Screen

Hal Bredbenner
Raleigh, NC

An active screen display is always an asset to any business or recreational program and there are many ways to animate or otherwise liven-up a drab display of information. Most active screens make use of special graphic characters, machine code routines, or special hardware functions such as a character blink mode generated by a video processor. This article describes a software routine that provides reverse field blinking anywhere on the screen at variable blink rates. The routine uses BASIC only and no special hardware is required. The PET jiffy counter, TI, is referred to by the program, however its contents are not modified. The TI jiffy counter is standard in the PET (all versions, all ROMs) and if another system is used, any Real Time Clock (RTC) can be used as a reference.

The Blinking GET Routine, shown in Listing 1, is useful for prompting the system operator during program execution. When called, the subroutine will display, at the bottom center of the screen, the words "HIT ANY KEY." The displayed words will then blink from normal to reverse video at a rate programmed in line 160. Each jiffy in the PET is 1/60th of a second and since line 160 waits for 15 jiffies the display will blink every quarter of a second. If you don't yet understand how this timing works, I will decribe it further in the next paragraph. The subroutine continues blinking the words until a key is depressed, at which time the routine returns to the main calling program with string variable S$ equal to the input key. The routine is very straightforward in its operation and, although it is not very classical in its construction, it is efficient and can be easily understood and adapted for use in other programs.

This paragraph gives a line-by-line description of the actual operation of the routine and, if you have advanced in your BASIC programming skills, perhaps you will find it a little boring. When you are just getting started though, a line-by-line description can really be a help. Line 120 places the cursor at the bottom of the screen, however this could be modified with the cursor movement characters to place the cursor at any screen location. X$, in lines 130 and 140, is used as a switch to alternately hold the REVERSE FIELD ON or REVERSE FIELD OFF character. Initially X$ holds the REVERSE FIELD ON character, line 130, and a jump to line 150 is made. After the cursor is moved over thirteen spaces by the TAB function, the reverse field is either turned on or off by printing X$. Since X$ initially holds the REVERSE FIELD ON character the reverse field function is turned on and then the words "HIT ANY KEY" are printed. A cursor up character is also printed to leave the cursor on the line just printed. The last task done in line 150 is to set variable H equal to the current value of TI, the jiffy counter. In line 160 a comparison is made to see if 15 jiffies have elapsed since the words were printed on the screen. This is done by comparing the number of jiffies in TI at the time the words were printed, which we called H, to the present value of TI. Any time a program refers to TI, the current value is returned and this is what makes a realtime clock a very nice feature. If the time elapsed is less than 15 jiffies then line 170 checks to see if a key is depressed. If any key is pressed its value is assigned to the string variable S$ and a return to the calling program is made. If no key is pressed line 180 redirects the routine back to line 160. This loop continually checks the elapsed time and the state of the keyboard. After 15 jiffies have elapsed line 160 will redirect the program to line 190. If X$ is equal to the REVERSE FIELD ON character a branch is made to line 140 that changes X$ to the REVERSE FIELD OFF character. If X$ was equal to the REVERSE FIELD OFF character then it alternates to the REVERSE FIELD ON character by jumping to line 130 from line 200. After this, the words "HIT ANY KEY" are printed again with the REVERSE FIELD being opposite from the previous time, causing the words to blink. The program continues in this loop until a key is depressed and a return is made through line 170.

Listing 2 is a simple demonstration program using the blink routine in a slightly different manner. The three Blinking Message Subroutines are called by a main program that is printing the current time at the top of the screen. Each subroutine checks to see if it is time for it to blink and, if so, it reverses the field of its message. If the blink time has not been reached, then the subroutine immediately returns to the main program. Notice that the GET statements have been dropped and that each routine must reposition the cursor before printing its message. Each subroutine prints a different message in a different screen location and blinks at a different rate in this DEMO program. Notice also that the blink rate (25 in line 270) could be made a variable that could be modified elsewhere in a program to increase or decrease the blink rate of certain messages. A very large number could be used to turn the blink function

Let Us...

flick your VIC

Vic 20 memory expansion — you get 8k and 176x176 graphics for only $59.95

12VDC Power Converter — use the Vic in your car, camper, etc. $49.95

Brass Pounder — learn morse code the new and exciting way! $5.95

Music Machine — play songs from sheet music using the Vic's sound. $5.95

Vic Monitor — lets you do machine code programming on the Vic! $6.95

Talking Vic — lets the Vic speak the numbers one to ten! $8.95

Joysticks ....$24.95, **Schematics** $7.95

* *

OSI Rip Off 4 $5.95, Battle Zone $5.95, Adventure Game $9.00

TRS-80 Robot Simulator program $7.95

Add $2.50 per order to cover shipping and handling.

micro systems by
PETTED
hardware with a soft touch

P.O. Box 21851,
Milwaukee, Wisconsin 53221

DataMax

For CBM 8032 Computer-8050 Drive

3049 Cleveland Ave., Fort Myers, FL 33901

MAILMAN — Mail List Management System
by: Randy Stack — DataMax Software

Options available:

1. Enter a new file
2. Modify an existing file
3. Display an existing file
4. Delete a file
5. List entries by category
6. Generate mailing labels
7. Initialize a new data disk
8. Exit program

Enter your choice.

File Name - "Mail List Data"

Drive 0 - 1200 per file
Max 7,000 using both drives

S. Sort entries alphabetically
W. Create **WordPro** file
Z. Labels for a range of zip codes

This program will accept 5 or 9 digit zip codes.

- Links to **WordPro 4** word processor for printing form letters, billings, statements, etc.
- Our program disk may be removed after loading for extra storage.
- Unlimited files on multiple disks with up to 1200 records per file.
- Seven individual fields of information including: name, company name, address, phone, code and 25 character comment field.
- Sorts entries alphabetically for easy modification, i.e. to delete duplicate names, etc.
- Prints reports or mailing labels by any field or in order of zip codes for mass mailings.
- Customization for your specialized needs is available at a nominal fee.

Price: $150 plus $5 shipping and handling:
Visa and Mastercharge accepted.

Call: (813) 332-7556 for immediate shipping.

A Wooden Computer?

Not from Commodore!

So why should the desk look like wood? A pleasant cream and charcoal trimmed desk looks so much better with Commodore systems. One look and you'll see. Interlink desks are right. By design.

The specifications only confirm the obvious:

- Cream and charcoal color beautifully matches the Commodore hardware and blends with your decor.
- An ideal 710 mm (28") keyboard height yet no bumping knees because a clever cutout recesses the computer into the desk-top.
- High pressure laminate on both sides of a solid core for lasting beauty and strength.
- Electrostatically applied baked enamel finish on welded steel legs—no cheap lacquer job here.
- T-molding and rounded corners make a handsome finish on a durable edge that won't chip.
- Knocked down for safe, inexpensive shipment.
- Patented slip joints for quick easy assembly.
- Leveling glides for uneven floors.
- Room enough for a Commodore printer on the desk, yet fits into nearly any den or office niche— H: 660 mm (26") W: 1170 mm (46") D: 660 mm (26").
- Matching printer stand available with slot for bottom feeding.

Price: $299

In short, as Commodore dealers, we won't settle for anything that looks good only in the catalog! Our customers won't let us. They don't buy pictures. And neither should you. This is why **we will let you use one of our desks for a week** and then decide. If for any reason you don't like it, just return it in good condition for a cheerful refund.

If your Commodore dealer doesn't carry our desks yet, send a check for $299 and we will ship your desk freight paid!

Name____________________

Address____________________

City__________ **St**_____ **Zip**_____

Interlink, Inc., Box 134, Berrien Springs, MI 49103
Master Charge and Visa welcome. Call our order line:
616-473-3103

on or off if desired.

The blinking routines shown can be used in many applications. They give the impression that the computer is doing more than one operation at a time. Let these short routines tell your secretary to turn on the printer, or let them highlight when an inventory item is about depleted, or, most importantly, let them warn of unseen Klingons and those inoperative phasor banks!

Program 1.

```
 100 REM  BLINKING GET ROUTINE
 110 REM
 120 PRINT"SQQQQQQQQQQQQQQQQQQQQQQ"
 130 X$="▣": GOTO150
 140 X$="■"
 150 PRINTTAB(13);X$;"HIT ANY KEY]":H=TI
 160 IFTI>H+15THEN190
 170 GETS$:IFS$<>""THENRETURN
 180 GOTO160
 190 IFX$="▣"THEN140
 200 GOTO130
READY.
```

Program 2.

```
 10 PRINT"]"
 20 PRINT"STHE TIME IS ";LEFT$(TI$,2);":
    ";MID$(TI$,3,2);":";RIGHT$(TI$,2)
 30 GOSUB100:GOSUB200:GOSUB300
 40 GOTO20
 90 REM
 100 REM  BLINKING MESSAGE SUBROUTINES
 110 REM
 120 IFH1<>0THEN170
 130 X1$="▣":GOTO150
 140 X1$="■"
 150 PRINT"SQQQQQQQQQQQQQQQQQQQQQQ"
 160 PRINTX1$;"TIME IS PASSING]":H1=TI
 170 IFTI<H1+15THENRETURN
 180 IFX1$="▣"THEN140
 190 GOTO130
 200 REM
 210 REM
 220 IFH2<>0THEN270
 230 X2$="▣":GOTO250
 240 X2$="■"
 250 PRINT"SQQQQQQQQQQQQQQQQQ"
 260 PRINTTAB(15)X2$;"TIME HAS PASSED]":H2=TI
 270 IFTI<H2+25THENRETURN
 280 IFX2$="▣"THEN240
 290 GOTO230
 300 REM
 310 REM
 320 IFH3<>0THEN370
 330 X3$="▣":GOTO350
 340 X3$="■"
 350 PRINT"SQQQQQQQQQ"
 360 PRINTTAB(25)X3$;"TIME IS GONE]":H3=TI
 370 IFTI<H3+45THENRETURN
 380 IFX3$="▣"THEN340
 390 GOTO330
READY.
```

©

Interfacing A BSR X-10 AC Remote Control System To Your PET

C. W. Ward
Hazelwood, MO

The January 1980 issue of BYTE magazine published an article by Steve Ciarcia entitled "Computerize a Home." In this article Mr. Ciarcia described a method of expanding his home security system to include control of all the lights and AC outlets in the house. In his words, "*This expansion seems to be a contradiction considering previous concern over wiring costs. It would appear that every AC outlet would have to be directly wired to the computer through relays or some remote control capability would have to be added to each light and appliance.*"

It is this latter approach that is the basis for this and Mr. Ciarcia's article: more specifically, interfacing a BSR X-10 AC Remote Control System to a computer. His approach differs however, in that it proposes building a relatively simple, but not so inexpensive, hardware interface between his computer (a Radio Shack TRS-80 Model I) and the BSR X-10 command console. This article proposes a software oriented solution. In terms of hardware, it requires no more than the connection of an ultrasonic transducer (2 wires) to the appropriate pins of the PET's user port, thus resulting in an even lower cost. It should be noted that the concept applies to any system with an available output port (one line). The machine language program would require rework depending upon the speed of the system's clock (the PET has a 1 megahertz clock) and the number of cycles per instruction.

The BSR X-10 system consists of a group of electronic devices that control the electrical environment inside and outside of of your house. Lights can be turned on and off, dimmed or brightened. Plug-in appliances, television sets, and stereos can be turned on or off. No special wiring or complex installations are involved. The system is expandable by adding more modules, a cordless controller or other command consoles. The system is shown in photo 1 and photo 2. The components are: The command console, the wall switch module, the appliance module, and the lamp module. A cordless controller (not shown) is available, but is not an integral part of the system being presented in this article.

The BSR X-10 system is marketed by Sears, Penneys and Radio Shack, to mention just a few, under slightly different product names. The components of one vendor's system are interchangeable with those of another, with one very notable exception. There are two different types of command consoles: one model (X10-014301) that can be remotely controlled with a hand held cordless controller, and one model that cannot! The command console that can be controlled by the hand held cordless controller is a requirement. The buyer should make certain that the BSR X-10 system purchased includes this particular model.

Photo 1: The BSR X-10 command console, appliance module and lamp module.

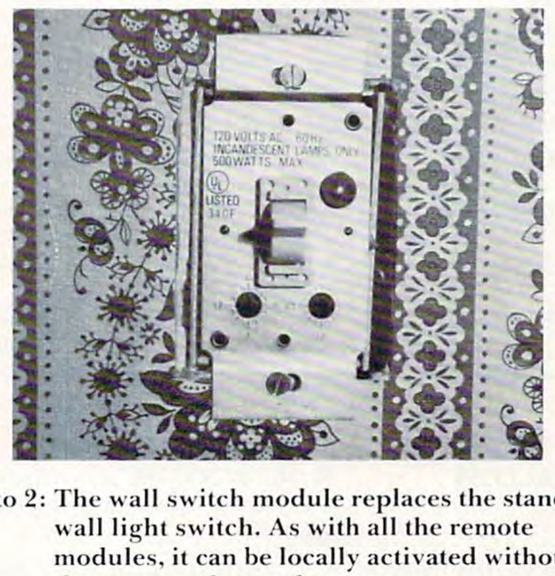


Photo 2: The wall switch module replaces the standard wall light switch. As with all the remote modules, it can be locally activated without the command console.

Photo 3: The 40K Hertz transducer, shielded cable and receptacle for the PET's parallel user port.

The BSR X-10 components are quite inexpensive when compared to the cost of the alternatives. The command console sells for $39 while each remote module sells for $15.

The command console of the system operates by sending coded signals through the house wiring to the lamp, appliance, and wall switch modules. Each remote module monitors these transmissions and responds only when its particular code is sent. The coded signals sent by the command console can be initiated by physically pressing a key on the console or by the receipt of a series of tone bursts in the proper sequence through the ultrasonic receiver section. This series of tone bursts would typically be transmitted by the hand-held cordless controller as a result of pressing one of its twenty-two keys. Its keyboard contains sixteen unit keys, numbered one through sixteen, each corresponding to the number set (or dialed) on the remote lamp, appliance, or wall switch module. The keyboard also contains six command keys labeled and defined as follows:

ON	Sends "TURN ON" command to selected module.
OFF	Sends "TURN OFF" command to selected module.
DIM	Sends "DIM" command to selected lamp or wall switch module.
BRIGHT	Sends "BRIGHTEN" command to selected lamp or wall switch module.
ALL LIGHTS ON	Sends "TURN ON" command to all lamp and wall switch modules simultaneously. (Does not affect appliance modules.)
ALL OFF	Sends "TURN OFF" command to every module, including appliance modules.

The user then selects a given lamp, appliance or wall switch module by pressing the appropriately numbered unit key and initiates the desired function by pressing the appropriate command key. This action actually transmits two separate messages to the command console.

Figure 1 describes the format of the coded messages (tone bursts) sent by the cordless controller to the command console via ultrasonnic communication. Each of the twenty-two keys on the controller has a unique 5-bit binary code (see table 1). A single message is made up of a start-of-message (SOM) code, one 5-bit binary code, the logical inversion of that 5-bit binary code, and an end of message (EOM) code. One message is approximately 100 ms in length and is composed of thirteen segments. Each segment is 8 ms in length. The start of message segment consists of a 4 ms 40K hertz tone followed by a 4 ms period of silence. Each segment of the data (5-bit binary code or inverted 5-bit binary code) consists of a 4 ms 40K hertz tone for a logic 1 or a 1.2 ms 40K hertz tone for a logic 0, followed by a silent period of the appropriate length. The end of message code consists of two 8 ms segments, each containing a 40K hertz tone for the complete duration. All messages use exactly the same format; only the 5-bit binary data codes vary.

Safety is the primary consideration. There is no hazard in using the controller or any of the remote modules as long as their cases remain intact. The BSR X-10 is Underwriters Laboratories listed. The PET must remain electrically isolated from the command console at all times. This is accomplished with communication in the form of ultrasonic sound transmitted through space by the transducer attached to the PET's user port. In essence, the PET software will simulate the activity of the hand-held cordless controller.

The hardware task consists of soldering the two wires of an output transducer to pin C (PAO) and pin A (GND) of a receptacle for the PET's parallel

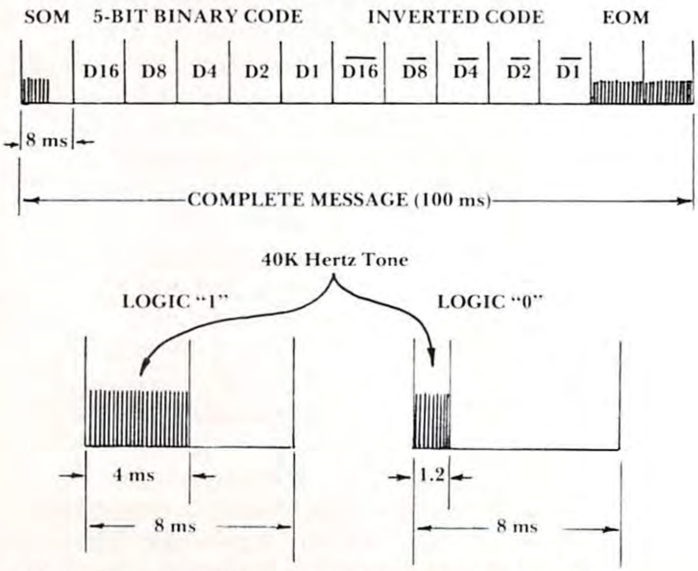


Figure 1: Description of coded message sent from cordless controller to the command console.

"STOCK BRIEF"

For Commodore Pet

A *USER-FRIENDLY* disk based file system for your securities
BUY / SELL / EVALUATE / EDIT / LIST / DIRECTORY

- Formatted reports to any IEEE interfaced printer
- Directory display of multiple stock files
- "Wild Card" search for stock name
- Dividend Yield calculated
- Automatic dating on all reports
- Automatic alphabetical stock name sort
- Automatic update of disk file when necessary

REQUIRES 16K MEMORY AND COMMODORE DISK UNIT

DISK & MANUAL $50.00 Post Paid

MYSTIC SOFTWARE

P.O. BOX 218 LEDYARD, CT 06339

Jinsam 8.0

has broken the sort limit. Sorts are now unlimited — you may sort to any desired depth: 3, 4, 5, 6, 7 ...

We now have:
unlimited record length, unlimited fields, and unlimited sorts, and limitless applications.

See our ad elsewhere in this issue.

NEED EXTRA RAM SPACE?

Running out of places to put your protection ROM's?

Or perhaps you are looking for protected RAM space to develop your Machine Language Program in?
Then you will be interested in our RAM Boards
We have RAM Boards with 1K bytes of RAM, 2K bytes and 4K bytes of RAM, that will plug in to any empty socket in your Pet or CBM computer with 24 pin sockets.
For those who are looking for utililty programs in EPROM (without empty sockets to put them in), we have a Board that plugs into the E900 (UD8) socket. It replaces the original ROM, which then plugs back into our little Board. This gives an additional 1¾K of M/L instructions to your machine. Such as DOS (wedge), screen dump, sort routine, upper lower case toggle, screen color invert, repeat key on all keys, etc, with single keystroke commands. The E900 Board can also be used as either a 1K or 1¾K RAM, with a simple move of a jumper.
All Boards are shipped with all IC's and RAM or EPROM in place ready to run, fully documented. All our Boards can be daisychained for even more RAM.
Please specify whether 3.0 or 4.0 ROM's and 40 or 80 column machine. And in the case of the E900 Boards, whether 1K, 2K, RAM or EPROM.
Prices are: 1K RAM - $35.00; 2KRAM - $65.00; 4K RAM - $120.00. E900 Board with 1K RAM - $55.00; with 1¾K RAM - $70.00; with EPROM only $65.00.
For orders or further information contact:

LAR MICROTRONIX
12987 - 96A Avenue
Surrey, B.C. Canada V3T 1A1
Tel. (604) 588-8967

Sorry, no charge cards accepted at this time.

ASERT yourself... with CFI's new Database Retrieval System

WHO CAN USE **ASERT**?

libraries
personnel departments
dating services

OFFICIALLY APPROVED BY COMMODORE

schools
employment agencies
accountants

ANY BUSINESS THAT KEEPS RECORDS CAN USE **ASERT** TO:

- Create up to 21 fields per record
- Restructure fields at any time
- Sort on any field at any time
- Use FREE-TEXT area for comments
- Create up to 90 searchwords
- Search & retrieve on any combination of 90 searchwords
- Search with MUST HAVE, MAY NOT HAVE and OPTIONAL operators
- Print out hardcopy including labels
- Output to any word processor
- Compile summary statistics
- Maintain 1900 records per disk with "virtual" 5K record length

ASERT — Aid for Search & Retrieval of Text — $495 complete

For the 8032 CBM and 8050 disk drive — Commodore Approved Software

OTHER CFI SOFTWARE

Federal Income Tax Preparation System*
Personal Tax Calculator*
Emergency Control Program*
VIC Animation Tutorial
*Distributed for CFI under the Commodore label

ALL CFI SOFTWARE AVAILABLE

from your local Commodore dealer
or direct from CFI

CFI . . . *computer solutions,* 875 West End Ave., New York, NY 10025 user port. The cable between the transducer and the receptacle is typically shielded (see photo 3).

Several sources for obtaining the 40K hertz transducer follow:

MASSA Laboratories Inc.
Hingham, MA
Part number TR-89

The Micromint
917 Midway
Woodmere, NY 11598
Part number 1002

The transducer shown in photo 2 was obtained from a local electronics surplus store, giving reason to believe that the 40K Hertz transducer is relatively common. Another possible source would be the service department of a local TV dealer. Most remote TV controllers today make use of ultrasonic transducers.

The effective range of the transducer is approximately twenty feet. When testing, various distances from the command module should be tried. Experience with several different transducers has shown that some failed to activate the console if positioned extremely close while others worked only in such a position.

Program 1 shows a simple BASIC driver program that serves only to demonstrate how the machine language sub-program should be used. Line 100 requests the user to enter the decimal data code for a given remote unit or a given function. If valid, the value is poked into location zero and the machine language program is executed. This is repeated several times to ensure success. During testing, this loop count should be increased. The REM statements in the program merely document the unit and function codes. If the program is listed just prior to running, the codes will be readily available (displayed on the PET screen) during the demonstration.

Program 2 is an assembled version of the 6502 machine language program required to generate the thirteen 8 ms segments that make up a single message. The routine begins by initializing the data direction register ($E843_{16}$) for VIA data output port A ($E84F_{16}$). PA0 is defined as an output line and PA1 through PA7, although not actually used, are defined as input lines. The remainder of the mainline routine is subdivided into segments containing calls to appropriate subroutines to produce the 8 ms segment in question. Labels on each of the subdivisions correspond to the field labels in figure 1 describing the complete message. The 40K hertz tone on the transducer is produced by alternating the logic value (0 or 1) of pin C (PA0) of the user port at a rate of 40,000 cycles per second. This is accomplished by the machine language subroutine labeled "X40KHZ". The period of time that a given 40K hertz tone is produced is determined by the number of times the subroutine continues to loop. The calling program sets register X accordingly. The periods of silence are accomplished by calling the machine language subroutine labeled "DELAY". Again, the

UNIT CODE	5-BIT BINARY CODE D16	D8	D4	D2	D1	DECIMAL EQUIVALENT
1	0	1	1	0	0	12
2	1	1	1	0	0	28
3	0	0	1	0	0	04
4	1	0	1	0	0	20
5	0	0	0	1	0	02
6	1	0	0	1	0	18
7	0	1	0	1	0	10
8	1	1	0	1	0	26
9	0	1	1	1	0	14
10	1	1	1	1	0	30
11	0	0	1	1	0	06
12	1	0	1	1	0	22
13	0	0	0	0	0	00
14	1	0	0	0	0	16
15	0	1	0	0	0	08
16	1	1	0	0	0	24

COMMAND	5-BIT BINARY CODE D16	D8	D4	D2	D1	DECIMAL EQUIVALENT
ALL OFF	0	0	0	0	1	01
ALL LIGHTS ON	0	0	0	1	1	03
ON	0	0	1	0	1	05
OFF	0	0	1	1	1	07
DIM	0	1	0	0	1	09
BRIGHT	0	1	0	1	1	11

Table 1: Cordless controller push-button codes and decimal equivalents.

```
100 INPUT"ENTER CODE";C
110 IFC<0ORC>30THENPRINT"INVALID CODE"
    :GOTO100
120 FORI=1TO3:POKE0,C:SYS(31744):NEXT
140 GOTO100
197 REM ------------------------------
198 REM ******************************
199 REM ------------------------------
200 REM   UNIT    CODE    UNIT    CODE
201 REM ------------------------------
202 REM    1       12       9      14
203 REM    2       28      10      30
204 REM    3        4      11       6
205 REM    4       20      12      22
206 REM    5        2      13       0
207 REM    6       18      14      16
208 REM    7       10      15       8
209 REM    8       26      16      24
210 REM ------------------------------
211 REM FUNCTION  CODE   FUNCTION CODE
212 REM ------------------------------
213 REM  ALL OFF    1  ALL LITES ON  3
214 REM    ON       5      OFF       7
215 REM   DIM       9    BRIGHT     11
216 REM ------------------------------
217 REM ******************************
218 REM ------------------------------
READY.
```

period of time is determined by the number of loops. The instruction timings are given in the comments field of the assembled instructions in each subdivision.

The machine language program, as assembled, will load into the high address end of RAM ($1C00_{16}$). It should be loaded and BASIC's upper memory limit reset prior to loading and executing the BASIC driver program (see a previous **COMPUTE!** article).

The approach used to time the various functions (ie. the program loops and sequences of instructions to produce the desired periods) does not interfere with the PET's clock nor does the clock interrupt handling software produce any undesired effects on these timing sequences. Turning on a table lamp, then, is as simple as poking a value and executing a machine language program. A sophisticated BASIC program can now be developed using the PET's time-of-day clock. Remote BSR X-10 modules can be placed around the home to control a variety of appliances and lights, and all at a much lower cost!

```
                      ORG    X'0000'
             * PAGE ZERO WORKING STORAGE
0000         CODE     DS     1X        CALLING PARAMETER
0001         ONE      DS     1X
0002         TEMP     DS     1X
             * SYSTEM EQUATES
             DDREG    EQU    X'E843'   DATA DIRECTION REGISTER
             PORT     EQU    X'E84F'   VIA DATA OUT PORT A
                      ORG    X'1C00'
1C00 A901    START    LDX    #1
1C02 8D43E8           STA    DDREG
1C05 8501             STA    ONE
             *--------------------------SEGMENT 1-------------
1C07 A2A0    SOM      LDX    #160
1C09 20631D           JSR    X40KHZ    4000,8
1C0C A24F             LDX    #79        2
1C0E 20791D           JSR    DELAY     6,3962
1C11 0502             ORA    TEMP      3
             *--------------------------SEGMENT 2-------------
1C13 A500    D16      LDA    CODE      3
1C15 2910             AND    $10       2
1C17 F00D             BEQ    OVER1     3*
1C19 A2A0             LDX    #160       2
1C1B 20631D           JSR    X40KHZ    6,4,4000,8
1C1E A24F             LDX    #79        2
1C20 20791D           JSR    DELAY     6,3962
1C23 4C331C           JMP    D8        3
1C26 A230    OVER1    LDX    #48        2
1C28 20631D           JSR    X40KHZ    6,4,1200,8
1C2B A287             LDX    #135       2
1C2D 20791D           JSR    DELAY     6,6762
1C30 4C331C           JMP    D8        3
             *--------------------------SEGMENT 3-------------
1C33 A500    D8       LDA    CODE
1C35 2908             AND    $08       (TIMINGS SAME AS ABOVE)
1C37 F00D             BEQ    OVER2
1C39 A2A0             LDX    #160
1C3B 20631D           JSR    X40KHZ
1C3E A24F             LDX    #79
1C40 20791D           JSR    DELAY
1C43 4C531C           JMP    D4
1C46 A230    OVER2    LDX    #48
1C48 20631D           JSR    X40KHZ
1C4B A287             LDX    #135
1C4D 20791D           JSR    DELAY
```

```
1C50 4C531C              JMP    D4
          *-------------------------SEGMENT 4-------------
1C53 A500   D4           LDA    CODE
1C55 2904                AND    $04       (TIMINGS SAME AS ABOVE)
1C57 F00D                BEQ    OVER3
1C59 A2A0                LDX    #160
1C5B 20631D              JSR    X40KHZ
1C5E A24F                LDX    #79
1C60 20791D              JSR    DELAY
1C63 4C731C              JMP    D2
1C66 A230   OVER3        LDX    #48
1C68 20631D              JSR    X40KHZ
1C6B A287                LDX    #135
1C6D 20791D              JSR    DELAY
1C70 4C731C              JMP    D2
          *-------------------------SEGMENT 5-------------
1C73 A500   D2           LDA    CODE
1C75 2902                AND    $02       (TIMINGS SAME AS ABOVE)
1C77 F00D                BEQ    OVER4
1C79 A2A0                LDX    #160
1C7B 20631D              JSR    X40KHZ
1C7E A24F                LDX    #79
1C80 20791D              JSR    DELAY
1C83 4C931C              JMP    D1
1C86 A230   OVER4        LDX    #48
1C88 20631D              JSR    X40KHZ
1C8B A287                LDX    #135
1C8D 20791D              JSR    DELAY
1C90 4C931C              JMP    D1
          *-------------------------SEGMENT 6-------------
1C93 A500   D1           LDA    CODE
1C95 2901                AND    $01       (TIMINGS SAME AS ABOVE)
1C97 F00D                BEQ    OVER5
1C99 A2A0                LDX    #160
1C9B 20631D              JSR    X40KHZ
1C9E A24F                LDX    #79
1CA0 20791D              JSR    DELAY
1CA3 4CB31C              JMP    NOTD16
1CA6 A230   OVER5        LDX    #48
1CA8 20631D              JSR    X40KHZ
1CAB A287                LDX    #135
1CAD 20791D              JSR    DELAY
1CB0 4CB31C              JMP    NOTD16
          *-------------------------SEGMENT 7-------------
1CB3 A500   NOTD16       LDA    CODE      3
1CB5 2910                AND    $10       2
1CB7 D00D                BNE    OVER6     3*
1CB9 A2A0                LDX    #160       2
1CBB 20631D              JSR    X40KHZ    6,4,4000,8
1CBE A24F                LDX    #79        2
1CC0 20791D              JSR    DELAY     6,3962
1CC3 4CD31C              JMP    NOTD8     3
1CC6 A230   OVER6        LDX    #48        2
1CC8 20631D              JSR    X40KHZ    6,4,1200,8
1CCB A287                LDX    #135       2
1CCD 20791D              JSR    DELAY     6,6762
1CD0 4CD31C              JMP    NOTD8     3
          *-------------------------SEGMENT 8-------------
1CD3 A500   NOTD8        LDA    CODE
1CD5 2908                AND    $08       (TIMINGS SAME AS ABOVE)
```

```
1CD7 D00D                BNE    OVER7
1CD9 A2A0                LDX    #160
1CDB 20631D              JSR    X40KHZ
1CDE A24F                LDX    #79
1CE0 20791D              JSR    DELAY
1CE3 4CF31C              JMP    NOTD4
1CE6 A230       OVER7    LDX    #48
1CE8 20631D              JSR    X40KHZ
1CEB A287                LDX    #135
1CED 20791D              JSR    DELAY
1CF0 4CF31C              JMP    NOTD4
               *--------------------------SEGMENT 9-------------
1CF3 A500       NOTD4    LDA    CODE
1CF5 2904                AND    $04         (TIMINGS  SAME AS ABOVE)
1CF7 D00D                BNE    OVER8
1CF9 A2A0                LDX    #160
1CFB 20631D              JSR    X40KHZ
1CFE A24F                LDX    #79
1D00 20791D              JSR    DELAY
1D03 4C131D              JMP    NOTD2
1D06 A230       OVER8    LDX    #48
1D08 20631D              JSR    X40KHZ
1D0B A287                LDX    #135
1D0D 20791D              JSR    DELAY
1D10 4C131D              JMP    NOTD2
               *--------------------------SEGMENT 10------------
1D13 A500       NOTD2    LDA    CODE
1D15 2902                AND    $02         (TIMINGS SAME AS ABOVE)
1D17 D00D                BNE    OVER9
1D19 A2A0                LDX    #160
1D1B 20631D              JSR    X40KHZ
1D1E A24F                LDX    #79
1D20 20791D              JSR    DELAY
1D23 4C331D              JMP    NOTD1
1D26 A230       OVER9    LDX    #48
1D28 20631D              JSR    X40KHZ
1D2B A287                LDX    #135
1D2D 20791D              JSR    DELAY
1D30 4C331D              JMP    NOTD1
               *--------------------------SEGMENT 11------------
1D33 A500       NOTD1    LDA    CODE
1D35 2901                AND    $01         (TIMINGS SAME AS ABOVE)
1D37 D00D                BNE    OVER10
1D39 A2A0                LDX    #160
1D3B 20631D              JSR    X40KHZ
1D3E A24F                LDX    #79
1D40 20791D              JSR    DELAY
1D43 4C531D              JMP    EOM
1D46 A230       OVER10   LDX    #48
1D48 20631D              JSR    X40KHZ
1D4B A287                LDX    #135
1D4D 20791D              JSR    DELAY
1D50 4C531D              JMP    EOM
               *--------------------------SEGMENT 12 & 13-------
1D53 A2FF       EOM      LDX    #255
1D55 20631D              JSR    X40KHZ
1D58 A2FF                LDX    #255
1D5A 20631D              JSR    X40KHZ
1D5D A282                LDX    #130
```

```
1D5F 20631D              JSR   X40KHZ
1D62 60                  RTS
            * GENERATE A 40K HERTZ FREQUENCY ON VIA PORT A BIT 0
            * FOR A LENGTH OF TIME DETERMINED BY REGISTER X
1D63 A900   X40KHZ       LDX   #00         2
1D65 EA                  NOP               2
1D66 8D4FE8 LOOP1        STA   PORT        4 ╮
1D69 6901                ADC   #1          2 |
1D6B D8                  CLD               2 |13
1D6C D8                  CLD               2 |
1D6D 8502                STA   TEMP        3 ╯
1D6F 8D4FE8              STA   PORT        4 ╮
1D72 6501                ADC   ONE         3 |12
1D74 CA                  DEX               2 |
1D75 D0EF                BNE   LOOP1       3*╯
1D77 EA                  NOP               2
1D78 60                  RTS               6
            * DELAY FOR A LENGTH OF TIME DETERMINED BY
            * THE VALUE  IN REGISTER X
1D79 0502   DELAY        ORA   TEMP        3
1D7B 0D0200              ORA   TEMP        4
1D7E 20901D LOOP2        JSR   RETURN      6,6╮
1D81 20901D              JSR   RETURN      6,6|
1D84 20901D              JSR   RETURN      6,6|
1D87 0502                ORA   TEMP        3   |50
1D89 0D0200              ORA   TEMP        4   |
1D8C EA                  NOP               2   |
1D8D CA                  DEX               2   |
1D8E D0EE                BNE   LOOP2       3* ╯
1D90 60     RETURN       RTS               6
1D91                     END   START-----------------------------------
```

©

Using Non-Pin-Feed Forms In The 2022 Tractor Printer

Rev. Jack Weaver
Homestead FL

Most of the forms we use in our operation are punched for our 2022 Tractor Feed CBM Printer. However there are some things we need to print out on our printer, things for which we could not justify the high cost of having printed. We would use relatively few over a year's time. Two examples that come to mind are Bank Deposit Slips and payroll checks. In my son's business we write three payroll checks each week and we have about 50 to 75 checks to deposit in the bank each week.

We have been using a very easy and unique way to print on standard checks and standard bank deposit forms using our 2022 printer. We have found it very exact and that it can be perfectly registered each time the forms are used.

Basically the method we use is as follows: We take one or more sheets of blank white fan-fold pin feed paper and very carefully lay the proposed forms out on the paper — tracing the outline of each one on the pin-feed paper. The corners of each form are clearly marked in black ink.

The pin-feed paper is then taken and laid on a flat surface outside and carefully sprayed with #1301 KRYLON brand Crystal Clear Acrylic spray coating. We mention this brand because we have found that it will be less likely to crack or peel. After the one side is completely dry — turn it over and spray the back side. Repeat for a total of four or five coats — allowing time for each coat to fully dry.

When the paper is completely dry you will in effect have a plastic sheet — perforated for your pin feed printer.

It is very important that you use a transparent type opaque cellophane tape and tape the tear perforations so it will not separate as it is being used. The tape should also be taped over the side and top margins of each form outline — so each form can be taped to the page and then removed without doing damage to the plasticized paper.

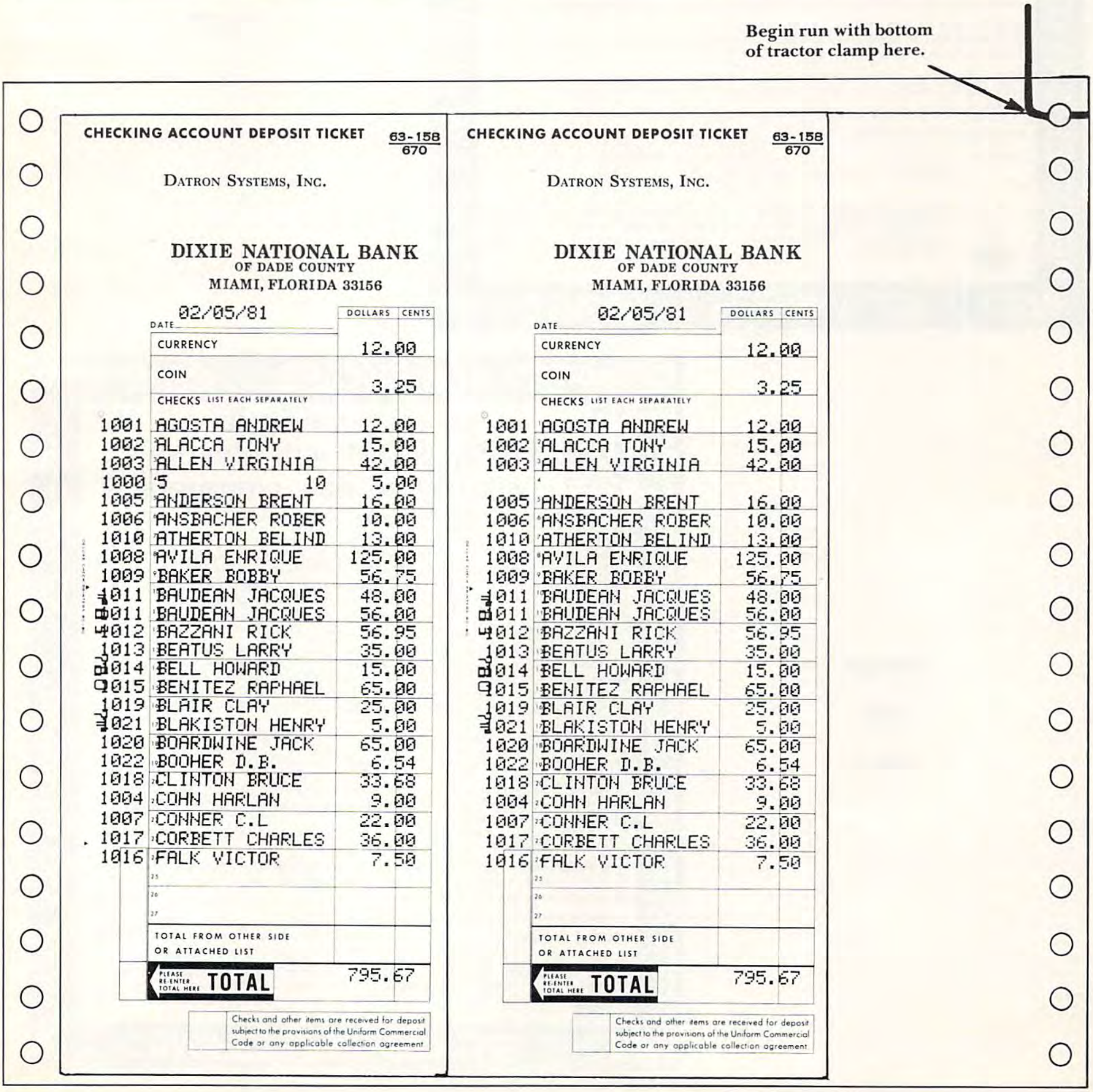
If you are very careful in drawing your outline of the forms — and careful in the format programming of the printer — you can make each copy exactly like the previous one.

Very important — It is necessary to draw a line on the plasticized sheet at the bottom of the lever on the right hand pin-feed mechanism. This will let you know where to start your printing to get perfect registration each time.

The tape on each form is almost unnoticeable since we use only a square one half-inch square at top and bottom of each form — which, when removed from the plasticized paper form holder, is simply folded over to the back side.

Our bank tellers were impressed and could not figure out how we had done it. When making Bank Deposits we use the form in conjunction with a program which credits the check to the customer while at the same time makes a very neat Bank Deposit Slip.

The possibilities for this are unlimited. We have been using one sheet for 6 months and it is still going strong — maybe for another 6 months!!



CHECKING ACCOUNT DEPOSIT TICKET 63-158/670

DATRON SYSTEMS, INC.

DIXIE NATIONAL BANK
OF DADE COUNTY
MIAMI, FLORIDA 33156

DATE 02/05/81

		DOLLARS / CENTS
	CURRENCY	12.00
	COIN	3.25
	CHECKS LIST EACH SEPARATELY	
1001	AGOSTA ANDREW	12.00
1002	ALACCA TONY	15.00
1003	ALLEN VIRGINIA	42.00
1000	5 10	5.00
1005	ANDERSON BRENT	16.00
1006	ANSBACHER ROBER	10.00
1010	ATHERTON BELIND	13.00
1008	AVILA ENRIQUE	125.00
1009	BAKER BOBBY	56.75
1011	BAUDEAN JACQUES	48.00
1011	BAUDEAN JACQUES	56.00
1012	BAZZANI RICK	56.95
1013	BEATUS LARRY	35.00
1014	BELL HOWARD	15.00
1015	BENITEZ RAPHAEL	65.00
1019	BLAIR CLAY	25.00
1021	BLAKISTON HENRY	5.00
1020	BOARDWINE JACK	65.00
1022	BOOHER D.B.	6.54
1018	CLINTON BRUCE	33.68
1004	COHN HARLAN	9.00
1007	CONNER C.L	22.00
1017	CORBETT CHARLES	36.00
1016	FALK VICTOR	7.50
	25	
	26	
	27	
	TOTAL FROM OTHER SIDE OR ATTACHED LIST	
	PLEASE RE-ENTER TOTAL HERE TOTAL	795.67

Checks and other items are received for deposit subject to the provisions of the Uniform Commercial Code or any applicable collection agreement.

CHECKING ACCOUNT DEPOSIT TICKET 63-158/670

DATRON SYSTEMS, INC.

DIXIE NATIONAL BANK
OF DADE COUNTY
MIAMI, FLORIDA 33156

DATE 02/05/81

		DOLLARS / CENTS
	CURRENCY	12.00
	COIN	3.25
	CHECKS LIST EACH SEPARATELY	
1001	AGOSTA ANDREW	12.00
1002	ALACCA TONY	15.00
1003	ALLEN VIRGINIA	42.00
	4	
1005	ANDERSON BRENT	16.00
1006	ANSBACHER ROBER	10.00
1010	ATHERTON BELIND	13.00
1008	AVILA ENRIQUE	125.00
1009	BAKER BOBBY	56.75
1011	BAUDEAN JACQUES	48.00
1011	BAUDEAN JACQUES	56.00
1012	BAZZANI RICK	56.95
1013	BEATUS LARRY	35.00
1014	BELL HOWARD	15.00
1015	BENITEZ RAPHAEL	65.00
1019	BLAIR CLAY	25.00
1021	BLAKISTON HENRY	5.00
1020	BOARDWINE JACK	65.00
1022	BOOHER D.B.	6.54
1018	CLINTON BRUCE	33.68
1004	COHN HARLAN	9.00
1007	CONNER C.L	22.00
1017	CORBETT CHARLES	36.00
1016	FALK VICTOR	7.50
	25	
	26	
	27	
	TOTAL FROM OTHER SIDE OR ATTACHED LIST	
	PLEASE RE-ENTER TOTAL HERE TOTAL	795.67

Checks and other items are received for deposit subject to the provisions of the Uniform Commercial Code or any applicable collection agreement.

CBM/PET? SEE SKYLES ... CBM/PET?

"Should we call it Command-O or Command-O-Pro?"

That's a problem because this popular ROM is called the Command-O-Pro in Europe. (Maybe Command-O smacks too much of the military.)

But whatever you call it, this 4K byte ROM will provide your CBM BASIC 4.0 (4016, 4032) and 8032 computers with 20 additional commands including 10 Toolkit program editing and debugging commands and 10 additional commands for screening, formatting and disc file manipulating. (And our manual writer dug up 39 additional commands in the course of doing a 78-page manual!)

The Command-O extends Commodore's 8032 advanced screen editing features to the ultimate. You can now SCROLL up and down, insert or delete entire lines, delete the characters to the left or right of the cursor, select TEXT or GRAPHICS modes or ring the 8032 bell. You can even redefine the window to adjust it by size and position on your screen. And you can define any key to equal a sequence of up to 90 key strokes.

The Command-O chip resides in hexadecimal address $9000, the rightmost empty socket in 4016 and 4032 or the rearmost in 8032. If there is a space conflict, we do have Socket-2-ME available at a very special price.

Skyles guarantees your satisfaction: if you are not absolutely happy with your new Command-O, return it to us within ten days for an immediate, full refund.

Command-O from Skyles Electric Works **$75.00**

Complete with Socket-2-Me **95.00**

Shipping and Handling .... *(USA/Canada)* **$2.50** *(Europe/Asia)* **$10.00**

California residents must add 6%/6½% sales tax, as required.

Skyles Electric Works
231E South Whisman Road
Mountain View, California 94041
(415) 965-1735

Visa/Mastercard orders: call tollfree **(800) 227-9998** (except California). California orders: please call **(415) 965-1735.**

WATCH

THIS

SPACE

WATCH

THIS

SPACE

CBM/PET? SEE SKYLES ... CBM/PET?

"You mean this one little Disk-O-Pro ROM will give my PET twenty-five new commands?

And for just $75.00? Why, that's only $3.00 a command!"

The Disk-O-Pro in any PET with Version III (BASIC 2.0) ROMs **(### COMMODORE BASIC ###)** will give 19 software compatible disk instructions*: 15 identical with the new BASIC 4.0 (or with 8032 ROMs) compatible with both old and new DOS. Plus 4 additional disk commands...including appending **(MERGE)**, overlaying **(MERGE #________)** and **PRINT USING**, allowing formatting output of strings and numbers on the PET screen or on any printer.

**NOTE: Old DOS doesn't recognize three of the commands.*

Those are just 3 of the important commands—and there are 7 more beauties—on your Disk-O-Pro that have never been available previously to PET/CBM users. (Skyles does it again!)... Beauties like the softtouch key **(SET)** which allows you to define a key to equal a sequence of up to 80 keystrokes; like **SCROLL** whereby all keys repeat as well as slow scrolling and extra editing features; like **BEEP** which allows you to play music on your PET.

The Disk-O-Pro is completely compatible with the BASIC programmer's Toolkit. The chip resides in the socket at hexadecimal address $9000, the rightmost empty socket in most PETS. And for the owners of "classic" (or old) PETS, we do have interface boards.

(For those owning a BASIC 4.0 or 8032, even though the Disk-O-Pro may not be suitable, the Command-O is. Just write to Skyles for additional information. Remember, we have never abandoned a PET owner.)

Complete with 84-page manual written by Greg Yob...who was having so much fun that he got carried away. We had expected 32 pages.

Skyles guarantees your satisfaction: if you are not absolutely happy with your new Disk-O-Pro ROM chip, return it to us within ten days for an immediate full refund.

Disk-O-Pro from Skyles Electric Works **$75.00**

Complete with interface board (for "classic" PETS) **95.00**

Shipping and Handling .... *(USA/Canada)* **$2.50** *(Europe/Asia)* **$10.00**

California residents must add 6%/6½% sales tax, as required.

Skyles Electric Works
231E South Whisman Road
Mountain View, California 94041
(415) 965-1735

Visa/Mastercard orders: call tollfree **(800) 227-9998** (except California). California orders: please call **(415) 965-1735.**

How And Why You Should Use PEEK (155) Instead Of GET

David M. Miller, Howard Beach, NY

In many two player action games, one player has a keyboard-screen control advantage over the other. By this I mean if player A holds down the "2" key for instance, and player B presses the "W" key, the PET will only acknowledge the depression of the "2." But if player B holds down the "W" first and player A presses the "2," the PET will again only be aware of the depression of the "2." This problem arises when you use the GET command. To illustrate this problem more clearly for those of you who are not familiar with it, type in the following program:

```
10 GET A$:IF A$ = "" THEN 10
20 PRINT A$: GOTO 10
```

If the "2" and "W" are pressed together (in action games it is likely that two keys may at the same time be depressed) the PET prints only the "2." Even if the "W" is pressed first, and then the "2," in the end, when both are depressed the PET will only print the "2." That, of course, gives player A a large advantage in the game.

The easiest way to overcome this unfairness is to use the memory content of PEEK(155) in Upgrade ROMs. I think it is PEEK (512) in Original ROMs, but I did not test it. Using this instead of GET gives both players an equal keyboard-screen control over his "man."

The only drawback using this method is that each player has control over only two keys each, so this feature would only be advantageous in certain games.

The left player must use the RVS key and SPACE key; the right player must use the left carat <, and left bracket [.

Memory 155 has a value of 255 unless one or more of the four keys mentioned above are depressed. When 2, 3, or all 4 keys are depressed independently or simultaneously, memory 155 has a special value which is given in the chart below.

Key(s) Depressed	Value of PEEK(155)
NONE	255
RVS	254
SPACE	251
<	247
[	253
RVS, SPACE	250
RVS, [	252
RVS, <	246
SPACE, [	249
SPACE, <	243
[, <	254
RVS, SPACE, [	242
RVS, SPACE, <	241
RVS, SPACE, [, <	240

Note: PEEK(155) never has a value of 244.

Using these values in your program you can branch off to a step which will carry out the required function. To illustrate how you can use this in a game situation, consider a game in which two players have two guns each. Part of the program may look as follows:

```
10 IF PEEK(155) = 254 THEN REM SHOOT TOP
LEFT GUN
20 IF PEEK(155) = 249 THEN REM SHOOT TOP
RIGHT AND BOTTOM LEFT GUN
30 IF PEEK(155) = 240 THEN REM SHOOT ALL
FOUR GUNS
```

Using PEEK(155) in place of GET makes the keyboard-screen control, and, in effect, the entire game, fair and equal for both players. I hope you can take advantage of this feature in programming your next interactive action game, or revising an old one. If you have any comments on this idea, please send them directly to me.

David Miller
149-45 83 St.
Howard Beach, NY 11414

©

AIM 65 BASIC Floating-Point Arithmetic From Machine Language

Paul Beasley,
Mobile, AL

Writing floating-point operations in machine language on a microprocessor is a "messy" proposition. I avoid it like the plague unless I absolutely must do it. But I have discovered how to use the floating-point routines in the AIM 65 BASIC ROM's. It's so easy even I do not mind floating-point applications any more.

AIM 65 BASIC Floating-Point Numbers

For those who are unfamiliar with floating-point numbers, particularly on the AIM 65, I'll describe the floating-point number format. Floating-point representations are similar to scientific notation. An example of a number written in normalized, scientific notation is .27 x 10^2 (=27). Computers commonly use a similar scheme except instead of 10 as a base, the base 2 is used (e.g., 27=.84375 x2^5). By storing the sign, the exponent of 2, and the mantissa of the number, a broad range of values can be efficiently represented. In the AIM 65, this is accomplished by storing each floating-point number in five consecutive bytes as follows:

1	2	3	4	5
E	M3	M2	M1	M0

S=bit 7

E=exponent
M3,M2,M1,M0=mantissa
S=sign

Note: Bits in a byte are numbered 0 (LSB) to 7 (MSB).

The exponent, E, is a power of 2 and is biased so that E=\$80 actually corresponds to a power of 0, E=\$7F corresponds to -1, E=\$81 corresponds to +1, etc. When a floating-point number is normalized, the mantissa is shifted so that the first 1 bit of the mantissa falls in bit position 7 of M3. This means that bit 7 of M3 will always be 1 and the exponent reflects the number of bits that the mantissa was shifted in order to have the implied decimal in front of the first 1 bit. E=\$80 means no shifts were required; E=\$81 means the mantissa was shifted right one bit; E=\$7F means the mantissa was shifted left one bit; etc.

Since bit 7 of M3 is always 1 using the above method, it is stripped off and restored only when performing arithmetic operations (this process is explained later). So, when a number is stored in memory, this bit position is used to store the sign of the number — 0 for positive and 1 for negative. (Incidentally, the floating-point representation of 0 is all five bytes equal \$00.) My previous example of the number 27 would be stored in memory as follows:

85 58 00 00 00

AIM 65 BASIC Floating-Point Accumulators

In order to use floating-point numbers in arithmetic operations, BASIC reserves twelve bytes in Page 0 to provide two floating-point accumulators. Accumulator 1 (FPAC1) is in locations \$A9 through \$AE and accumulator 2 (FPAC2) is in locations \$B1 through \$B6. Each accumulator spans six bytes and has the following format:

1	2	3	4	5	6
E	M_3	M_2	M_1	M_0	S

S = \$00 for +
\$FF for -

As I mentioned earlier, when numbers are stored into memory, the sign is put into bit 7 of M3. Technically, this is accomplished as follows:

(M_3 ∧ \$7F) — strips off most significant bit of M3

(S ∧ \$80) — strips off all bits except leftmost bit

∨ denotes logical OR
∧ denotes logical AND

Table 1. Calling sequences for floating-point operations.

OPERATION	CALLING SEQUENCE		
1. Load FPAC1	LDA	AL	source address
	LDY	AH	
	JSR	$C8E1	
2. Load FPAC2	LDA	AL	source address
	LDY	AH	
	JSR	$C7CB	
3. Store FPAC1	LDX	AL	destination address
	LDY	AH	
	JSR	$C913	
4. Copy FPAC1 to FPAC2	JSR	$C94B	
5. Copy FPAC2 to FPAC1	JSR	$C93B	
6. Convert fixed-point to floating-point	LDY	IL	
	LDA	IH	
		(result in FPAC1)	
7. Convert floating-point to fixed-point	Load FPAC1 with floating-point value		
	JSR	$C536	
	(result right-justified in M3-M0 of FPAC1		
8. Addition	Load FPAC1 with operand 1		
	LDA	AL	source address for minuend
	LDY	AH	
	JSR	$C58F	
	(Addressed value is loaded into FPAC2, FPAC1 is subtracted from FPAC2 and result in FPAC1; FPAC2 unchanged.)		
10. Multiplication	Load FPAC1 with operand 1		
	Load FPAC2 with operand 2		
	JSR	$C76F	
	(result in FPAC1; FPAC2 unchanged.)		
11. Division	Load FPAC1 with divisor		
	Load FPAC2 with dividend		
	JSR	$C851	
	(result in FPAC1; FPAC2 unchanged.)		
12. Power operation	Load FPAC1 with exponent		
	Load FPAC2 with base number		
	JSR	$CC7F	
	(FPAC2 is raised to the power in FPAC1; result in FPAC1; FPAC2 unchanged.)		
13. Multiply FPAC1 by 10;	JSR	$C821	
14. Divide FPAC1 by 10	JSR	$C83D	
15. Add .5 to FPAC1	JSR	$C588	
16. Convert floating-point number to ASCII string	Load FPAC1 with number		
	JSR	$CB1C	
	(result at $0200)		

Note: Resulting ASCII string starts at location $0200. The first character is a space, followed by the ASCII digits and ended with a $00 byte.

OPERATION			
17. Compare FPAC1 to memory	LDA	AL	source address of number in memory
	LDY	AH	
Branch to xxxx if:	JSR	$C99A	
memory<FPAC1	BCC	xxxx	
memory = FPAC1	BEQ	xxxx	
	BEQ	LABEL	
memory>FPAC1	BCS	xxxx	
LABEL	.		
	.		
	.		

Table 2. Intrinsic Function Subroutine Addresses

Basic Function	Address	Description
ABS	$C997	Absolute Value of FPAC1
COS	$CDD2	Cosine of FPAC1
EXP	$CCF1	Raises e to power in FPAC1
INT	$CA0B	Integer portion of FPAC1
LOG	$C729	Natural logarithm of FPAC1
NEG	$CCB8	Negation of FPAC1
RND	$CD96	Generates random number
SGN	$C978	Sign function of FPAC1
SIN	$CDD9	Sine of FPAC1
SQR	$CC75	Square root of FPAC1
TAN	$CE22	Tangent of FPAC1

The logical OR places the sign bit into M3.

When a number is loaded into one of the accumulators, the sign bit is separated out and made the sixth byte of the accumulator (as shown above) so that bit 7 of M3 can be restored to 1. This makes arithmetic operations easier and explains why the accumulators are six bytes each. My example of the number 27 would appear in an accumulator as:

85 D8 00 00 00 00

In addition to the accumulators, there are two other bytes in Page 0 that you should know about. These are the overflow (at $B0) and underflow (at $B8) bytes. The underflow byte is used for rounding M0 of FPAC1. The overflow byte becomes non-zero when a computational result becomes too large. It is important that these two bytes be initialized to zero before the first floating-point operation is performed. In relation to this, I must give a word of caution. The BASIC floating-point routines still "think" they are operating in the context of a BASIC program. This means that any computation error (e.g., overflow) which is normally trapped by BASIC will still be caught and your program terminated. The termination message may look peculiar since the BASIC statement and variable pointers in Page 0 probably contain meaningless values.

Performing The Floating-Point Operations

I have prepared Table 1 as a

— AIM 65 —

P/N		QTY 1-9
A65-1	AIM-65 w/1K RAM	$399
A65-4	AIM w/4K RAM	$439
A65-A	Assembler ROM	$ 85
A65-B	BASIC ROMS	$100
A65-PL	PL/65 ROMS	$125
A65-F	FORTH ROMS	$150

SPECIALS

A65-4AB	**AIM-65 w/4K RAM, Assembler & BASIC**	**$599**
A65-4B	**AIM-65 w/4K RAM, BASIC**	**$529**

Power Supplies (AIM-65 Compatible, Industrial Quality Open Frame)

PRS3	+5V at 3A, +24V at 1A w/mtg hardware, cord, etc	$ 75
PRS4	+5V at 2A, +24V at .5A w/mtg hardware, cord, etc	$ 60
PRS5	+5V at 2A, +24V at .5A, ± 12V to ± 15V at .4A	$ 75
PRS6	+5V at 3A, +24V at .5A, + 12V at 2A	$ 85

From The Enclosure's Group

ENC1	AIM-65 case	$ 45
ENC1A	AIM-65 case w/space for one expansion bd	$ 50

Cases With Power Supplies (for ENC1A Add $5)

ENC3	ENC1 w/PRS3 mounted inside	$125
ENC4	ENC1 w/PRS4 mounted inside	$110
ENC5	ENC1 w/PRS5 mounted inside	$125
ENC6	ENC1 w/PRS6 mounted inside	$135

From Optimal Technology

ADC1	A/D eight channels, D/A 2 channels, requires ± 12V to ± 15V at 100 MA & 2-I/O Ports from AIM-6522	$126
	w/Cable for AIM-65	$150

From The Computerist

MCP1	Mother Plus™ Dual 44 pin mother card & card cage, fully buffered, 5 expansion slots underneath the AIM	$150
MEB1-2	DRAM Plus™ 32K RAM, 16K PROM sockets, 2-6522 I/O chip and programmer for 5V EPROMS	16K RAM $325 32K RAM $395
PTC1	Proto Plus II™ Prototype card same size as KIM-1, MEB1-2, VIB1 (Bare Bd $60) assembled	$ 75
VIB1	Video Plus™ bd with 128 char, 128 user char, up to 4K display RAM, light pen and ASCII keybd interface	$325
CBL1	CABLE for MEB1-2, VIB1, PTC1	$ 25

NEW PRODUCTS! **CUBIT** featuring a 4½x6½ AIM-65* for $195
*less monitor, printer, display & keybd.
APPLIED BUSINESS COMPUTER featuring
64K RAM Board - $495
80 column Video Controller - $325
5¼" Disk Controller w/ADOS™ - $485

From Seawell MicroSystems

MCP2	Little Buffered Mother™ Single 44 pin (KIM-4 style) mother card. Has on bd 5V regulator for AIM-65, 4 expansion slots. Routes A&E signals to duplicates on sides with 4K RAM	$199
MEB2	SEA 16™ 16K static RAM bd takes 2114L with regulators and address switches	$250
MEB2-3	CMOS RAM, realtime clock, EPROM bd, up to 8K RAM, 16K EPROM. (w/1k CMOS, 7K NMOS)	$395
PGR2	Programmer for 5V EPROMS with ROM firmware, regulators, low force sockets, up to 8 EPROMS simultaneously, can execute after programming	$299
PIO2	Parallel I/O bd with 4-6522's	$260
PTC2	Proto/Blank™ Prototype card that fits MCP2	$ 49
PTC2A	Proto/Pop™ with regulator, decoders, switches	$ 99
FDC2	Floppy disk controller bd & DOS, up to four 8" drives, double sided, double density (DD/DS)	$425
SBC2	SBC/CPU card, 9K RAM, 18K EPROM, 3 serial ports, 1 parallel port, audio tape interface	$495

From Micro Technology Unlimited (MTU)

DAC3	8 bit DAC bd	$ 49
FDC3	Floppy disk controller bd & DOS, up to four 8" drives, double sided, double density, 16K DRAM, Boot PROM	$595
MCP3	Card file w/4 slot expansion mother bd w/keybd brackets	$ 85
MEB3-2	Banker Board™, low power, 32K DRAM	$450
PIO3	24K PROM, 4-8 bit I/O ports w/RS-232 port to 4800 bps, PROM Programmer.	$295
VIB3	8K DRAM bd, low power, w/composite video out in 200 lines 320 dot/line format	$240
PTC3	Prototype Bd w/regulators	$ 42
MPS3	AIM-65 Power Supply w/12V for MTU Bds, can drive one 8" disk drive.	$ 65
CBL3	Cable for MEB3-2, VIB3, FDC3	$ 25

All MTU Software Available For These Products

Miscellaneous

TPT3	Approved Thermal Paper Tape, 3/165' rolls	$7.50
MEM6	6/2114 RAM Chips	$ 45
CAS1	Audio Cassette Recorder	$ 40
CAS1-1	CAS1 w/cable	$ 65
2716	16K 5V EPROM	$ 10
2532	32K 5V EPROM	$ 35
A65-P	Printer	$ 70
A65-DM	Display Module-DL1416	$ 30
FDD8	QUME Data Trak 8, DD/DS Disk Drive/up to 1 Megabyte	$650
FD8C-1	Cable set for 1 drive w/AC cord	$ 50
FD8C-2	Cable set for 2 drives w/AC cords	$ 65
MON1	9" composite video monitor w/80 char line resolution. Requires 12V DC only at .8A	$125

All AIM-65 Spare Parts Are Available
All "Microflex" Products Available
COMING SOON!
AIM-65/40 w/32K RAM- $1395

ASSEMBLED & TESTED SYSTEMS

We have been specializing in assembled and tested systems made from the above items for over 2 years. Normally, the price will be the total of the items, plus $5 for handling; shipping is extra on all COD's or invoiced orders. Please call or write for exact prices or if questions arise. **Six month warranty on all systems.**

Higher quantities quoted upon request, COD's accepted, shipping will be added. Add $5 for shipping, insurance, and handling on prepaid orders. Minnesota residents add 5% sales tax. Prices subject to change without notice.

July 1, 1981

Mail Check or Money Order to:

Educational Computer Division
EXCERT INCORPORATED

- SALES
- SERVICE
- INSTALLATION
- CONSULTING

P.O. Box 8600
White Bear Lake, MN 55110
(612) 426-4114

reference for the fundamental floating-point operations along with their appropriate machine language calling sequences. All operations are executed with the subroutine jump instruction (JSR) plus minimal parameter set-up. In preparing the table I used the following notation:

AL – Address Low; the least significant 8 bits of the source or destination memory address.

AH – Address High; the most significant 8 bits of the source or destination memory address.

IL – Integer Low; the memory address of the least significant 8 bits of a 2-byte integer value.

IH – Integer High; the memory address of the most significant 8 bits of a 2-byte integer value.

FPAC1 – Floating Point Accumulator 1.

FPAC2 – Floating Point Accumulator 2.

In addition to the fundamental operations in Table 1, the BASIC intrinsic functions may also be used. The common calling sequence for these functions is as follows:

```
load FPAC1 with the argument
value
JSR   $xxxx (select function
  address from Table 2)
(result in FPAC1)
```

The entry point address for each of the functions is given in Table 2.

Sample Program

In order to illustrate what I have just described, I have included the following sample program. It is a very simple calculation of the volume of a cylinder using the formula $V = \pi r^2 h$, where r = radius and h = height. I know that r^2 can be computed as r times r very efficiently, but I used the power function to illustrate its use. When the program finishes (successfully), it will display V = 88357.2935. Another tidbit I'll point out is that the floating-point representation for 2π is at location $CE53 of the BASIC ROM's.

Sample Program: Calculate Volume of Cylinder ($V = \pi r^2 h$)

```
*=$0220
COMIN=$R1A1                       ; monitor entry for command input
EQUAL=$E7D8                       ; output "=" to display/printer
OUTPUT=$E97A                      ; output char. in A to diaplay/printer
CRLOW=$EA13                       ; output CR & LF to display/printer
FMUL=$C76F                        ; floating-point multiply
CONVIF=$C0D1                      ; convert fixed-point to floating-point
CONVFA=$CB1C                      ; convert floating-point to ASCII string
FST1=$C913                        ; store FPAC1
FLD2=$C7CB                        ; load FPAC2
CPY12=$C94B                       ; copy FPAC1 to FPAC2
FDIV=$C851                        ; division
FPWR=$CC7F                        ; power operation
PI2=$CE53                         ; 2*
START     LDY    R                ; get radius
          LDA    #0
          STA    $B8              ; initialize underflow
          STA    $B0              ; and overflow bytes
          JSR    CONVIF
          LDX    #<TEMP
          LDY    #>TEMP
          JSR    FST1             ; store R in TEMP
          LDY    #2
          LDA    #0
          JSR    CONVIF           ; exponent 2 in FPAC1
          LDA    #<TEMP
          LDY    #>TEMP
          JSR    FLD2             ; load R in FPAC2
          JSR    FPWR             ; raise R to power 2
          LDX    #<TEMP
          LDY    #>TEMP
          JSR    FST1             ; store R squared in TEMP
          LDY    H
          LDA    #0
          JSR    CONVIF           ; height H in FPAC1
          LDA    #<TEMP
          LDY    #>TEMP
          JSR    FLD2             ; load FPAC2 with R squared
          JSR    FMUL             ; FPAC1=H times R squared
          LDA    #<PI2
          LDY    #>PI2
          JSR    FLD2             ; load 2*     innto FPAC2
          JSR    FMUL             ; FPAC1=H times R squared times 2
          JSR    CPY12            ; save FPAC1 in FPAC2
          LDY    #2
          LDA    #0
          JSR    CONVIF           ; FPAC1=2
          JSR    FDIV             ; divide by 2
          JSR    CONVFA           ; resulting volume in FPAC1
          JSR    CRLOW
          LDA    #'V'
          JSR    OUTPUT           ; display 'V'
          JSR    EQUAL            ; display '='
          LDX    #0
LABEL1    LDA    $0200,X          ; fetch & display ASCII digits
          BEQ    LABEL2
          JSR    OUTPUT
          INX
          JMP    LABEL1
LABEL2    JSR    CRLOW
          JMP    COMIN
R         .BYTE  25               ; radius=25
H         .BYTE  45               ; height=45
TEMP      .BYTE  0,0,0,0,0
          .END
```

©

A General Purpose BCD-To-Binary Routine

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

A number of routines have been published [1,2,3] that will convert either a two-digit number or a four-digit number in BCD code to a binary number, and Butterfield[4] has published a routine to handle a six-digit BCD number. The routine described here can be easily modified to handle any number of BCD digits. It is a 6502 assembly language interpretation of an algorithm found in Peatman's[5] book. The BCD-to-binary routine assumes its importance from the fact that human beings usually input numbers to a computer in a decimal representation. A number of scientific instruments have BCD outputs that may be interfaced to a microcomputer, requiring some kind of conversion routine before the data from such a device can be processed. Finally, if you want to interface some of the calculator chips to a microprocessor in order to do more complex arithmetic, you will very likely need a BCD-to-binary routine somewhere in your software. A 6502 assembly language routine to go the other way (binary-to-BCD) can be found as a subroutine in reference six at the end of this article.

The BCD-to-binary routine is based on a familiar technique for converting a base-ten number to a base-two number. The decimal number is successively divided by two, and the remainders are noted as either a one or a zero. Each division gives the next more significant binary digit or bit. Example 1 illustrates the process.

Example 1. Convert 59_{ten} to a binary number.

Solution: Successively divide 59_{ten} by two, with the divisions beginning from the right and proceeding to the left.

0	1	3	7	14	29
2 1	2 3	2 7	2 14	2 29	2 59
0	2	6	14	28	58
1	1	1	0	1	1

$59_{ten} = 111011_{two}$

Referring to Example 1 it can be seen that the algorithm requires that the BCD number be successively divided by two and the remainders are saved to become the binary number. The first division remainder is the least significant bit, while the remainder from the last division is the most significant bit. If in Example 1 we wanted to convert 59_{ten} to an eight-bit binary number, namely 0011 1011, we would simply perform two more divisions than shown, providing the two leading zeros in the eight-bit representation.

If you are mildly familiar with BCD numbers you will recall that each digit requires four bits (or one nibble). So an eight-digit decimal number requires four memory locations. Conversely, four memory locations can represent a decimal number as large as 99999999, which is more easily expressed as 10^8-1. Question: How many bits are needed to represent a given number of decimal digits? Let N be the largest number of decimal digits that we need for our particular application, so the largest decimal number is (10^N-1). Let n be the number of binary digits (bits) needed to represent the same number. By analogy, the largest binary number that can be represented by n bits is (2^n-1). Since we wish to represent the same number, we may equate (10^N-1) and (2^n-1) and then solve for n. Thus, with some mathematical magic, the answer to the question posed above is

$$N = N/\log 2 = N/0.30103$$

where a base ten logarithm is implied.

If N = 8 then n = 26.6 which becomes n = 27 when rounded upward (fractional numbers of bits are not allowed as answers for this problem). Twenty-seven bits can be handled quite nicely by four bytes, *but please do not create* your own theorem that the number of memory locations needed to represent a number in binary is equal to the number of memory locations to represent the same number in binary-coded decimal (BCD). Use the equation, and be sure to allocate enough memory to handle the number in either binary or BCD representations. Note that, in the program described by Listing 1, we assume an eight-digit decimal number is being converted to a binary number that will also be stored in four memory locations. The program is easily modified to handle situations where the number of memory locations needed for the BCD number is *different* than the number of memory locations needed for the binary number. Using the immortal words of many authors, "we leave this problem for the student."

So we know how many memory locations to assign to represent the number, and we have a simple algorithm (divide by two and store the remainder) to perform the conversion. Enter some corollary to Murphy's Laws: "nothing is as simple as it seems." Dividing by two is neat and easy for a binary number: successive shifts to the right (LSR or ROR) give successive divisions by two. Dividing by two is considerably more complex for a BCD number. Fortunately, Peatman[5] has pointed out a few tricks that accomplish division-by-two for a BCD number.

The eight bit "weights" in a byte of memory that represent a binary number are 1, 2, 4, 8, 16, 32, 64, and 128, proceeding from the right-most bit to the left-most bit. Clearly, shifting the number to the right divides each bit weight by two. That is why an LSR or an ROR instruction may be used to divide a binary number by two. However, if the same memory location represents a BCD number, then the bit weights are 1, 2, 4, 8, 10, 20, 40, 80. consequently, a shift-right or a rotate-right instruction results in division-by-two only for bits zero, one, two, three, five, six, and seven. Shifting bit four (with a weight of ten) to the right changes its weight to eight. Eight is three more than five, the number you usually get when you divide ten by two. So, the trick to dividing a BCD number by two is to shift right or rotate right as usual, but if a one is shifted from bit four to bit three, then you must subtract three from the shifted-right result to get the correct answer. That's it folks. I wish I could say it was my idea, but I found it in Peatman's[5] book.

```
BCDNUM  = $0000;   Base address of the BCD number to be converted to binary. The
                   most-significant digit of an N digit BCD number is in the high-
                   order nibble of BCDNUM.
BINUM   = $0010;   Base address of the binary number whose most-significant byte
                   will be in BINUM.
BYTE    = $FC;     Twos complement of the number of bytes needed to hold
                   the BCD number; in this program four bytes ($0000 - $0003)
                   are used.

$ 0D00  D8         START   CLD               Clear decimal mode.
  0D01  A9 00              LDA #00           Clear locations that will
  0D03  A2 FC              LDX #BYTE         hold the binary number.
  0D05  95 14      BACK    STA BINUM+4,X
  0D07  E8                 INX
  0D08  D0 FB              BNE BACK          Locations have been
  0D0A  38                 SEC               cleared.
  0D0B  A2 FC      THERE   LDX #BYTE         Rotate the binary number
  0D0D  76 14      RETURN  ROR BINUM+4,X     right, moving the remainder
  0D0F  E8                 INX               from the BCD division into
  0D10  D0 FB              BNE RETURN        the binary number.
  0D12  B0 2B              BCS OUT           If the carry is set, the conver-
  0D14  A2 FC              LDX #BYTE         sion is complete.
  0D16  76 04      AGAIN   ROR BCDNUM+4,X    Start the division-by-two by
  0D18  E8                 INX               shifting BCD number right.
  0D19  D0 FB              BNE AGAIN         Remainder will be in carry
  0D1B  08                 PHP               flag so save it on the stack.
  0D1C  A2 FC              LDX #BYTE         Test bit three of each byte to
  0D1E  38                 SEC               see if a one was shifted in.
  0D1F  B5 04      LAKE    LDA BCDNUM+4,X
  0D21  29 08              AND #08           If so, subtract three.
  0D23  F0 06              BEQ FORWD         If not, no correction needed,
  0D25  B5 04              LDA BCDNUM+4,X    so test bit seven of each byte
  0D27  E9 03              SBC #03           to see if a one was shifted in.
  0D29  95 04              STA BCDNUM+4,X
  0D2B  B5 04      FORWD   LDA BCDNUM+4,X    Here bit seven is checked.
  0D2D  29 80              AND #$80
  0D2F  F0 06              BEQ ARND          No correction.
  0D31  B5 04              LDA BCDNUM+4,X    Correction: subtract 30.
  0D33  E9 30              SBC #$30
  0D35  95 04              STA BCDNUM+4,X
  0D37  E8         ARND    INX
  0D38  D0 E5              BNE LAKE          Repeat for all N bytes.
  0D3A  28                 PLP               Get the carry back because it
  0D3B  B0 CE              BCC THERE         held the remainder.
  0D3D  90 CC              BCC THERE         Go back and put it in the
  0D3F  60         OUT     RTS               binary number. Then finish.
```

If the BCD number is to be represented by several bytes, an added complication occurs. Bit seven in the least-significant byte has a weight of 80. Bit zero in the next most significant byte has a weight of 100. Clearly, shifting a one from bit zero of this byte to bit seven of the least-significant byte does not result in a division-by-two because 100/2 is not 80. However, if we subtract 30 after the shift we do get the correct answer. When performing a divide-by-two operation on a multi-byte BCD number, each byte in the number must be tested to see if a one was shifted into either bit three or bit seven, and then the appropriate remedies must be applied if the tests are positive. In short, if a one is shifted into the most-significant bit position of any of the N nibbles used to represent the N digits in BCD, then the nibble must be corrected by subtracting three.

One other point remains to be made. From Example 1 it is clear that we are interested in the remainder after division-by-two. When dividing by two, the remainder is either zero (even dividend) or one (odd dividend). The remainder will be found in the carry flag after a shift-right operation.

Although the comments should make most of the routine understandable, a brief explanation follows. The first instruction in Listing 1 is not needed if the program is already in the binary mode, in which case the routine starts by clearing the locations used to store the binary number. Rather than inserting some kind of loop counter to keep track of the number of divisions-by-two (refer to Example 1), the carry is set by the instruction location at $0D0A, and the remainders are rotated into the binary number until the carry bit that was initially set has rotated through the binary number and into the carry flag once more. Thus, the conversion stops at the BCS OUT instruction at location $0D12. The division-by-two routine takes up the remainder of the program. Note that the carry flag holds the remainder, and it is stored on the stack while the division-by-two routine is finished, after which it is rotated into the binary number in the RETURN loop.

Very likely some improvements in the speed of the routine could be made. In most cases the number of bytes needed for the binary number will be sufficiently close to the number of bytes occupied by the BCD number that no modification will be needed on that score. Remember, BYTE is the *twos complement* of the number of bytes used to represent the BCD number and the binary number. The BCD number is stored in locations $0000 - $0003 in Listing 1, in the sense that the least-significant digit is in the low-order nibble of location $0003 and the most-significant digit is in the high-order nibble of location $0000. These locations must be filled before the routine is called.

REFERENCES

1. **Programming and Interfacing the 6502, With Experiments**, De Jong, Marvin L., Howard W. Sams & Co., Inc., Indianapolis, 1980, p 129.
2. **6502 Assembly Language Programming**, Leventhal, Lance A., Osborne/McGraw-Hill, Inc., Berkeley, 1979, p 7-9.
3. **6502 Software Design**, Scanlon, Leo J., Howard W. Sams & Co., Inc., Indianapolis, 1980, p 147.
4. "Multi-Mode Adder," Butterfield, Jim, **6502 User Notes**, No. 13, p 23.
5. **Microcomputer-Based Design**, Peatman, John B., McGraw Hill, New York, 1977, p 400.
6. "A Floating-Point Binary to BCD Routine," De Jong, Marvin L., **COMPUTE!**, 1981, in press.

©

CAPUTE!
Corrections And Clarifications

COMPUTE! #14, pg. 68 — To clarify Mr. Victor's comments on conversions:

> In Applesoft BASIC, "DIM A$(40)" means that memory should be reserved for the string variables A$(0), A$(1),...,A$(40). (Note: this is 41 variables, not 40, a quirk of BASIC.) The variable "A$" is another variable (the 42nd) used to store the various strings discussed in the article. The statement "DIM A$(40)" at the start of the programs in the article does not mean that A$ will be 40 characters long. The DIM statement does not refer to A$, it refers to the subscripted variable A$(i).

COMPUTE! #15, pg. 80 — the Editor's Note on halting the dynamic mode should have read POKE 842, 12.

COMPUTE! #15, pg. 64 — at the bottom of the page, following the words "clears the flag," this paragraph was omitted:

> But is this efficient? Of course! How could you possibly reduce such a trivial, two-line program segment? Well, maybe you can! Consider the following alternative:

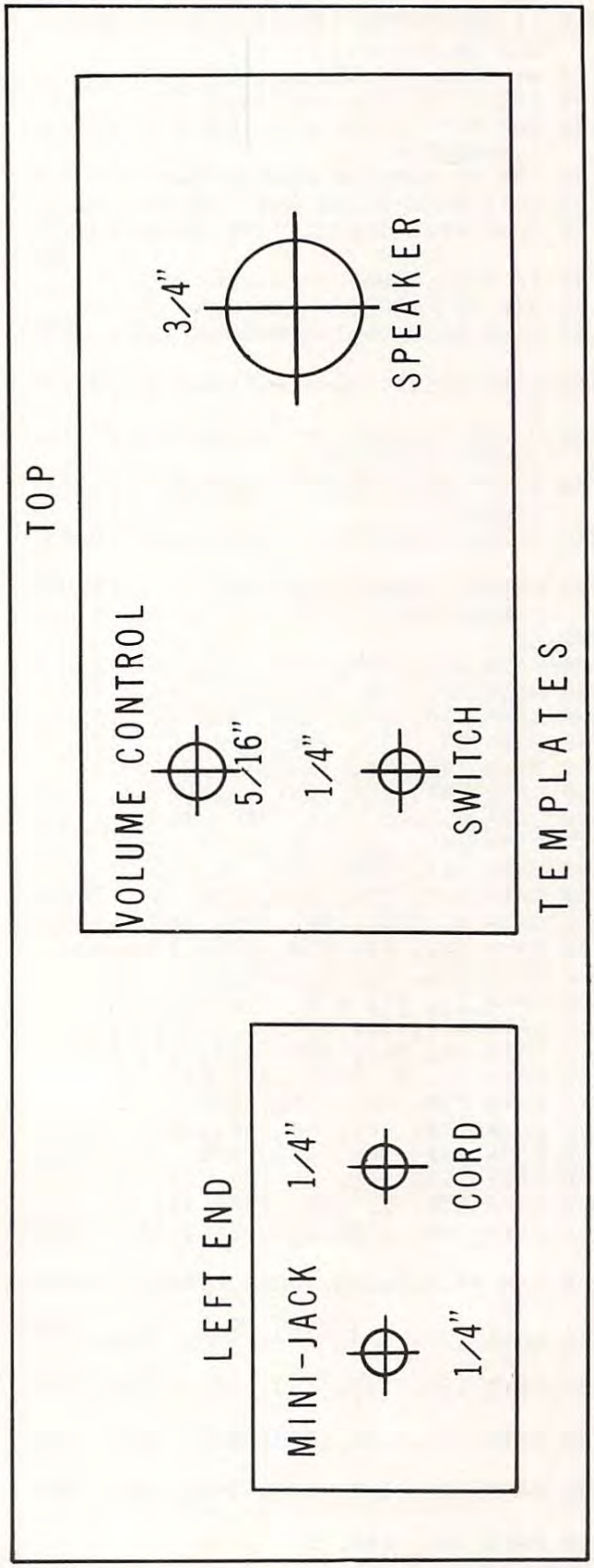
```
     SEC
     ROR ALFALK        ;Enable Alpha-Lock Mode
sets the flag, and
     LSR ALFALK        ;Disable Alpha-Lock Mode
clears the flag.
```

COMPUTE! #15, pg. 99 — the following program was not printed:

```
10000 REM DEFINE A LINE
10010 GRAPHICS 8+16:SETCOLOR 4,5,10
10020 POKE 204,COLUML:POKE 205,COLUMH
10030 FOR I=1600 TO 1636
10040 READ X:POKE I,X:NEXT I
10050 X=USR(1600)
10060 DATA 104,120,166,205,141,10,212,14
1,24,208,200,177,204,141,23,208,200,208,
2,230,205,173,11
10070 DATA 212,201,16,16,4,160,0,134,205
,177,204,24,144,223
```

COMPUTE! #16, pg. 86 — To permit "SHOOT" to run on the new (Revision B) ROMs, change line 201 to: BUF$(589,589) =CHR$(95):BUF$ (590,590)=CHR$(228). To permit it to run on black and white TVs, change line 202 to: BUF$ (97,97)=CHR$(0):BUF$ (98,98)=CHR$(0). The source code correction is: 124B 4C 5F e4 JMP SYS VBV.

COMPUTE! #15, pg. 79 — these templates should have appeared full-size:



COMPUTE! #15, pg. 120 — Charles Brannon has improved his "Keyword" program to cancel the keyword action in the quote mode and to permit it to be deactivated.

```
100 REM KEYWORD LOADER SEPTEMBER 8, 1~
   ~981
110 IF PEEK(PEEK(53)*256)<>120 THEN P~
   ~OKE 53,PEEK(53)-1:CLR
120 HI=PEEK(53):BASE=HI*256
130 PRINT "{CLEAR}PATIENCE..."
140 FOR I=Ø TO 164:READ A:POKE  BASE+~
   ~I,A:NEXT I
150 REM RELOCATION ADJUSTMENTS
160 POKE BASE+22,HI:POKE BASE+58,HI
170 POKE BASE+100,HI:POKE BASE+110,HI~
   ~
180 IF PEEK(50003)=160 THEN 230
190 REM CONVERSIONS FOR 3.Ø  BASIC
200 POKE BASE+65,146:POKE BASE+69,192~
   ~
210 POKE BASE+131,46:POKE BASE+132,23~
   ~Ø
220 PRINT "{CLEAR}*** KEYWORD 3.Ø ***~
   ~{DOWN}":GOTO240
230 PRINT "{CLEAR}*** KEYWORD 4.Ø ***~
   ~{DOWN}"
240 PRINT "ON/OFF:    SYS {REV}";BASE~
   ~
250 PRINT "{DOWN}TABLE AT:      {REV}~
   ~";BASE+138
260 END
270 DATA 120, 165, 144, 201, 37, 208
280 DATA 10, 169, 85, 133, 144, 169
290 DATA 228, 133, 145, 88, 96, 169
300 DATA 37, 133, 144, 169, 63, 133
310 DATA 145, 88, 96, 234, 234, 234
320 DATA 234, 234, 234, 234, 234, Ø
330 DATA Ø, 165, 205, 208, 89, 234
340 DATA 165, 217, 201, 193, 144, 82
350 DATA 201, 219, 176, 78, 56, 233
360 DATA 193, 170, 189, 138, 63, 162
370 DATA Ø, 134, 158, 170, 160, 178
380 DATA 132, 31, 160, 176, 132, 32
390 DATA 160, Ø, 10, 240, 16, 202
400 DATA 16, 12, 230, 31, 208, 2
410 DATA 230, 32, 177, 31, 16, 246
420 DATA 48, 241, 200, 177, 31, 48
430 DATA 17, 8, 142, 164, 63, 230
440 DATA 158, 166, 158, 157, 111, 2
450 DATA 174, 164, 63, 40, 208, 234
460 DATA 230, 158, 166, 158, 41, 127
470 DATA 157, 111, 2, 169, 20, 141
480 DATA 111, 2, 230, 158, 76, 85
490 DATA 228, 234, 234, 234, 234, 234~
   ~
500 REM ** KEYWORD TOKEN LIBRARY HERE~
   ~ **
510 DATA 153, 194, 152, 157, 132, 129~
   ~
520 DATA 137, 200, 133, 161, 135, 155~
   ~
530 DATA 163, 130, 159, 151, 160, 138~
   ~
540 DATA 148, 167, 187, 149, 147, 148~
   ~
550 DATA 141, 158, Ø
READY.
```

©

New Products

AUTO-CAT ... The Automatic Answer Direct Connect Modem

An auto answer, FCC approved, direct connect 300 baud modem, is now available from Novation, Inc., Tarzana, California.

The Bell 103 compatible AUTO-CAT will communicate at 300 baud over dial-up telephone lines using a standard modular jack. It has three data modes: automatic answer, manual answer, and manual originate. It will operate in either full or half-duplex, and features both local and remote-loopback test functions. The interface between computer and modem is the EIA RS-232.

Pressure sensitive switches on the end of the AUTO-CAT case select its answer or originate functions. LED's give a constant indication of the unit's operational status. The compact 10", 4.7", 1.2" modem uses a separate AC power supply that eliminates heat and voltage hazards.

In addition to communication over standard telephone lines, the AUTO-CAT will also automatically answer each call. This feature offers users the benefits of unattended operation for personal and business computers and terminals. Data can be made available 24 hours-a-day. Using AUTO-CAT, executives can communicate with their office computers in the evenings, on week-ends, even on holidays or vacations, and hobbyists can access their home computers from just about any location with a phone.

The AUTO-CAT is priced at $249.00 for quantity one and is available throughout the country

from authorized Novation distributors, dealers, computer stores, and retail electronic outlets. The AUTO-CAT provides the user with access to data banks, and the ability to swap personal programs with other computer users. The AUTO-CAT is designed to maximize the potential of communications with any computer or terminal.

For further information contact Novation, Inc., 18664 Oxnard St., Tarzana, CA 91356. Telephone 213-996-5060.

TV/Monitor Stand For Atari And Apple II

BYTM Systems, Inc. announces the introduction of a new TV/Monitor stand for ATARI 400/800™ and Apple II™ consoles.

The stand is made of clear polished high-impact acrylic. It securely holds a TV/Monitor up to 15 inches (diagonal) in screen size. Its open design allows maximum air flow around the console and provides easy access to top, sides and rear for cable or ROM cartridge insertion.

The use of the stand reduces workspace crowding while raising the TV/Monitor to eye level.

For ATARI 400/800 owners with ATARI 810™ Disk Drives, it also provides a simple means of minimizing the potential problem identified by ATARI that:

> Placing the ATARI 810 Disk Drive within 12 inches of a television set or monitor causes alteration of the data. Use the disk drive a minimum of 12 inches from the TV or monitor.

The CLEAR STAND raises the TV/Monitor 10 inches above the table level and makes leaving a diagonal space of 12 inches from the disk drive an easy problem to solve.

CLEAR STAND is available for $59.95 from BYTM Systems, Inc., 389 Fifth Avenue (Suite 400), New York, New York 10016.

VisiFile Program Simplifies Electronic Filing

Sunnyvale, CA — September 1, 1981 — File management on a personal computer — record filing, searching, sorting, printing reports and mailing labels — is fast and simple with the new

VisiFile™ program from Personal Software Inc.

VisiFile greatly increases the usefulness of a personal computer for keeping business records. The program's flexibility allows many different applications — inventory, client lists and records, sales information, medical records and other word or numerical data — to be stored, sorted and printed in a variety of formats.

"The 'human interface' of the program makes it extremely easy to use," said Ed Esber, director of merchandising. "Even people who are unfamiliar with computers will be able to master the program and begin to use it right away. Instructions are simple and direct and are selected from an easy-to-understand 'menu' system."

The VisiFile FlexFormat™ feature makes it easy to change, rearrange and add unforeseen information to records, or combine business records into new

files. This means that record keeping can adapt quickly as business information needs change. For example, the change from a five-digit to nine-digit zip code could be handled without rekeying all the data. Users may also creata a "partial file definition" for extremely fast data entry of specific portions of records.

The ability of VisiFile to communicate with other Personal Software products (such as the VisiCalc®, VisiPlot™ and VisiTrend™/VistPlot™ programs) makes it part of an information management software system for personal computers. This ability to exchange data reinforces the power of the total software system; information created or stored by one program need not be rekeyed in order to be used by another.

VisiFile records (and information stored by the other "Visi" products) may be transferred over phone lines by Personal Software's VisiTerm™ program.

Moving Cursor Menu

VisiFile is controlled by a "moving cursor menu" with prompting, similar to the kind of user interface in other Personal Software products. Information entry is simple and allows the user to custom design a "form-like" format on screen for input. Machine language sorting routines and multiple keyed-field indexes assure information retrieval within three seconds.

Mailing labels up to five across and reports may be printed with VisiFile. When ready to print a report, the user may use a simple "row-and-column" format, or develop a custom format area six rows deep by the width of the paper. Elements of each record may be arranged in that area to suit the user. Records may be selected for printing (in addition to being sorted or indexed): for example, all customers from Kansas with orders greater than $10,000 may be selected at the time the report is printed.

All reports can be easily set up, saved with the program. Arithmetic calculations, including column totals, also can be handled in reports.

VisiFile runs on the Apple® II (with language card or Applesoft Basic card) and Apple II Plus personal computers with 48K minimum memory and one disk drive; two disk drives and a printer are recommended. Suggested retail price of the program is $250 (U.S. price only). It will be

we beat the price...

Exclusive One Year Warranty Included

800™ $779

WITH ONE YEAR EXTENDED WARRANTY

ATARI®
Computers for people.™

ATARI 810 DISC DRIVE

$444

Available without warranty for even less.

ATARI SOFTWARE

CX401 General Accounting	$399.00
CX402 Accounts Receivable	$399.00
CX403 Inventory Control	$399.00
CX404 Word Processor	$119.00
CX405 PILOT	$68.00
CX413 MICROSOFT BASIC	$68.00
CX4101 Invitation to Programming 1	$17.00
CX4102 Kingdom	$13.00
CX4103 Statistics	$17.00
CX4104 Mailing List	$17.00
CX4105 Blackjack	$13.00
CX4106 Invitation to Programming 2	$20.00
CX4107 Biorhythm	$13.00
CX4108 Hangman	$13.00
CX4109 Graph It	$17.00
CX4110 Touch Typing	$20.00
CX4111 Space Invaders	$17.00
CX4112 States & Capitals	$13.00
CX4114 European Countries & Capitals	$13.00
CX4115 Mortgage & Loan Analysis	$13.00
CX4116 Personal Fitness Prog	$59.00
CX4117 Invitation to Programming 3	$20.00
CX4118 Conversational French	$45.00
CX4119 Conversational German	$45.00
CX4120 Conversational Spanish	$45.00
CX4121 Energy Czar	$13.00
CX4125 Conversational Italian	$45.00
CX6001 U.S. History	$23.00
CX6002 U.S. Government	$23.00
CX6003 Supervisory Skills	$23.00
CX6004 World History	$23.00
CX6005 Basic Sociology	$23.00
CX6006 Counseling Proced	$23.00
CX6007 Principal of Act	$23.00
CX6008 Physics	$23.00
CX6009 Great Classics	$23.00
CX6010 Business Comm	$23.00
CX6011 Basic Psychology	$23.00
CX6012 Effective Writing	$23.00
CX6014 Principals of Econ	$23.00
CX6015 Spelling	$23.00
CX6016 Basic Electricity	$23.00
CX6017 Basic Algebra	$23.00
CX8106 Bond Analysis	$20.00
CX8107 Stock Analysis	$20.00
CX8108 Stock Charting	$20.00
CXL4001 Education System Master	$21.00
CXL4002 Basic Computing Language	$46.00
CXL4003 Assembler Editor	$46.00
CXL4004 Basketball	$24.00
CXL4005 Video Easel	$24.00
CXL4006 Super Breakout	$30.00
CXL4007 Music Composer	$45.00
CXL4009 Chess	$30.00
CXL4010 3-D Tic-Tac-Tow	$24.00
CXL4011 Star Raiders	**$32.00**
CXL4012 Missle Command	**$32.00**
CXL4013 Asteroids	**$32.00**
CXL4015 TeleLink	$20.00
Compuhome	$74.95
Visicalc	$149.00
Letter Perfect (Word Processor)	$119.00
Source	$89.00

Atari® Peripherals:

400 16K	$329.00
410 Recorder	$59.00
822 Printer	$359.00
825 Printer	$ CALL
830 Modem	$159.00
850 Interface	$ CALL

Atari® Accessories

New DOS 2 System	$21.00
CX70 Light Pen	$64.00
CX30 Paddle	$18.00
CX40 Joy Stick	$18.00
CX853 16K RAM	$89.00
Microtek 16K RAM	$75.00
Microtek 32K RAM	$169.00

commodore

CBM 8032 $1099

4016	$799.00
4032	$999.99
8096	$1795.00
CBM4022 Printer	$629.00
Tally 8024	$1699.00
CBM C2N Cassette Drive	$69.00
CBM4040 Dual Disk Drive	$999.00
CBM8050 Dual Disk Drive	$1349.00

VIC 20 $259

Vic-TV Modual	$19.00
Vic Cassette	$69.00
Vic Disk Drive	$ Call
Vic 6 Pack program	$44.00

CBM Software

WordPro3 Plus	$199.00
WordPro4 Plus	$299.00
Commodore Tax Package	$399.00
Visicalc	$149.00
EBS Accts. Rec./Inventory Interactive Syst.	$595.00
BPI General Ledger	$329.00
OZZ Information System	$329.00
Dow Jones Portfolio	$129.00
Pascal	$239.00
Legal Time Accounting	$449.00
World Craft 80	$289.00
Word Check	$180.00
Create-A-Base	$219.00
Power	$89.00
Socket-2-Me	$20.00
Jinsam	Call

Disks

CX8100 Blank Disk (5)	$22.00
Sycom Blank Disk (10)	$29.00
Maxell Blank Disk (10)	$36.00
Maxell Blank Disk (10)	$46.00

Printers

Epson MX-70	Call for Prices
Epson MX-80	Call for Prices
Epson MX-80 FT	Call for Prices
Diablo 630	Call for Prices
Nec 5530	$2495.00
TEC 1500 Starwriter 25cps	$1495.00
TEC 1500 Starwriter 45cps	$1795.00

No Risk, No Deposit On Phone Orders, COD or Credit Card, Shipped Same Day You Call *

* on all in stock units

IN PA, CALL (717) 327-9575

(800) 233-8950

To Order:

Phone orders invited (800 number is for order desk only). Or send check or money order and receive free shipping. Pennsylvania residents add 6% sales tax. Add 3% for Visa or M.C. Equipment is subject to price change and availability without notice. Please call between 11 AM & 6 PM.

COMPUTER MAIL ORDER
501 E. 3RD ST., WILLIAMSPORT, PA 17701

available in October.

Atari Sponsors Research Efforts In Education

Sunnyvale, CA — The founding of the Atari Institute for Educational Action Research, an organization that will foster the innovative, yet practical, use of personal computers in education, was announced today by Raymond E. Kassar, chairman and chief executive officer of Atari®, Inc.

The Institute will provide grants of Atari computer products and/or cash stipends to selected institutions, individuals or organizations able to develop and promulgate new uses for computers in education, whether that usage will take place in established institutions, in community programs or in the home.

Dr. Ted M. Kahn, 32, formerly an education consultant to the computer division of Atari, has been named executive director of the institute, which will be located in the company's headquarters here. Kahn has been active in research and development in the use of computers in education for more than ten years.

Grants totaling more than $250,000 in cash and equipment will be given during the Institute's first year of operation. Initially, all support will be for domestic U.S. programs, and not for those from overseas.

"The use of personal computers is fast becoming an integral part of the mainstream of American culture," Kassar said. "We feel we should take an active role in supporting those who are working to push the computer to its full potential as a learning tool. We are looking for applications which will appeal to and serve broad sectors of society."

"Atari has, within the past year, already given major cash and equipment grants to projects at the Lawrence Hall of Science Computer Education Project (University of California at Berkeley); the future center at the Capital Children's Museum (Washington, D.C.); and the I.E.C. Mobile Computer Van (Santa Clara County, co-sponsored by the Industry Education Council and the Computer-Using Educators of California)," Kassar added. "The Institute will give us a formal channel through which to focus our efforts in this area."

One key program is the support of a small number of model schools or alternative learning centers to act as "centers of excellence: to illustrate various uses of computers in education."

For further information contact J. Peter Nelson at Atari, Inc., 1265 Borregas Ave., P.O. Box 427, Sunnyvale, CA 94086.

LO•BALL COMPUTERS LO•BALL COMPUTE

apple

ATARI

EPSON

... and more

Apple II

32K	$1099.00
48K	$1149.00
Disk II w/3.3 DOS	$529.00
Disk II	$450.00
Apple III w/128K	$3600.00

EPSON

MX-70	$399.00
MX-80	CALL
Apple card & cable	$99.00

ATARI

400 w/16K	$439.00
800 w/16K	$775.00
810 Disk Drive	$449.00

We'll attempt to meet or beat any advertised price.

HEWLETT PACKARD

HP-85A w/16K	$2697.30

NEC

5510-2 w/Tractor	$2550.00
5520-2 w/Tractor	$2850.00

SOROC

IQ 120	$725.00
IQ 135	$799.00

C.ITOH

Comet	$499.95
Starwriter	$1450.00

Northstar, Altos and Zenith All 25% Discount

LO-BALL COMPUTERS
7677 S.W. Cirrus Dr.
Beaverton, OR. 97005

TO ORDER
CALL (503) 641-0211

Ordering Information: For fastest service, send money order, cashier's check or bank wire. Visa and MC orders, add 3%. Personal checks accepted (allow minimum 10 days to clear). Hours 9-5, M-F.

Call for our Free Catalog.

LO•BALL COMPUTERS LO•BALL COMPUTE

LOWER THAN DISCOUNT

ON ALL MAJOR BRANDS

SUPER SPECIAL

ATARI 800
$747.00

ATARI D.D.
$ 449.00

DYSAN DISKETTES	17.50/box
ATARI SOFTWARE	20% off
EPSON MX80	469.00
DC HAYES MICROMODEM II	279.00
NOVATION D CAT	160.00
NEC 12" MONITOR	219.00

VISA

unicomm INC.

THE PURCHASING SERVICE FOR HOME & BUSINESS COMPUTERS
P.O. BOX 77-C BEVERLY HILLS, CA. 90213 (213) 451-8089

CALL OR WRITE FOR FREE CATALOG

PAY LESS.

ATARI
A Warner Communications Company

GET MORE.

PRICE EXAMPLES:

ATARI 800 with 16K memory	$749
ATARI 400 with 16K memory	$339
ATARI 810 Disk Drive	$439
ATARI Software 10% OFF	

Mail Order Customers

- Indicate Unit(s) you want
- Shipping Address • Mass. res. 5% sales tax
- Include check, money order, MC or VISA number

We ship by best means possible freight collect

WARREN'S COMPUTER SYSTEMS
4 Elm St., Braintree, MA 02184 (617)843-1133
After 5:30 EST please dial (617) 961-3285
SEND FOR CATALOG

commodore spectacular

8032-32K 80 COL CRT REG. $1495	**$1095**
4032-32K 40 COL CRT REG. $1295	**$995**
4016-16K 40 COL CRT REG. $995	**$795**
8050-DUAL DISK 950K REG. $1795	**$1345**
4040-DUAL DISK 343K REG. $1295	**$995**
C2N-CASSETTE DRIVE REG. $95	**$69**
4022-80 COL PRINTER REG. $795	**$629**
8024-MANNSMAN TALLEY REG. $1995	**$1595**
25CPS-STARWRITER REG. $1895	**$1445**
CBM-IEEE MODEM REG. $279	**$229**
VOICE SYNTHESIZER REG. $395	**$329**
VIC 20 ★★★★ REG. $299	**$269**
PET TO IEEE CABLE REG. $39.95	**$34**
IEEE TO IEEE CABLE REG. $49.95	**$39**

SOFTWARE

	LIST	YOU PAY
OZZ—The Information Wizard	$395	$299
Wordcraft 80	$395	$299
IRMA—Info Retrieval & Mgmt. Aid	$495	$399
Dow Jones Portfolio Mgmt.	$149	$119
Pascal Development Pkg.	$295	$229
EBS—Receivables, Inventory	$750	$579
BPI—General Ledger	$395	$299
Word Pro 3—40 Column	$250	$179
Word Pro 4—80 Column	$375	$269
Word Pro 4 Plus	$450	$329

CUSTOM PRINTED CONTINUOUS MINI-FORMS

- LOOK PROFESSIONAL
- LOW PRICES FOR SMALL RUNS

1000-11″x9½″ BLACK

PARTS	LETTERHEAD	INVOICE	STATEMENT
1	37.50	—	37.50
2		63.50	63.50
3		79.50	
5		115.50	

- SPECIAL LOGOS, 2 COLOR PRINTING & SPECIAL STOCK AVAILABLE

MINI-FORMS PRICES & ORDERS:
(800) 523-3442
PA & West of Rockies (215) 455-9705

Main Line Computer Discount

P.O. BOX 170
ST. DAVIDS, PA 19087

MAIL ORDER PHONE
800-345-8112
IN PA CALL 1-800-662-2444

- PREPAID ORDERS SHIPPED FREE
- VISA & MASTERCARD ADD 3%
- COD-UPS
- PA RESIDENTS ADD 6% PA TAX

NECC-82 Call For Participation

The 1982 National Educational Computing Conference (NECC-82) will be held in Kansas City, Missouri, June 28-30, 1982. The purpose of the conference is to provide a broad and rich forum for discussion among individuals interested in educational computing. Based on previous conferences, approximately 1000 people from institutions at all levels are expected to attend.

Authors are invited to submit papers describing actual experiences with computer use in the classroom or the consequences of concrete results or be survey or tutorial papers which include a synthesis and thorough evaluation. Generally, papers that describe projects presented at previous conferences are not considered unless substantial new information can be reported. In this case provide a brief synopsis of the earlier paper clearly indicating the new information. It is expected that most papers will report on specific accomplishment. Papers reporting negative results are also encouraged, especially when the results could have a profound effect in the way educational computing should be viewed. The deadline for submission of papers is Jan. 15, 1982.

Specifications for paper submittal may be obtained from:

Gerald L. Engel
Computer Science Dept.
Christopher Newport College
50 Shoe Lane
Newport News, VA 23606

or from:

E. Michael Staman
NECC-82 General Chairman
Campus Computing Services
University of Missouri-Columbia
305 Jesse Hall
Columbia, MO 65211

Individuals are also invited to submit proposals for special sessions, exhibits, project presentations, and "Birds of a Feather" sessions. For further information write to E. Michael Staman.

New Book For OSI C1P/C4P Users

Los Alamos, NM — TIS, Inc., the company that for the past two years has provided workbooks for the Ohio Scientific C1P computer, announces a new book. "Understanding Your C1P/C4P: A Woorkbook of BASIC Exercises" is a 112 page softcover book that is designed specifically for the Ohio Scientific C1P, SUPERBOARD II, and Challenger C4P. It introduces the fundamentals of OSI BASIC: calculator and program mode, input and output, data representation, and program storage on cassette. The book describes OSI control and logic including testing and branching, subroutine use, and logical operations. It covers character strings and array handling. Understanding Your C1P/C4P contains many exercises and sample programs. It is available from your Ohio Scientific dealer or by writing TIS, Inc., P.O. Box 921, Los Alamos, NM 87544. Price is $7.95 plus $2.00 shipping and handling.

Computer Graphics Spotlighted At National Conference On Visual Communications

National Conference on Computer Graphics: Tools for Productivity, December 7–9, 1981, Washington, D.C. The Conference will spotlight trends in usage and applications of computer graphics, as well as perspectives in emerging services and management needs for visual information. Computer graphics equipment and services demonstrations and corporate and government case histories will be included for an overall look at effectively integrating visual and textual information. Contact U.S. Professional Development Institute, 12611 Davan

Drive, Silver Spring, MD 20904. Telephone: (301) 622-0066.

Software Package For Small Manufacturers

COMPUMAX ASSOCIATES, Inc., of Palo Alto, CA, announces the newest addition to its MicroBiz product line of Accounting Software, with the release of MICROBOMP. A 'bill of manufacturing/materials program' for the microcomputer.

MICROBOMP is a software package specifically for the small manufacturer. This package is designed to handle inventory control, materials requirement planning and it will also 'explode' your bills of material. The program sets-up and maintains three files; 1) a stock file that contains the inventory master information; 2) a bill of materials file that defines the product structures and 3) a schedule file that contains production schedule data.

In addition a data base manager allows the operator to input and update information in these files. The bill of materials processor generates the 'product tree' up to six levels deep. Shortages are flagged at run time. Seperate reports provide stock valuation, ABC analysis, materials requirement listings and a 'MaxMin' report.

The program will, finally, create a journal file automatically, for those using MICROBOMP interactively, either with COMPUMAX's MICROLEDGER program, or with MAXILEDGER, the new Compumax program for divisional accounting.

MICROBOMP is written in Microsoft Basic, requires 48K and retails for $350.00. For further information contact:

COMPUMAX ASSOCIATES, Inc.
P.O. Box 1139
Palo Alto, CA, 94302
415/321-2881 X56

Software Development System For CBM, Apple, Atari

Eastern House Software has announced the release of their unbundled MAE Software Development System for the 2001/4001/8032 Commodore, Apple II 3.2/3.3 DOS, and Atari 400 or 800 computers.

If a software developer has a microcomputer in his laboratory or office, he can transform it into a development system via purchase of the MAE software package.

A main advantage of this software is that it is available for several different microcomputers. When a software developer moves from a PET to an APPLE to an ATARI etc., he need not be concerned with the time consuming task of having to relearn a new set of syntaxes, commands, and other peculiarities because this software works similarly on each of these microcomputers.

This software was designed for ease of use and to aid programmer productivity via its extensive text editing capabilities and numerous other programmer development utilities included in the package.

Since the Assembler/Editor are co-resident, continual manual loading of the editor, then assembler, then editor, etc. is not necessary. The main features include: Macros, Conditional Assembly, Interactive Assembly, up to 31 characters/label, string search and/or replace, plus numerous other editing facilities. Even a word processor is included to aid in the development of program descriptions, manuals, and other text. A number of other utilities are also provided — depending on the version, they include Word Processor, forward/reverse scrolling, tape interface, machine language macro library, library of disk driver subroutines, source of information file, relocating loader, etc.

A cross-assembler version which assembles 6800 source code is available and other cross assemblers are in the works. A detailed spec sheet is available to anyone interested in this software. Price $169.95. Contact:

Eastern House Software
3239 Linda Drive
Winston-Salem, NC 27106

Utility Package For PET/CBM Users

Professional Software Inc. (PSI) of Needham, MA has introduced POWER, a programmer's utility package in a 4K ROM designed for use with Commodore CBM/PET computers.

POWER is for any CBM/PET user who would benefit by reduced program entry time, debugging tools, and easy, quick modifications and updates.

POWER contains a series of new commands and utilities which are added to the BASIC Interpreter. Designed for the user of CBM/PET BASIC, POWER also contains special editing, programming, and debugging features. Included are special keyboard "instant action" features which make up for and go beyond the limitations of CBM/PET BASIC. POWER is sold with complete documentation written by Jim Butterfield of Commodore fame who has been working with preliminary versions of POWER for over a year.

To make POWER even more "user-friendly", PSI has included new "stick-on" keycap labels (which denote POWER's most commonly used features) in every program.

There are currently three (3) versions of POWER, one each for:

40 column CBM/PET with 3.0 BASIC;

40 column (9" or 12") CBM/PET with 4.0 BASIC;

and CBM 8032 (4.0 BASIC) ©

PRICE BREAKTHROUGH

We've done it again! As a leader in small computer sales we must constantly find new ways to save customers money while offering quality products. Our most popular seller, the Apple II Plus, is now available with 64K of user RAM at the unbeatable price of only $1249. This is possible because we manufacture the 16K RAM Card that expands the factory Apple II 48K to it's maximum capacity of 64K. The Ram Card allows use of Integer and Applesoft Basic and other languages like Pascal. It's a must for large data bases, Visicalc, and the Z-80 cards. The card is made from high quality components and has a full one year warranty.

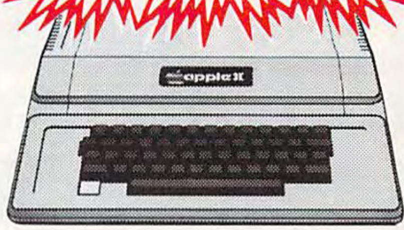
64K APPLE II PLUS*

**48 K Apple II Plus with 16K Ram Card*

ONLY **$1249**

FREE SHIPPING*

*on all pre-paid cash orders



16K ONLY **$1025** **48K** ONLY **$1089**

DISK II DRIVE WITH CONTROLLER CARD **$499**

DISK II DRIVE ADD ON **$439**

Apple Cards and Hardware

16K Ram Card by CCI	130
Language System w/Pascal	379
Silentype Printer w/Interface Card	349
Hayes Micromodem II	299
Novation Apple-Cat II	339
Videx Videoterm 80 Column w/Graphics	269
Z-80 Softcard by Microsoft	299
16K RamCard by Microsoft	159
ABT Numeric Keypad (old or new kybrd)	110
ALF 3 Voice Music Card	239
ALF 9 Voice Music Card	169
Lazer Lower Case Plus +	55
Micro-Sci Disk Drives (A-40 & A-70)	CALL
SSM AIO Serial/Parallel Card A&T	189
Sup-R-Terminal 80 Column Card	329
SVA ZVX4 Megabyter 8" Disk Controller	589
SVA 2+2 Single Den. 8" Disk Controller	345
ThunderClock Plus	119
Symtec Hi-Res Light Pen	210
Integer or Applesoft Firmware Card	145
Graphics Tablet	619
Parallel Printer Interface Card	135
Hi-Speed Serial Interface Card	135
Smarterm 80 Column Card	299
Joystick by Keyboard Co.	45
Music System (16 Voices)	479
A/D + D/A Interface	289
Expansion Chassis	599
Introl/X-10 Controller Card	169
Clock/Calendar Card	225
CPS Multi-function Card	189
Supertalker SD-200	239
Romplus + Card	135
Romwriter Card	149
Clock/Calendar Module	99
GPIB IEEE-488 Card	249
Asynchronous Serial Interface Card	129
Centronics Parallel Interface Card	99
Arithmetic Processor Card	299

We carry all CCS cards, please call for best prices.

Software for the Apple

Visicalc 3.3	169
CCA Data Management	85
DB Master	169
WordStar (Apple 80 col. version)	299
Applewriter	65
Easywriter	225
Peachtree Business Software	CALL
Visiterm	129
Visitrend/Visiplot	219
Real Estate Analyzer	89
Tax Preparer	99
DOS Toolkit	65
Tax Planner	99
Dow Jones Portfolio Evaluator	45
Dow Jones News & Quotes Reporter	85
Apple Fortran	165
Controller Gen. Bus. System	499

Printers

Epson MX-80 CALL

Anadex 9500/9501 w/2K Buffer	1299
C. Itoh Starwriter 25 CPS	1499
C. Itho Starwriter 45 CPS	1899
Epson MX-70	CALL
Epson MX-80 F/T	CALL
Paper Tiger IDS-445	699
Paper Tiger IDS-460	949
Paper Tiger IDS-560	1249
Silentype w/Apple II interface card	349
Qume Sprint 5/45	2495

TOLL FREE ORDER LINE

800-854-6654

California and outside Continental US

(714) 698-8088

TELEX 695000 BETA CCMO

Video Monitors

Amdek/Leedex Video 100 12" B&W	139
Amdek (Hitachi) 13" Color	359
NEC 12" Green Phospher Screen	CALL
NEC 12" RGB Hi-Res Color	CALL
Panasonic 13" Color	449
Sanyo 9" B&W	185
Sanyo 9" Green Phospher Screen	CALL
Sanyo 12" B&W	269
Sanyo 12" Green Phospher Screen	285
Sanyo 13" Color w/excellent resolution	449

Atari Personal Computer

ATARI 800 16K $749

Atari 400 16K	349
410 Program Recorder	65
810 Disk Drive	449
815 Dual Disk Drive (Dbl. Den.)	1295
822 Thermal Printer	349
850 Interface Module	159
16K Ram Memory Module	149
16K Ram Memory Module (by ConComp)	89

NEC Microcomputer

PC-8001A 32K Ram Computer	CALL
PC-8012A I/O Unit w/32K Ram	CALL
PC-8031A Dual Mini-Disk Drive Unit	CALL

Please write for more information about the NEC computer.

Ordering information: Phone orders using VISA, MASTERCARD, AMERICAN EXPRESS, DINER'S CLUB, CARTE BLANCHE, bank wire transfer, cashier's or certified check, money order, or personal check (allow ten days to clear). Unless prepaid with cash, please add 5% for shipping, handling and insurance (minimum 5.00). California residents add 6% sales tax. We accept CODs. OEM's, Institutions and corporations please send for a written quotation. All equipment is subject to price change and availability without notice. All equipment is new and complete with manufacturer's warranty (usually 90 days). Showroom prices may differ from mail order prices.

Send Orders to:

consumer computers Mail Order

8314 Parkway Drive
La Mesa, California 92041

STOP

BEING OVERCHARGED!

DON'T
Pay Credit Card Surcharges
Pay a U.P.S. C.O.D. "Fee"
Even Pay for the Call!

DO CALL OMEGA TOLL FREE!

Special of the Month!

EPSON MX-80

INTERFACES:
IEEE $55, TRS-80 $35,
APPLE INTERFACE & CABLE $90,
RS-232 $70

ATARI 800 16K
$ 769

WEST COAST
1-800-235-3581

EAST COAST
1-800-556-7586

CALL OMEGA AND FIND OUT WHY WE ARE AMERICA'S FASTEST GROWING COMPUTER SALES COMPANY!

ΩMEGA SALES COMPANY

WEST COAST / EAST COAST

ANADEX $ 1249

ATARI 800 32K $ 749

DIABLO 630 $1995

Tractor option $ 245

NEC 12" MONITOR $ 229

NEC
INTERTEC
OKIDATA
APPLE
TELEVIDEO
COMMODORE
RADIO SHACK
AMDEK
QUME
ATARI

INTERTEC SUPERBRAIN 64K RAM	$2799
QD SUPERBRAIN	$2999
NEC 5510 SPINWRITER (7710)	$2345
NEC 5520 SPINWRITER (7720)	$2695
NEC 5530 SPINWRITER (7730)	$2345
NEC 12" MONITOR	$ 229
OKIDATA MICROLINE-80	$ 399
OKIDATA MICROLINE-82	$ 529
OKIDATA MICROLINE-83	$ 769
DIABLO 630	$1995
APPLE II PLUS 48K	$1139
APPLE DISK w/3.3 DOS Controller	$ 525
APPLE DISK w/o Controller	$ 449
HAZELTINE 1420	$ 799
NORTHSTAR HORIZON II 32K QD	$2925
ANADEX DP-9500/9501	$1249
TELEVIDEO 912C	$ 669
TELEVIDEO 920C	$ 729
TELEVIDEO 950	$ 929
CBM 8032 COMPUTER	$1149
CBM 8050 DISK DRIVE	$1349
CBM 4032 COMPUTER	$1029
CBM 4040 DISK DRIVE	$1029
CBM 4022	$ 649
CBM VIC-20	$ 269
RADIO SHACK II 64K	$3245
RADIO SHACK III 16K	$ 849
LEEDEX/AMDEK 100	$ 139
LEEDEX/AMDEK 100G	$ 169
LEEDEX/AMDEK COLOR-1 13" Color Monitor	$ 329
MICROTEK 16K RAMBOARD for Atari	$ 79
MICROTEK 32K	$ 149
QUME SPRINT 9/45 (Full Panel)	$2295
ATARI 400 16K	$ 349
ATARI 825 PRINTER	$ 650
ATARI 850 INTERFACE	$ 139
ATARI 810 DISK DRIVE	$ 449
ATARI 800	$ 749

Call for price list of ATARI software

PRICES ARE SUBJECT TO CHANGE WITHOUT NOTICE

We carry the complete line of Personal Software

We Accept C.O.D.'s • Stock Shipments Same Day or Next
No Surcharge for Credit Cards • All Equipment Factory Fresh with MFT Warranty

WEST COAST
1-800-235-3581

OMEGA SALES CO.
3533 Old Conejo Rd. #102
Newbury Park, CA 91320
1-805-499-3678
CA. TOLL FREE 1-800-322-1873

ΩMEGA SALES CO.

EAST COAST
1-800-556-7586

OMEGA SALES CO.
12 Meeting St.
Cumberland, RI 02864
1-401-722-1027

ΩMEGA SALES COMPANY

Advertiser's Index

The 6502 Resource Magazine

COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service, Call Our **Toll-Free** US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**________
☐ **Don't yet have one...**

☐ Please enter my 1 year (12 issue) subscription to **COMPUTE!**
☐ New subscription ☐ Renewal subscription
☐ $20.00 US subscription
☐ $25.00 Canada and international surface mail. Payment in US funds, drawn on US bank or money order.
☐ Bill me (US only). $1.00 billing fee

Name, Address: ______________________

____

____

____

Charge my: ☐ Visa ☐ MC
Number______________________ Expires______/______/

10 11 12

The Editor's Feedback:

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**________
☐ **Don't yet have one...**

During the next year I expect to buy:
☐ **computer** ☐ **printer**
☐ **disk drive** ☐ **other peripherals**

Content:
Best Article This Issue (page #, title)

____

Other suggestions:

10 11 12

The 6502 Resource Magazine

COMPUTE! PET•ATARI•APPLE OSI•KIM•SYM•AIM

For Fastest Service, Call Our **Toll-Free** US Order Line
800-345-8112
In Pennsylvania call 800-662-2444

My computer is:
☐ **PET** ☐ **APPLE** ☐ **ATARI** ☐ **OSI**
☐ **KIM** ☐ **SYM** ☐ **AIM** ☐ **OTHER**________
☐ **Don't yet have one...**

☐ Please enter my 1 year (12 issue) subscription to **COMPUTE!**
☐ New subscription ☐ Renewal subscription
☐ $20.00 US subscription
☐ $25.00 Canada and international surface mail. Payment in US funds, drawn on US bank or money order.
☐ Bill me (US only). $1.00 billing fee

Name, Address: ______________________

____

____

____

Charge my: ☐ Visa ☐ MC
Number______________________ Expires______/______/

10 11 12

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Place
Stamp
Here

COMPUTE! Magazine
Post Office Box 5406
Greensboro, NC 27403

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

FREE NEECO CATALOG!

PLACE STAMP HERE

NEECO
NEW ENGLAND ELECTRONICS CO., INC.
679 Highland Avenue
Needham, MA 02194

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 236 BROOMALL, PA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPUTE! Magazine
515 Abbott Drive
Broomall, PA 19008

Author Feedback:

____

____

____

____

____

____

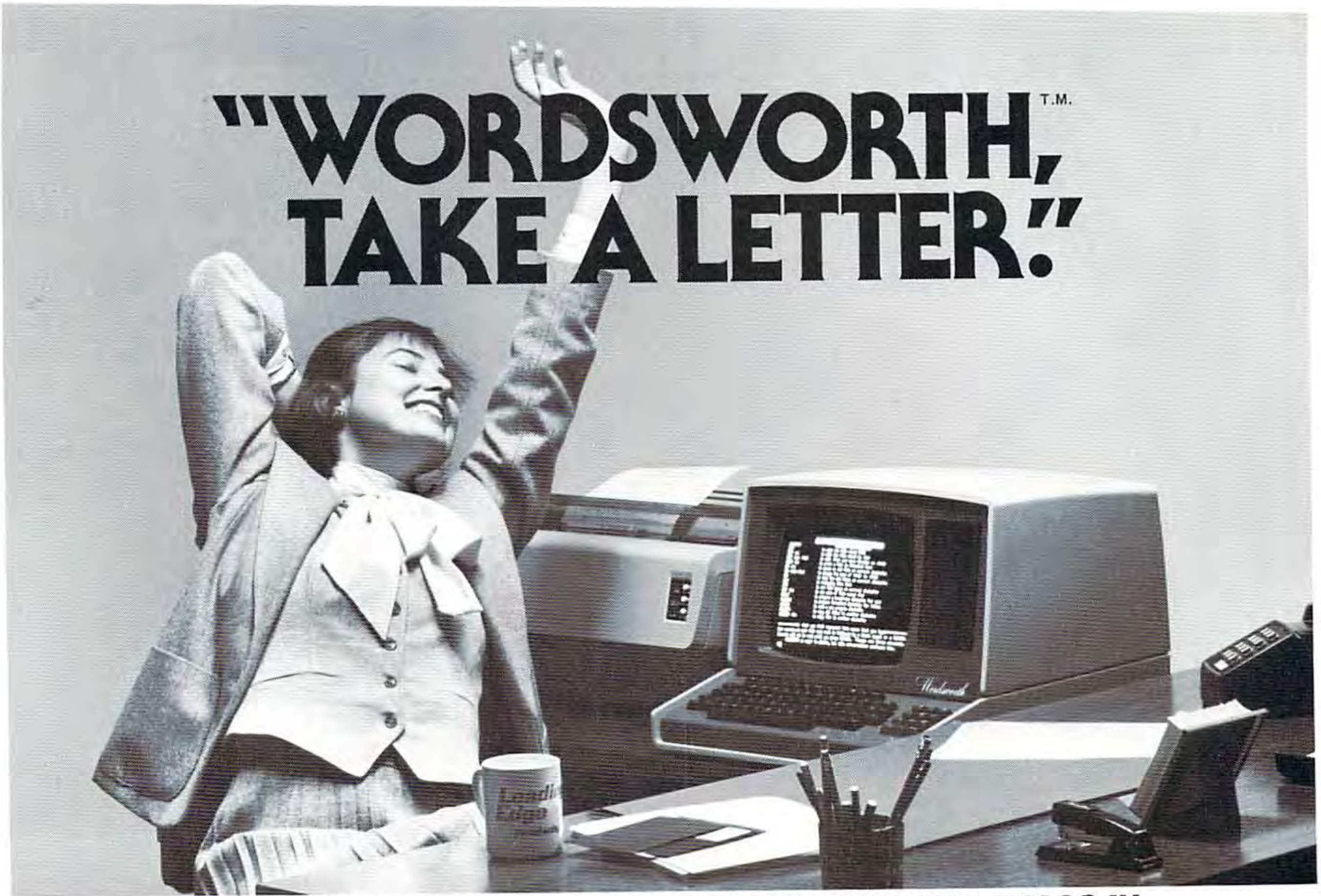
____

____

____

____

COMPUTE!



THE FIRST SUPER-SIMPLE LETTER QUALITY WORD PROCESSOR IN THE WORLD THAT CAN SIT NEXT TO YOUR SECRETARY— FOR LESS THAN $5,000.

"Wordsworth™" removes the fear and loathing many people have about things called "computers" and "word processing." Because Wordsworth just sits at a desk and does what he's told. With unique new, simplified software, it can tell its typist what to do—step by simple step—to perform even the most complicated tasks.

It can not only "take a letter," but it can revise it, customize it in millions of ways (quite literally), personalize it, print it on your letterhead or business form so you can't tell it from hand-typed . . . in short, everything but put it in an envelope.

Moreover, it can perform all sorts of mundane business functions, like general ledgers and accounting reports, using easily available industry-compatible CP/M™ software.

Installation? Plug it in. (Cable supplied.)

Size? Bigger than a breadbox—but not by much. It's about half as big, or less, than systems that can't do half as much. Definitely desktop—a sub-compact 20″ by 40″.

Service? Available everywhere.

Price? About $4,995. Complete and ready to go.

For the name of your nearest dealer—and a free hands-on demonstration—just pick up the nearest telephone.

TOLL-FREE 1-800-343-6833.

In Massachusetts, call collect (617) 828-8150. Telex 951-624.

LEADING EDGE.™

Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021

DEALERS: For immediate delivery from the Leading Edge Inventory Bank,™ on "Wordsworth" and other of the industry's most popular products, just give us a call.

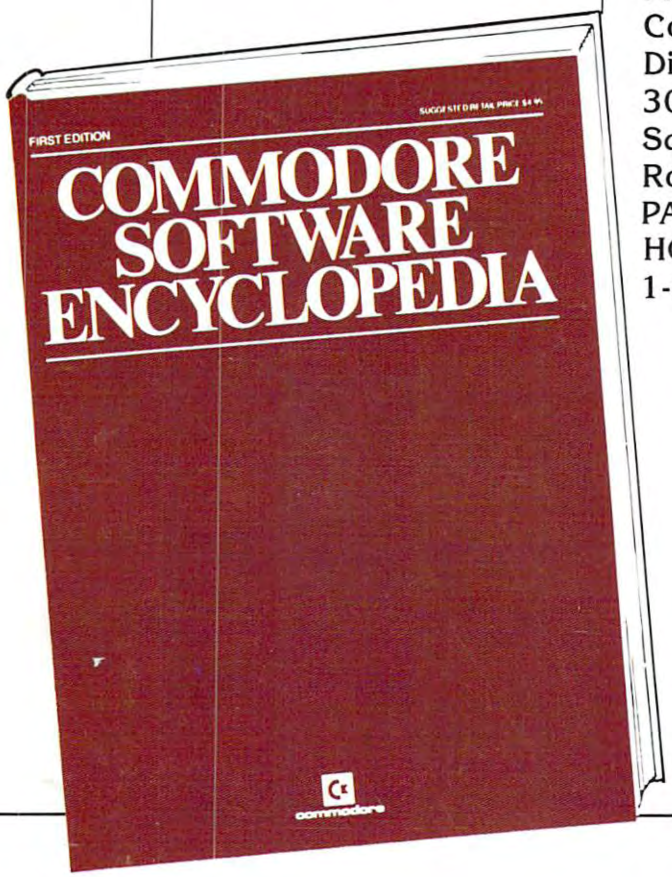
Commodore Software Encyclopedia

The COMMODORE SOFTWARE ENCYCLOPEDIA is now available from your authorized COMMODORE Dealer, for . . .

$4.95

The next edition of the COMMODORE SOFTWARE ENCYCLOPEDIA will be available in 90 to 120 days. If your software is not listed and you would like to have it listed in the next edition, please submit details to:

Software Department
Commodore Business Machines, Inc.
Computer Systems Division
300 Valley Forge Square, 681 Moore Road, King of Prussia, PA 19406.
HOTLINE Number
1-800-523-5622



commodore