the categories up to ten numbers can be entered — five for the taxpayer and five for the spouse. The program is smart enough to fill up all of the alternatives with the value given for the first alternative unless it is specifically given new values for subsequent alternatives. To move from one specific alternative to another "U" is used to go "up" and "D" is used to go "down." No control keys are required here — the editing is very easy to use.

ESCape can be used at nearly any time to exit from data entry and save on disk all of the values that have been entered for all of the alternative schemes. One minor irritant here is that the Pascal volume numbers are used to specify the disk drives. The documentation explains that disk drive #1 needs to be specified as volume four and so on, but the program should have been written to accept simple drive numbers. The name that is given for the file is first checked against those currently on the disk in order to prevent inadvertent over writing of a file that should be maintained. An option is also provided to see the directory of items that have been stored.

There are no charts included to indicate which of the 74 possible tax input questions are to be entered if, for example one were filing "married with a joint return." A glossary of terms would also be a welcome addition. However, execution speed is an outstanding feature of this program. All calculations are performed in under 60 seconds, regardless of complexity. The program appears to be written entirely in machine code, which would account for its exceptional speed.

While the ranges of input data appear to be sufficiently checked, disk error codes are vaguely defined. If RESET is pressed, all existing data not saved on disk is lost and the program requires rebooting for continued operation. This can be most annoying and could possibly prove fatal if done during a disk storage operation.

The Documentation

Documentation for the individual tax plan program consists of an attractive 3-ring binder with a 31 page illustrated instruction manual which includes a simple appendix and printouts. The documentation, although sufficient for the tax professional, is not designed to be a comprehensive overview of tax preparation for the layman.

With the exception of the misnumbered categories, the documentation is clear and complete. About ten pages are used to lead the user through two simple examples that do a good job of demonstrating how to move about in the program. Sample printed output for each of these examples is given in an Appendix (misidentified as Appendix "B"). About ten more pages are used to specifically describe the program options and to identify the various categories for data entry. Throughout the instructions, 27 photographs of screen images appear. The photographs were apparently taken with a wide angle lens and therefore appear distorted, but they are readable and provide an accurate representation of what the program displays.

General Overview

Panelist #1: "Negative and detracting hindrances:

(1) There should be a subroutine within the program which would enable the user to enter directly into a mini-directory to review a directive or procedure.

(2) The ability to only do the filing status routine should be looped so that only an individual taxpayer entry is verified and utilized when there is no spouse involved.

(3) Provision to exercise the use of only one disk drive should be available when only one is involved or necessary.

(4) An ending directive within the program (other than in the manual) should be provided after all statistics have been entered.

(5) A 'short form' alternative option could be incorporated.

Positive and useful aspects:

(1) Exceedingly fast access time.

(2) Ease of use in the main menu parameters.

(3) Printer parameters and linefeed status changes.

(4) Aardvark's updating procedures)annually or when the tax structure/laws change)

(5) Comparative analysis of defined numerical statistics to take advantage of the lowest tax amount to be paid.

(6) The 'step' feature: accessing forward and backward through the program via a single keystroke.

(7) Ability to access any part of the program by entering the input of the area and return.

(8) User defined changes: save data (Y/N), screen or printer display, program user return (ability to re-enter your numerical statistics and make any changes necessary in any of the alternative figures prior to executing the calculations).

(9) Ability to handle positive and negative integers as well as figuring out its compound percent.

(10) User ability to make any and all necessary backup copies in the event of catastrophes."

Panelist #2: "The software is easy to use and effectively compares calculations done on the basis of different tax preparation schemes. It does not do all of the calculations that a taxpayer needs to do, nor does it identify a correspondence between specific line numbers on form 1040 and the categories within the program. The software package could be very useful for professional tax preparers, but is not likely to be worth the expense for an ordinary taxpayer. For someone with substantial capital gains to declare, it could be helpful, but that person is probably going to benefit from advice from a professional anyway. Whom should you select as that professional? Someone who has an Apple and Aardvark's Individual Tax Plan."

Panelist #3: "This program was designed by a group of CPA's with over 17 years of "Big Eight" experience to meet the needs of the professional tax practitioner.

This program is not, nor was it designed to be, everyone's answer to H & R Block. With some additional documentation, a much wider range of people could benefit from it. While not intended for the layman, the professional tax preparer should find this program an outstanding value."

Sample Output

Table 1.

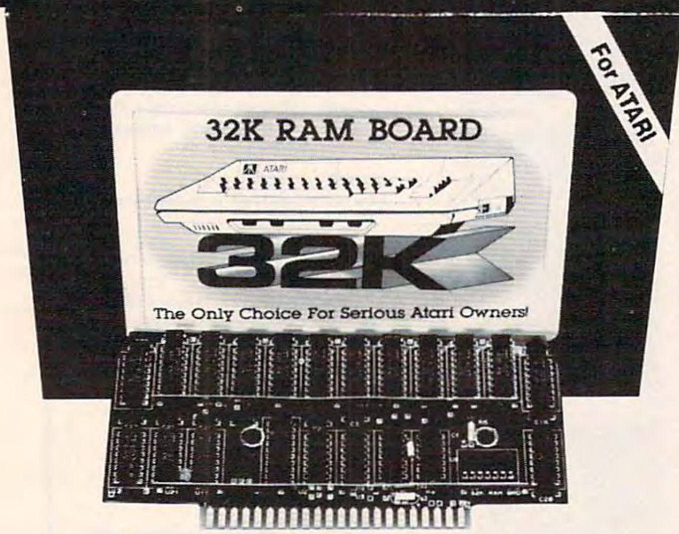
```
1981                              ALTERNATIVE
                                        1
FILING STATUS                        JOINT
EXEMPTIONS                               2
WAGES, SALARIES                     28,480
INTEREST AFTER EXCLUSION               350
DIVIDENDS AFTER EXCLUSION                0
CAPITAL GAIN/LOSS                        0
PARTNERSHIP INCOME/LOSS                  0
OTHER INCOME/LOSS                    2,000
                                  --------
TOTAL INCOME                        30,830
ADJUSTMENTS TO INCOME                1,600
                                  --------
ADJUSTED GROSS INCOME               29,230

DEDUCTIONS
MEDICAL & DENTAL EXPENSES              170
STATE & LOCAL INC TAXES              1,681
OTHER TAXES                              0
INTEREST EXPENSE                     1,690
CHARITABLE CONTRIBUTIONS               943
CASUALTY LOSS                        1,090
MISCELLANEOUS                          787
                                  --------
TOTAL DEDUCTIONS                     6,361
ZERO BRACKET AMOUNT                  3,400
                                  --------
EXCESS ITEM. DEDUCTIONS              2,961
```

TAX TABLE INCOME	26,269
EXEMPTIONS TIMES $1,000	** N/A
TAXABLE INCOME	** N/A
TAX - TAX TABLES/XYZ	4,359
TAX - QUAL. CAP. GAINS	** N/A
TAX - INCOME AVERAGING	*
TAX - MAXIMUM TAX	** N/A
TAX SELECTED	4,359
ADDITIONAL TAXES	0
GROSS REGULAR TAX	4,359
CREDITS	0
NET REGULAR TAX	4,359
MINIMUM TAX	0
ALTERNATIVE MINIMUM TAX	0
OTHER TAXES	0
TOTAL TAX LIABILITY	4,359
FEDERAL PAYMENTS	4,998
BALANCE DUE (REFUND)	-639

Individual Tax Plan. Aardvark Software, Inc., 783 North Water Street, Milwaukee, WI 53202. 48K Apple, two disk drives, DOS 3.3 or Pascal, $250. ©

THE MOSAIC 32K RAM FOR ATARI

THE BEST:



THE MOSAIC ADVANTAGE	MOSAIC 32K RAM	OTHER 32K RAMS
Works in both Atari 400 & 800	■	■
Gold edge connectors for better reliability	■	■
Fits Atari 400 without modification	■	■
Custom components for better performance & reliability	■	
Highest quality components for the best screen clarity	■	
Full year warranty	■	
Designed to take advantage of Atari 800's superior bus structure.	■	
Can be used with 8K, 16K and future products.	■	
Allows Atari 800 to have 2 slots for future expansion	■	
Designed so there's no danger of damaging your computer	■	
Designed for inter-board communication in Atari 800	■	
Easy to follow instructions for simpler no-solder installation in Atari 400	■	
Available companion board ($5) to allow running 32K board independent of other boards	■	
Full flexible memory configuration	■	

*Trade Mark of Atari, Inc.

✓ DESIGN
✓ MATERIALS
✓ SAFETY
✓ GUARANTEE

THE ONLY CHOICE

You own the best micro computer available, the Atari*. At Mosaic we've made Atari computers our only business and have made our products the best anywhere. You've seen the advantage of having a 32K RAM board. The Mosaic 32K RAM is the only board designed to meet your needs now and in the future too. It has designed-in flexibility to be compatible with products available in the near future. See for yourself, Mosaic is the only choice for the serious Atari owner.

THE BEST SCREEN CLARITY

Here's what A.N.A.L.O.G. magazine had to say: "Mosaic uses, what we feel are the highest quality components, which may improve reliability." and "The Mosaic showed no sign of interference and gave the best screen clarity."

Atari 800 Memory Configuration	with the MOSAIC 32K RAM	with other 32K Boards
Empty / 32K / 16K	48K RAM 40K With BASIC Cartridge	48K RAM 40K With BASIC Cartridge
Empty / 32K / 8K	40K RAM	**Danger!** This Configuration Can Damage Computer
Empty / 16K / 32K	48K RAM 40K With BASIC Cartridge	**Danger!** This Configuration Can Damage Computer

Atari 800 Memory Configuration	with the MOSAIC 32K RAM	with other 32K Boards
Empty / 8K / 32K	40K RAM	**Danger!** This Configuration Can Damage Computer
8K / 32K / 8K	48K RAM 40K With BASIC Cartridge	**Danger!** This Configuration Can Damage Computer
Empty / Companion / 32K	32K RAM	**Danger!** This Configuration Can Damage Computer

Now from your nearest Mosaic dealer **$179.95**

Direct from Mosaic $189.95

MOSAIC ELECTRONICS

P.O. Box 748 Oregon City, Oregon 97045 503/655-9574

THE ATARI® GAZETTE

Cryptogram

Ronald and Lynn Marcuse
Freehold, NJ

Word games are just one of many applications which can be programmed on the versatile ATARI computers. While not challenging your reflexes as does Space Invaders or Star Raiders, they do challenge your mind. With the number of graphic action games approaching infinity in our house, we predicted that our two school-age offspring will be competent space shuttle pilots by the time they reach 18. Unfortunately, they may not be able to read. An educational computer word game may be capable of swinging the pendulum the other way, or at least slow down the onrushing invaders.

Most of the electronic word games currently being marketed are variations of the "hangman" game, where you are required to guess an unknown word by specifying its letter content within a certain number of tries. If you fail, you are then "hanged" or punished in some similar manner. A *cryptogram*, however, challenges your ability to decipher coded phrases or messages. Not wanting to generate an "unbreakable" code, we used a simple letter substitution. Each letter in the statement is replaced uniformly by some other alphabetic character. For example, all of the A's may appear as G's, the B's as Y's, etc. Don't bother to memorize these relationships because the structure of the code changes each game. Spaces between words and punctuation remain intact.

There are two skill levels in the game, selectable through the OPTION key. With the first, the program will decipher the vowels for you, leaving you only the consonents to decode. With the second option you are on your own. We had originally programmed three options, the third being a compression of the statement into one long string (removal of spaces), but deciphering of the phrase became rather difficult. The SELECT key is utilized to vary the number of participants. For the two person version, one player would enter a statement for the other to decode. In the one player game, the computer will randomly select one of fifty popular (?) expressions stored in the program. These phrases appear as the DATA statements in lines 1010-1500. You may change this list if you desire. Just make sure that you wind up with exactly 50 statements and that each one is no longer than 75 characters (including spaces and punctuation). You may use any punctuation with the exception of the comma.

How To Play

A game in progress may be saved to either disk or tape. Program 1 contains the disk version. Program 2 displays changes required for the recorder. In saving the game, the disk version will request a three character (or less) name which will be appended to the file name. The tape version will require you to insert a blank tape in the recorder. Make a note of the recorder counter. The procedure for loading a saved game is similar, but you must supply either the file name extender or tape that has been positioned (using the counter) in the recorder.

To start a game type "N" to the program prompt "SHOULD I LOAD A SAVED GAME" and then pick the skill level and number of players by pressing the OPTION and SELECT keys. Press the START key to begin. You may need to depress the keys for a second or two to register your action. This may be speeded up by shortening the timing loop at line 990. If you had chosen the two player option, the program will prompt you to enter a phrase or message to be encoded. This must be from 20 to 75 characters in length. Shorter phrases are actually harder to decipher than long ones. In the one player option, the program will randomly select one of the fifty DATA statements.

After the encoding process is completed (it takes approximately 15 seconds to generate the code and substitute the letters), the game screen is displayed. At the top of the display is a table showing the code letters and values that you have assigned to the code. The next group of lines contain the "secret phrase" and your working translation. These alternate if the phrase is longer than one line in length. If skill level 1 had been selected, the vowels would have already been translated for you. At the bottom of the display is the input area for code letters and values. Enter a code letter and then the substitution you would like to perform. An arrow cursor alternates between the two input lines. To erase a previous entry, first type the code letter and then press the space bar.

When you have correctly substituted all of the characters, the program will notify you graphically. You may also press the ESCAPE key to end the game. This will allow you to save the game, quit, or try a new phrase. If you are short on RAM (under 24K), the REMARK statements may be omitted with no ill effects.

Lines 18 through 30 comprise the "housekeeping" section of the program. The left screen margin is set to 1 (POKE 82,1) for those TV sets that overscan, the keyboard is OPENed and the variables are DIMensioned here. The alphabet is stored in A$, the substitution code in B$, and the table entries for the game display in T$, P$, C$, and Q$ are the actual phrase, the coded phrase, and the working translation, respectively. The array X (with 26 elements) is used by the code generation routine.

If a saved game is being reloaded (prompt in line 40), the data is input and control is sent directly to the main game display at line 400. Otherwise, the variables are cleared (lines 80-90) and the option screen is generated (starting at line 100). Memory location 53279 is the register used to read the console keys on the ATARI computer. The address is first cleared by POKE 53279,8 and then queried by PEEKing at it in the loop from lines 120 through 180. We are concerned with the binary value that is stored in that address.

The START key is assigned to bit 0, the SELECT key to bit 1, and the OPTION key to bit 3. A value of zero in the bit position means that the key was pressed. For example, if the START key is hit, the SELECT and OPTION keys would register decimal values of 2 and 4 in their respective bit positions. The START key would return a zero in the low order bit, giving a total of 6 (decimal). Likewise, the SELECT key would equal a decimal 5 (4 + 1) and the OPTION key would be 3 (2 + 1).

If the two player option was selected, the phrase would be input in line 220, otherwise the program will randomly select one of the fifty data statements in line 240. In lines 250 through 290, the program generates the substitution code. A random letter (from A$) is selected and, if that element of the X array is still set to zero, the B$ sub-string position is equated to the letter. The array is used to check off letters that have already been used. This type of algorithm could easily be expanded to a card shuffling routine if you prefer poker to word games.

The substitution of the code letters into the phrase is done in lines 300 to 380. If skill level 1 was selected (SK = 1) then the ATASCII value of the phrase letter is checked to see if it is a vowel (values of 1,5,9,15, and 21). If it is, the letter is moved into the translation line Q$, otherwise the character "–" occupies that position. The program must also count backwards from 38 looking for the first space to break the line on.

The game board is displayed in lines 400-430 and the input of code letters and substitutions is performed in lines 500-520. After the data is received, the modified table elements are redisplayed in line 530 and the revised translation line in 540-560. If the translation is the same as the phrase (line 560), you are sent to the winners circle at line 700, otherwise you go back to 510 for more data. Pressing the ESCAPE key (an ATASCII value of 27) would cause a jump to line 800 for your exit options. The POP statement in line 915 is necessary to reset the stack pointer for the non-RETURN exit out of the subroutine.

The remainder of the program is routines for the winning and losing displays, input and printing of data, the exit options, and the saving of games in progress. The variables saved, either on disk or tape, are P$, C$, and Q$ (the original phrase, the coded phrase the the current translation), T$ (the assignment table), and the lengths of the phrase (L) and its first line segment (L1). For the disk version of the program, the format of the saved game is D:CRYPTG. + the 3 character name that was entered.

Program 1.

```
10 REM ********    CRYPTO-GRAM    ******
*******
11 REM *
      *
12 REM *  RONALD & LYNN MARCUSE, FREEHOL
D NJ  *
13 REM *
      *
14 REM *********************************
*******
15 REM
18 POKE 82,1:OPEN #4,4,0,"K:"
20 DIM A$(26),B$(26),T$(26),I$(1),D$(1),
P$(81),C$(80),Q$(80),X(26)
25 DIM N$(3),F$(12)
30 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":GOSUB
 900:R=40
36 REM
37 REM LINES 40 THRU 70 ALLOW THE LOADIN
G OF A SAVED GAME ON DISK
38 REM SEE LISTING 2 FOR TAPE VERSION
39 REM
40 ? "{DOWN} SHOULD I LOAD A SAVED GAME
(Y/N) ";
45 GOSUB 970:IF I$<>"Y" THEN 80
50 GOSUB 960:TRAP 70:OPEN #2,4,0,F$:INPU
T #2,P$
```

Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky

Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software

ATARI (tm) GRAPHICS AND SOUND MADE EASY!

SANTA CRUZ EDUCATIONAL SOFTWARE HAS WRITTEN A SERIES OF AFFORDABLE PROGRAMS THAT DEMONSTRATES MANY OF THE SPECIAL "TRICKS" THAT ONLY THE ATARI 400/800 COMPUTER IS CAPABLE OF DOING. WE OFFER EVERYTHING FROM A PROGRAM THAT DOES FANTASTIC HIGH RESOLUTION GRAPHICS TO ONE THAT ACTUALLY DIALS YOUR PHONE.....

★★★

MASTER MEMORY MAP(tm) — This is really the key to using the ATARI'S capabilities. We start out by explaining how to PEEK and POKE values into memory so that even new programmers can use this. Then, we give you over 15 pages of the memory locations that are the most useful. The information is condensed from both the ATARI'S Operating System Manual and various articles and programs. It is, of course, useful even for experienced programmers as a reference. Also, we highly suggest that dealers offer this Memory Map to customers who request to be told how to use the power of the machine. We guarantee it will answer many of the questions you have about the machine. **$5.95**

TRICKY TUTORIALS(tm)

#1: DISPLAY LISTS — This program teaches you how to alter the program in the ATARI that controls the format of the screen. For example: when you say graphics 8 the machine responds with a large graphics 8 area at the top of the screen and a small text area at the bottom. Now, you will be able to mix the various modes on the screen at the same time. Just think how nice your programs could look with a mix of large and small text, and both high and low resolution graphics. this program has many examples plus does all of the difficult caluculations! **$14.95**

#2: HORIZONTAL/VERITCAL SCROLLING — The information you put on the screen, either graphics or text, can be moved up, down or sideways. This can make some nice effects. You could move only the text on the bottom half of the screen or perhaps create a map and then move smoothly over it by using the joystick. **$14.95**

#3: PAGE FLIPPING — Normally you have to redraw the screen every time you change the picture or text. Now you can learn how to have the computer draw the next page you want to see while you are still looking at the previous page, then flip to it instantly. You won't see it being drawn, so a complicated picture can seem to just appear. Depending on your memory size and how complicated the picture, you could flip between many pages, thus allowing animation or other special effects with your text. **$14.95**

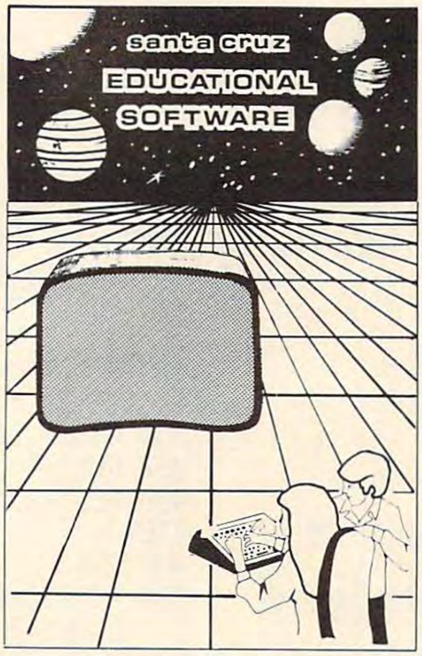
#4: BASICS OF ANIMATION — Shows you how to animate simple shapes using the PRINT and PLOT commands, and also has nice little PLAYER/MISSILE Graphics demo to learn. This would be an excellent way to start making your programs come alive on the screen. Recommended for new owners. **$14.95**

#5 PLAYER MISSILE GRAPHICS — This complex subject is demonstrated by starting with simple examples, and building up to a complete game and also an animated business chart on multiple pages! As always, the computer does most of the calculations. Requires 32K disk or tape and costs **$29.95**

#6: SOUND — From explaining how to create single notes, to demonstrating complex four channel sound effects, this newest tutorial is great. Even those experienced with ATARI's sound capabilities will find the menu of sound effects a needed reference that can be used whenever you are in the need of a special sound for your programs. Everyone will learn something new! Written by Jerry White. **$14.95**

Tricky Tutorials (except #5) require 16K memory for cassette orders and 24K for disk. The price is $14.95 each. You may order 1,2,3, & 4 for $49.95! All six in a colorful binder cost $89.95.

THE GRAPHICS MACHINE!! — Turn your computer into an incredible graphics tool with advanced commands like circle, box, fill, polygon, line, help, etc. 3 colors in graphics 8 with instant text!!! Create colorful business charts or beautiful drawings and then save or retrieve them from disk in 5 SECONDS. YES, it's that fast. Needs all 48K, disk, and costs **$19.95**



MINI-WORD PROCESSOR — This is for those of you who have a printer, but don't want to spend $100 or more for a fancy word processor. It is suitable for simple editing of text, accepts most control characters for your printer, and text is stored on disk for easy retrieval. Holds 2½ typed pages at a time. Requires 32K, disk or tape. **$14.95**

BOB'S BUSINESS — 14 small business type programs for home or office, all chosen from a nice menu. Supports printed output. 169 sectors of output require 16K tape, or 32K disk. **$14.95**

KID'S #1 — Includes the following: 1) TREASURE — search for the lost treasure while trying to keep from falling into the sea. Nice graphics if you find it!; 2)DIALOGUE — talk back to the computer about four subjects; 3) MATH QUIZ — Nice musical and graphical rewards for good scores. Parents input the level of difficulty.

KID'S #2 — A spelling quiz, a "scrabble" type game, and a version of Touch with the computer giving all the directions! Both Kid's programs require 16K tape or 24K disk and cost **$14.95** each.

MINI-DATABASE/DIALER — This unique new program stores and edits up to 8 lines of information such as name, address, and phone numbers, or messages, inventories or anything you want. It has the usual sort, search, and print options, but it also has an unusual feature: If your files include phone numbers and you have a touch-tone phone, the program will DIAL THE PHONE NUMBERS FOR YOU! This is perfect for those who make a lot of calls like salesmen, teens, or those trying to get through to busy numbers (acts as an auto-redialer). It is also a lot of fun to use. Requires 16K cassette or 24K disk and costs **$24.95**

FONETONE — For those who only want to store name and phone numbers and have the dialer feature as above, we offer this reduced version. Same memory requirements, but only costs **$14.95.** Don't forget you must have a touch-tone phone.

PLAYER PIANO — Turns your keyboard into a mini-piano and more. Multiple menu options provide the ability to create your own songs, save or load data files using cassette or diskette, fix or change any of up to 400 notes in memory, and play all or part of a song. The screen displays the keyboard and indicates each key as it is played from a data file or the notes you type. You don't have to be a musician to enjoy this educational and entertaining program. Requires 24K cassette or 32K disk. **$14.95**

BOWLERS DATABASE — Provides the league bowler with the ability to record and retrieve bowling scores providing permanent records. The data may then be analyzed by the program and displayed or printed in summary or detail form. Data may be stored on cassette or diskette and updated quickly and efficiently. The program proivdes such information as highest and lowest scores by individual game, (first, second, and third games throughout the season), high and low series, current average, and more. The program listing and documentation provided are a tutorial on ATARI basic and record keeeping. Requires 16K for cassette or 24K for disk. **$14.95**

By the time you read this all computers (400/800) being produced should have the fabled GTIA chips included. ATARI service may upgrade older computers...call and ask (it's easy to do yourself). We have one and the improvements that graphics modes 9,10, and 11 offer are great!! To help you figure out what to do with the new modes a new Tricky Tutorial will be offered in March on Modes 9 to 11. Either give us a call or write around that time.

★★★

SEE YOUR LOCAL DEALER OR ORDER FROM:

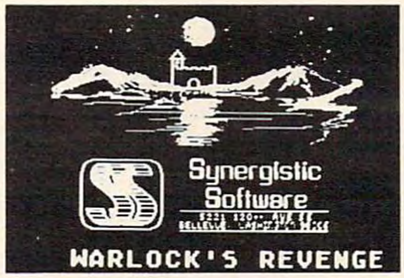
SANTA CRUZ EDUCATIONAL SOFTWARE
5425 Jigger Dr., Soquel, CA 95073
C.O.D. Orders call (408) 476-4901
*Include $2.00 postage ($.75 for Memory Map)
*In California, include 6.5% tax

Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky *

Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutorials * Santa Cruz Educational Software * Tricky Tutori

```
60 INPUT #2,C$:INPUT #2,Q$:INPUT #2,T$:I
NPUT #2,L:INPUT #2,L1
65 CLOSE #2:XIO 33,#2,0,0,F$:TRAP 40000:
GOTO 400
70 CLOSE #2:? "{TAB}!DISK ERROR!":GOSUB
990:GOTO R
80 FOR I=1 TO 26:X(I)=0:T$(I)="_":NEXT I
:PL=1:SK=1:RESTORE
90 FOR I=1 TO 80 STEP 10:C$(I)="
  ":Q$(I)="          ":NEXT I
97 REM
98 REM OPTION SELECTION MENU
99 REM
100 GRAPHICS 17:SETCOLOR 0,3,10:SETCOLOR
 4,3,2
105 POSITION 4,2:? #6;"CRYPTO-GRAM":POKE
 53279,8
110 POSITION 3,5:? #6;"* * * * * * *"
120 POSITION 2,11:? #6;"SKILL LEVEL - ";
SK
130 POSITION 1,14:? #6;"# OF PLAYERS - "
;PL
140 POSITION 4,20:? #6;"PRESS START":? #
6;"     TO BEGIN"
150 GOSUB 990:A=PEEK(53279):IF A=6 THEN
200
160 IF A=5 THEN PL=PL+1:IF PL>2 THEN PL=
1
170 IF A=3 THEN SK=SK+1:IF SK>2 THEN SK=
1
180 GOTO 120
200 GOSUB 900:ON PL GOTO 240,210
207 REM
208 REM TWO PLAYER OPTION
209 REM
210 ? "{2 DOWN} ENTER PHRASE (20 TO 75 C
HARACTERS){DOWN}"
220 INPUT P$:IF LEN(P$)<20 OR LEN(P$)>75
 THEN 210
230 GOTO 250
237 REM
238 REM ONE PLAYER OPTION, COMPUTER PICK
S RANDOM PHRASE
239 REM
240 J=INT(RND(0)*50)+1:FOR I=1 TO J:READ
 P$:NEXT I
247 REM
248 REM ALPHABETIC SUBSTITUTION CODE GEN
ERATED
249 REM
250 GOSUB 910:? "{2 DOWN}    PLEASE WAIT
WHILE I GENERATE"
260 ? "{DOWN}{TAB}AN ! UNBREAKABLE ! COD
E":FOR I=1 TO 26
270 J=INT(RND(0)*26)+1:IF X(J)=1 THEN 27
0
280 IF SK=1 THEN IF I=1 OR I=5 OR I=9 OR
 I=15 OR I=21 THEN T$(J,J)=A$(I,I)
290 B$(I,I)=A$(J,J):X(J)=1:NEXT I
297 REM
298 REM CHARACTERS IN PHRASE SUBSTITUTED
 WITH CODE LETTERS
299 REM
300 L=LEN(P$):FOR I=1 TO L:J=0:I$=P$(I,I
):IF I$=" " THEN 360
310 J=J+1:IF J>26 THEN C$(I)=I$:Q$(I)=I$
:GOTO 360
320 IF I$<>A$(J,J) THEN 310
340 C$(I)=B$(J,J):Q$(I)="_"
350 IF SK=1 THEN IF J=1 OR J=5 OR J=9 OR
 J=15 OR J=21 THEN Q$(I)=I$
360 NEXT I
365 L1=L:IF L<38 THEN 400
367 REM
368 REM FIRST LINE SPACING MEASURED
369 REM
370 FOR I=38 TO 18 STEP -1:IF C$(I,I)="
" THEN L1=I:GOTO 400
380 NEXT I
397 REM
398 REM MAIN GAME BOARD DISPLAYED
399 REM
400 GOSUB 910:POSITION 2,3:? "CD/LT":FOR
 N=1 TO 26:I$=A$(N,N)
410 POSITION 3+INT(N/5)*6,3+N-INT(N/5)*5
:? I$;" ";T$(N,N):NEXT N
420 POSITION 1,9:? C$(1,L1):IF L>L1 THEN
 POSITION 1,13:? C$(L1+1)
430 GOSUB 950
497 REM
498 REM PROMPTS FOR INPUT OF CODE AND LE
TTER
499 REM
500 POSITION 3,20:? "ENTER CODE LETTER":
? "{DOWN}{TAB}    AND VALUE"
510 K=20:GOSUB 920:N=A-64:D$=CHR$(A)
520 K=22:GOSUB 920:T$(N,N)=CHR$(A):IF A=
32 THEN T$(N,N)="_"
530 POSITION 5+INT(N/5)*6,3+N-INT(N/5)*5
:? T$(N,N)
540 FOR I=1 TO L:IF C$(I,I)=D$ THEN Q$(I
,I)=T$(N,N)
550 NEXT I
560 GOSUB 950:IF P$=Q$ THEN 700
590 POSITION 24,22:? " ":GOTO 510
697 REM
698 REM WINNER SCREEN DISPLAYED
699 REM
700 FOR J=0 TO 14 STEP 2:GRAPHICS 18:SET
COLOR 4,J,2:POSITION 3,5
710 ? #6;"* * * * * * *":? #6;"   *  COR
RECT  *":? #6;"   * * * * * * *"
```

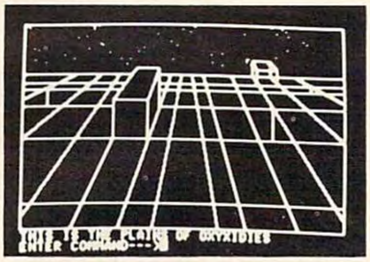
WARLOCK'S REVENGE



NEW FROM

Synergistic Software

AN EXCITING ATARI* HIGH-RES ADVENTURE GAME



Over 100 high resolution pictures in this two disk adventure, guide you to treasure as well as dangers. As you explore caverns and castles you use your various skills to obtain the riches therein, while eluding the pitfalls and creatures that abound. You provide the Atari, Warlock's Revenge provides the challenge. Requires at least a 40K, 400/800 with Atari Basic and disk. **$35.00**

SYNERGISTIC SOFTWARE
5221 120th Ave. S.E.
Bellevue, Washington 98006
(206) 226-3216

Available for the Apple II Computer as Oldorf's Revenge from Highlands Computer Services

ATARI* 400/800 are trademarks of Atari Computers Inc.

RAM

For ATARI

48K RAM BOARD FOR THE 400

- Increases memory capacity
- Reduces power consumption
- Reduces heat

48K Board (400) **$299**
32K Board (800) **$150**

INTEC PERIPHERALS CORP
3389 Del. Rosa Avenue
San Bernardino, CA 92404
(714) 864-5269

ATARI, 400, 800 are Trademarks of ATARI, Inc.

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

® Trademark of ATARI, Inc.

DIVISION OF CUSTOM ELECTRONICS, INC.
CE SOFTWARE
238 Exchange St., Chicopee, Massachusetts 01013
(413) 592-4761
Mastercard & VISA Accepted
• Dealer And Distributor Inquiries Invited
• Closed Mondays — Open Daily 'Til 5:30 — Fridays 'Til 8

```
720 FOR K=0 TO 250 STEP 10:SOUND 0,K,10,
15:NEXT K
730 GOSUB 900:GOSUB 950
740 FOR K=250 TO 0 STEP -10:SOUND 0,K,10
,15:NEXT K
750 NEXT J:SOUND 0,0,0,0
760 GOSUB 900:GOSUB 950:? "{2 DOWN}{TAB}
PLAY AGAIN (Y/N) ";
PLAY AGAIN (Y/N) ";
765 GOSUB 970:IF I$="Y" THEN 80
770 GOTO 880
797 REM
798 REM END OF GAME (NO WINNER) OPTIONS
799 REM
800 GOSUB 910:? "{DOWN}{TAB}OPTIONS ARE:
"
810 ? "{DOWN}{TAB}S - SAVE GAME":? "{DOW
N}{TAB}Q - QUIT":? "{DOWN}{TAB}P - PLAY
AGAIN"
820 ? "{DOWN}{TAB}SELECT => ";:GOSUB 970
:IF I$="S" THEN 850
830 IF I$="Q" THEN GOSUB 980:GOTO 880
840 IF I$="P" THEN GOSUB 980:GOTO 80
845 GOTO 800
847 REM
848 REM GAME IS SAVED TO DISK, SEE LISTI
NG 2 FOR TAPE VERSION
849 REM
850 R=800:GOSUB 960:TRAP 70:OPEN #2,8,0,
F$:TRAP 40000
870 ? #2;P$:? #2;C$:? #2;O$:? #2;T$:? #2
;L:? #2;L1:CLOSE #2
880 GRAPHICS 0:END
897 REM
898 REM PRINT CRYPTO-GRAM TITLE
899 REM
900 GRAPHICS 0:SETCOLOR 1,3,10:SETCOLOR
2,3,2:POKE 752,1
910 ? "{CLEAR}{DOWN}{2 TAB}{2 LEFT}CRYPT
O-GRAM":RETURN
917 REM
918 REM CHARACTER INPUT EDITING
919 REM
920 POSITION 21,K:? "==> {LEFT}";:GET #4
,A
925 IF A=27 THEN POP :GOTO 800
930 IF A=32 AND K=22 THEN 940
935 IF A<65 OR A>90 THEN 920
940 POSITION 21,K:? "   ";CHR$(A):RETURN

947 REM
948 REM PRINT ANSWER
949 REM
950 POSITION 1,11:? Q$(1,L1):IF L>L1 THE
N POSITION 1,15:? Q$(L1+1)
955 RETURN
957 REM
958 REM DISK FILE SAVED WITH PERSONS NAM
E (3 CHAR)
960 ? "{DOWN} ENTER 3 LETTERS OF NAME =>
 ";:INPUT N$:F$="D:CRYPTG."
965 F$(10)=N$:RETURN
967 REM
968 REM GET AND PRINT CHARACTERS
969 REM
970 GET #4,A:I$=CHR$(A):? I$:RETURN
977 REM
978 REM LOOSER DISPLAY SCREEN
979 REM
980 ? "{3 UP}{TAB}THE ANSWER WAS:":Q$=P$
:GOSUB 950:? "{2 DOWN}{TAB}! LOOK HOW EA
SY IT WAS !"
985 FOR I=10 TO 250:SOUND 0,I,10,I/20+2:
NEXT I:SOUND 0,0,0,0:RETURN
987 REM
988 REM DELAY LOOP
989 REM
990 FOR I=1 TO 100:NEXT I:RETURN
997 REM
998 REM THE STORED PHRASES FOLLOW, MAXIM
UM OF 50 ALLOWED
999 REM EACH MUST BE UNDER 75 CHARACTERS
 LONG
1000 REM
1010 DATA A STITCH IN TIME SAVES NINE
1020 DATA EARLY TO BED AND EARLY TO RISE
 MAKES A MAN HEALTHY WEALTHY AND WISE
1030 DATA THE EARLY BIRD CATCHES THE WOR
M
1040 DATA DO UNTO OTHERS AS YOU WOULD HA
VE OTHERS DO UNTO YOU
1050 DATA PLOP PLOP FIZZ FIZZ OH WHAT A
RELIEF IT IS
1060 DATA A LONG TIME AGO IN A GALAXY FA
R FAR AWAY
1070 DATA WHY DID THE CHICKEN CROSS THE
ROAD?
1080 DATA TO BE OR NOT TO BE. THAT IS TH
E QUESTION
1090 DATA THOU SHALT NOT COVET THY NEIGH
BORS WIFE
1100 DATA MAY THE FORCE BE WITH YOU
1110 DATA BEGINNERS ALL-PURPOSE SYMBOLIC
 INSTRUCTION CODE
1120 DATA WE THE PEOPLE OF THE UNITED ST
ATES OF AMERICA
1130 DATA DON'T FIRE UNTIL YOU SEE THE W
HITES OF THEIR EYES
1140 DATA YOU CAN FOOL SOME OF THE PEOPL
E ALL OF THE TIME
1150 DATA PROGRESS IS OUR MOST IMPORTANT
 PRODUCT
```

```
1160 DATA ONE SMALL STEP FOR MAN; ONE GI
ANT STEP FOR MANKIND
1170 DATA EVERY ACTION HAS AN EQUAL AND
OPPOSITE REACTION
1180 DATA I HAVE NOT YET BEGUN TO FIGHT
1190 DATA FASTER THAN A BULLET; MORE POW
ERFUL THAN A SPEEDING LOCOMOTIVE
1200 DATA WHO WAS THAT MASKED MAN
1210 DATA THEIRS NOT TO REASON WHY; THEI
RS BUT DO OR DIE
1220 DATA TO ERR IS HUMAN. TO FORGIVE DI
VINE
1230 DATA A LITTLE LEARNING IS A DANGERO
US THING
1240 DATA HE'D FLY THROUGH THE AIR WITH
THE GREATEST OF EASE
1250 DATA LAUGH AND THE WORLD LAUGHS WIT
H YOU; CRY AND YOU CRY ALONE
1260 DATA MARRIED IN HASTE; WE MAY REPEN
T AT LEISURE
1270 DATA O CAPTAIN! MY CAPTAIN! OUR FEA
RFUL TRIP IS DONE
1280 DATA THESE ARE THE TIMES THAT TRY M
EN'S SOULS
1290 DATA TIGER! TIGER! BURNING BRIGHT I
N THE FORESTS OF THE NIGHT
1300 DATA TALL OAKS FROM LITTLE ACORNS G
ROW
1310 DATA I THINK THAT I SHALL NEVER SEE
 A POEM LOVELY AS A TREE
1320 DATA FOR FOOLS RUSH IN WHERE ANGELS
 FEAR TO TREAD
1330 DATA WHEN IN ROME DO AS THE ROMANS
DO
1340 DATA FRIENDS ROMANS COUNTRYMEN LEND
 ME YOUR EARS
1350 DATA FROM THE HALLS OF MONTEZUMA TO
 THE SHORES OF TRIPOLI
1360 DATA IN FOURTEEN HUNDRED NINETY TWO
 COLUMBUS SAILED THE OCEAN BLUE
1370 DATA MARY HAD A LITTLE LAMB ITS FLE
ECE AS WHITE AS SNOW
1380 DATA I SHOT AN ARROW INTO THE AIR;
IT FELL TO EARTH I KNEW NOT WHERE
1390 DATA IF AT FIRST YOU DON'T SUCCEED
TRY TRY AGAIN
1400 DATA HE THAT FIGHTS AND RUNS AWAY M
AY TURN AND FIGHT ANOTHER DAY
1410 DATA IT TAKES A HEAP OF LIVING IN A
 HOUSE TO MAKE IT HOME
1420 DATA OH WHAT FUN IT IS TO RIDE IN A
 ONE HORSE OPEN SLEIGH
1430 DATA IT WAS A ONE-EYED ONE-HORNED F
LYING PURPLE PEOPLE EATER
1440 DATA I CAN'T BELIEVE I ATE THE WHOL
E THING
1450 DATA SAY THE SECRET WORD AND WIN A
HUNDRED DOLLARS
1460 DATA DAMN THE TORPEDOES; FULL SPEED
 AHEAD
1470 DATA TWAS THE NIGHT BEFORE CHRISTMA
S AND ALL THROUGH THE HOUSE
1480 DATA HEY MISTER TAMBOURINE MAN PLAY
 A SONG FOR ME
1490 DATA EVERYTHING THAT GOES UP MUST C
OME DOWN
1500 DATA HICKORY DICKORY DOCK THE MOUSE
 RAN UP THE CLOCK
```

Program 2.

```
15 REM FOR TAPE VERSION, USE THESE
   LINES INSTEAD
16 REM YOU MAY ALSO DELETE LINES
   960-965
17 REM
50 TRAP 70:OPEN #2,4,0,"C:":INPUT
   #2,P$
65 CLOSE #2:TRAP 40000:GOTO 400
70 CLOSE #2:? "[B]tape error":GOSUB
   990:GOTO R
850 R=800:TRAP 70:OPEN #2,8,0,"C:":
    TRAP 40000
```

©

SuperFont

Charles Brannon
Editorial Assistant

The ability to redefine the character set is one of the more useful features of the Atari. In a previous article, "Character Generation on the Atari," (**COMPUTE!** #9) I explained the principles and techniques of customizing the character set. Basically, it involves the plotting of a character on an eight by eight matrix and then converting each row into a binary number.

This process, however, is slow and tedious for the programmer. Fortunately, it is an obvious candidate for computerization. The computer could display a grid, let you set and clear points on it, and then do the binary-to-decimal conversion for you. It could also let you save and load completed *fonts* (character sets) from tape or disk.

Although SuperFont may lack some of the features of commercial products, it is quite powerful and versatile. SuperFont is written in BASIC, but what makes it special is that it has several machine language subroutines as well. One of these (thanks to DLI) enables the redefined character set to be displayed on the screen at the same time as the regular one. This permits you to see the effects of your changes without letting the command menu or prompts turn into starships.

SuperFont uses player/missile graphics for fast updates and a colorful grid. Since the special character window is set off in a different color than the rest of the screen (again via DLI's), you get eight different colors to delight the eye. The human interface is enhanced with the use of a joystick to plot points in the eight by eight grid.

SuperFont has 18 commands:

E:Edit	**R:Restore**
F:Copy From	**X:Switch**
T:Copy To	**C:Clear**
O:Overlay	**I:Invert**
S:Save Font	**L:Load Font**
:Delete	**:Insert**
:Scroll Left	**:Scroll Right**
W:Write Data	**Q:Quit**
:Reverse	**G:Graphics**

This menu is displayed on the screen along with a "checkerboard" plotting grid, the 128 characters of the character set, and the 128 characters of the alternate character set. Some commands require you to select a character. A cursor will be placed on each of the character sets. You can move the cursors around the sets simultaneously. When the cursor is on the desired character, press the fire button to indicate it. An explanation of each command follows:

Edit: The basic editing command. The selected character is copied into the grid and a flashing cursor is homed into the grid. You move the cursor with the joystick. Pressing fire will set a point (if a point is clear) or reset (clear) a point (if a point is already set). You can draw lines by holding down the button while moving the joystick. Any changes are immediately visible in the character set and the character displayed in GRAPHICS mode one and two lines at the bottom of the screen. To completely redesign a character, use the Clear command, and then design the character from scratch.

Restore: This command will "fix" a character by copying the original bit pattern into it. Very useful if you have mangled a character or changed the wrong one.

Copy From: You select a character which is copied into the current character. The grid is updated, and you can further edit the character.

Copy To: The current character is copied to (replaces) the indicated character.

Switch: Exchanges the current character's bit pattern with the selected character.

Overlay: The selected character is overlaid upon the current character. This lets you combine two characters to form a new one.

Clear: Clears out the current character. For creating unique characters.

Invert: Turns the current character "upside down." For example a re-defined M could be inverted and copied to the W.

Save Font: Saves the alternate character set in compact form with a machine-language routine. Answer "Filename?" with either C: or D:filespec. If you see an error message, press a key to return to the menu.

Load Font: Retrieves a character set from tape or disk. Answer the "Filename" prompt as you did in Save Font.

Cursor-up or SHIFT-DELETE: Similar to Delete Line in BASIC. The line of dots the cursor rests on is deleted; the following lines are pulled up to fill the gap.

Cursor-down or SHIFT-INSERT: Similar to Insert Line in BASIC. A blank line is inserted at the cursor position. The bottom line is lost.

Scroll left: The bit pattern of the character is shifted to the left.

Scroll right: The bit pattern of the character is

shifted to the right.

Write Data: The internal code (0-127) of the current character is printed in reverse-field followed by the eight bytes (in decimal) of the character. If you want a printout of the entire character set, use the auxiliary program CHPRINT (Program 3). Pressing any menu selection key will erase the nine bytes.

Graphics: Toggles the TEXT/GRAPHICS option of the GRAPHICS mode one and two lines to let you see each half of the character set.

Reverse: Puts the character in reverse field: all dots become blanks, and all blanks become dots. Reverse field versions of the characters are not normally stored in the character set, but you may want this for special graphics, such as reverse-field text in GRAPHICS modes one or two.

Quit: Exits program.

The commands offer flexibility in working with character sets, but there may be other functions you may want to add. The program is modular in structure; just follow the branching IF statements after line 790 to 1370 and replace the 520 (IF K<> ASC("G") THEN 520) with a link to your additional command(s). You may also want to change the colors. Besides the SETCOLOR statements in line 170, change the zero in line 300 (POKE 1538,0) to COLOR (0-15)*16 + LUMINANCE (0-14). Similarly, you can play with the player/missile colors in line 360.

It is also possible to use the character set data on tape or disk directly. It is written as a series of 1024 bytes: the bytes of the character set — no more, no less. I have included two extra utility programs which access the character data. Program 2 simply loads the set into memory and changes CHBASE (756) to point to it. Program 3 produces a formatted hex or decimal dump of the character set. Both programs should have the "filespec" changed to the filename of your character set.

The code of the main program is fairly straight forward. It uses several machine language subroutines: (1) A Display List Interrupt handler to maintain the special character window. (2) Copies the ROM character table into the RAM CHSET table (avoids the 15 second delay in BASIC). (3) A LOGIC subroutine that permits AND, OR, EOR (and any other 6502 function such as ROR) to be used on a binary level (see also "Make Your Atari a Bit Wiser," **COMPUTE!** #12, p. 74). (4) Implements a fast machine language memory save thanks to the IOCB PUTREC and GETREC commands.

You can do a lot with this capability: custom fonts (Greek, "Computeristic," script), graphics characters (special line drawing characters, spaceships, "invaders," bombs, tanks, planes, ships, even little people! (INTRUDER ALERT! INTRUDER ALERT!) SuperFont makes your task easier, even fun!

Program 1.

```
100 REM |  ***  SuperFont ***       |
110 REM | 11/10/81 Charles Brannon |
120 REM |  Character Set Editor     |
130 REM
140 DIM I(7),FN$(14),N$(3)
150 IF PEEK(1536)=0 THEN GOSUB 1400
160 GRAPHICS 0:POKE 752,1
170 SETCOLOR 2,7,2:SETCOLOR 4,7,2
180 DL=PEEK(560)+256*PEEK(561)+4
190 SD=PEEK(88)+256*PEEK(89)+13*40:ASD=S
D+5*40
200 A1=1630:FUNC=1631:A2=1632:LOGIC=1628

210 RAM=PEEK(106)-8:PMBASE=RAM*256
220 CHRORG=57344
230 POKE 559,46:POKE 54279,RAM
240 POKE 53277,3:POKE 53256,3
250 CHSET=(RAM-8)*256
260 POKE DL+23,6:POKE DL+24,7
270 POKE DL+18,130
280 POKE 512,0:POKE 513,6
290 POKE 54286,192
300 POKE 1549,RAM-8:POKE 1538,0
310 A=USR(1555,CHSET)
320 P0=PMBASE+512+20:P1=PMBASE+640+20:P2
=PMBASE+768+20:P=PMBASE+896+20:T=85
330 FOR I=0 TO 7:FOR J=0 TO 3:T=255-T:PO
KE P0+I*4+J,0:POKE P1+I*4+J,T:T=255-T
340 POKE P2+I*4+J,T:NEXT J:T=255-T:NEXT
I
350 POKE 53248,64:POKE 53249,64:POKE 532
50,64
360 POKE 704,198:POKE 705,240:POKE 706,6
8
370 POKE 53256,3:POKE 53257,3:POKE 53258
,3:POKE 623,1
380 ? " {Q}{8 R}{E}":FOR I=1 TO 8:? " {=
}        {=}":NEXT I:? " {Z}{8 R}{C}"
390 POKE 82,14:POSITION 14,1
400 ? "|E| Edit          |R| Restore"
410 ? "|F| Copy From     |X| Switch"
420 ? "|T| Copy To       |C| Clear"
430 ? "|O| Overlay       |I| Invert"
440 ? "|S| Save Font     |L| Load Font"
450 ? "{ESC}{DEL LINE} Delete        {ESC}
{INS LINE} Insert"
460 ? "{ESC}{CLR TAB} Scroll Left {ESC}{
SET TAB} Scroll{DOWN}{4 LEFT}Right"
```

```
470 ? "|W| Write Data  |Q|uit"
480 ? "{DOWN}|{F}{=}{G}| Reverse   |G| G
raphics"
490 FOR I=0 TO 3:FOR J=0 TO 31:POKE SD+J
+I*40+4,I*32+J:POKE ASD+J+I*40+4,I*32+J:
NEXT J:NEXT I
500 POKE 82,2:POSITION 0,0
510 OPEN #2,4,0,"K:"
520 P=PEEK(764):IF P=255 THEN 520
530 IF P=60 THEN 520
540 IF P=39 THEN POKE 764,168
550 GET #2,K
560 IF K<>ASC("E") THEN 790
570 GOSUB 1750
580 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I):FOR
 J=0 TO 3:POKE P0+I*4+J,A:NEXT J:NEXT I
590 POKE ASD+169,C:POKE ASD+190,C
600 JX=0:JY=0
610 POSITION JX+4,JY+1
620 ? CHR$(32+128*FF);"{LEFT}";:FF=1-FF
630 IF STRIG(0)=0 THEN 750
640 IF PEEK(764)<255 THEN ? " ";:GOTO 52
0
650 ST=STICK(0):IF ST=15 THEN 620
660 IF STRIG(0) THEN FOR I=0 TO 100 STEP
 20:SOUND 0,100-I,10,8:NEXT I
670 POSITION JX+4,JY+1:? " ";
680 JX=JX+(ST=7)-(ST=11)
690 JY=JY+(ST=13)-(ST=14)
700 IF JX<0 THEN JX=7
710 IF JX>7 THEN JX=0
720 IF JY<0 THEN JY=7
730 IF JY>7 THEN JY=0
740 GOTO 610
750 POKE A1,PEEK(CHSET+C*8+JY):POKE A2,2
^(7-JX):POKE FUNC,73:A=USR(LOGIC)
760 POKE CHSET+C*8+JY,A:FOR J=0 TO 3:POK
E P0+JY*4+J,A:NEXT J
770 FOR I=1 TO 10:SOUND 0,I*4,8,8:NEXT I
:SOUND 0,0,0,0
780 GOTO 650
790 IF K<>ASC("F") THEN 830
800 S=C:GOSUB 1750
810 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I):POK
E CHSET+S*8+I,A:NEXT I
820 C=S:GOTO 580
830 IF K<>ASC("T") THEN 870
840 S=C:GOSUB 1750
850 FOR I=0 TO 7:A=PEEK(CHSET+S*8+I):POK
E CHSET+C*8+I,A:NEXT I
860 C=S:GOTO 600
870 IF K<>ASC("O") THEN 920
880 S=C:GOSUB 1750
890 FOR I=0 TO 7:POKE A1,PEEK(CHSET+C*8+
I):POKE A2,PEEK(CHSET+S*8+I):POKE FUNC,9
:A=USR(LOGIC)
900 POKE CHSET+S*8+I,A:NEXT I
910 C=S:GOTO 580
920 IF K<>ASC("R") THEN 940
930 FOR I=0 TO 7:POKE CHSET+C*8+I,PEEK(C
HRORG+C*8+I):NEXT I:GOTO 580
940 IF K<>ASC("C") THEN 960
950 FOR I=0 TO 7:POKE CHSET+C*8+I,0:NEXT
 I:GOTO 580
960 IF K<>ASC("{R}") THEN 980
970 FOR I=0 TO 7:POKE CHSET+C*8+I,255-PE
EK(CHSET+C*8+I):NEXT I:GOTO 580
980 IF K<>ASC("X") THEN 1010
990 S=C:GOSUB 1750
1000 FOR I=0 TO 7:A=PEEK(CHSET+S*8+I):PO
KE CHSET+S*8+I,PEEK(CHSET+C*8+I):POKE CH
SET+C*8+I,A:NEXT I:GOTO 580
1010 IF K<>ASC("I") THEN 1030
1020 FOR I=0 TO 7:I(I)=PEEK(CHSET+C*8+I)
:NEXT I:FOR I=0 TO 7:POKE CHSET+C*8+I,I(
7-I):NEXT I:GOTO 580
1030 IF K<>ASC("{UP}") AND K<>ASC("{DEL
LINE}") THEN 1050
1040 FOR I=JY TO 6:POKE CHSET+C*8+I,PEEK
(CHSET+C*8+I+1):NEXT I:POKE CHSET+C*8+7,
0:GOTO 580
1050 IF K<>ASC("{DOWN}") AND K<>ASC("{IN
S LINE}") THEN 1070
1060 FOR I=7 TO JY STEP -1:POKE CHSET+C*
8+I,PEEK(CHSET+C*8+I-1):NEXT I:POKE CHSE
T+C*8+JY,0:GOTO 580
1070 IF K<>ASC("{LEFT}") THEN 1100
1080 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I)*2:
IF A>255 THEN A=A-256
1090 POKE CHSET+C*8+I,A:NEXT I:GOTO 580
1100 IF K<>ASC("{RIGHT}") THEN 1130
1110 FOR I=0 TO 7:A=PEEK(CHSET+C*8+I)/2
1120 POKE CHSET+C*8+I,A:NEXT I:GOTO 580
1130 IF K<>ASC("Q") THEN 1150
1140 POKE 53248,0:POKE 53249,0:POKE 5325
0,0:POKE 53277,0:GRAPHICS 0:END
1150 IF K<>ASC("S") THEN 1210
1160 GOSUB 1610:POKE 195,0
1170 TRAP 1190:OPEN #1,8,0,FN$
1180 A=USR(1589,CHSET)
1190 CLOSE #1:TRAP 40000:IF PEEK(195) TH
EN 1260
1200 POKE 54286,192:GOTO 580
1210 IF K<>ASC("L") THEN 1290
1220 GOSUB 1610:POKE 195,0
1230 TRAP 1250:OPEN #1,4,0,FN$
1240 A=USR(1619,CHSET)
1250 CLOSE #1:TRAP 40000:IF PEEK(195)=0
THEN 1200
1260 POSITION 14,0:? "{BELL}* ERROR -";P
EEK(195);" *"
1270 IF PEEK(764)<255 THEN POSITION 14,0
```

```
:? "                    ":GOTO 1200
1280 GOTO 1270
1290 IF K<>ASC("W") THEN 1370
1300 POSITION 2,10:N$="    ":L=LEN(STR$(C
)):N$(1,L)=STR$(C):L=LEN(N$)
1310 FOR I=1 TO L:? CHR$(ASC(N$(I,I))+12
8);:NEXT I:? ")";
1320 FOR I=0 TO 2:FOR J=0 TO 1+(I>0):A=P
EEK(CHSET+C*8+J+I*3)
1330 SOUND 0,(I*3+J)*10+50,10,8
1340 PRINT A;",";:NEXT J:? "(BACK S)":NE
XT I:SOUND 0,0,0,0
1350 IF PEEK(764)=255 THEN 1350
1360 POSITION 2,10:FOR I=1 TO 3:? "
      ":NEXT I:GOTO 520
1370 IF K<>ASC("G") THEN 520
1380 CF=1-CF:POKE 1549,RAM-8+2*CF
1390 GOTO 520
1400 GRAPHICS 2+16:SETCOLOR 4,1,4:POSITI
ON 5,3:? #6;"SUPER|FONT|"
1410 POSITION 4,5:? #6;"patience(3 ND":P
OSITION 2,7:? #6;"|charles brannon|"
1420 FOR I=1536 TO 1639:READ A:POKE I,A:
POKE 709,A:SOUND 0,A,10,4:NEXT I
1430 SOUND 0,0,0,0:RETURN
1440 DATA 72,169,100,141,10,210
1450 DATA 141,24,208,141,26,208
1460 DATA 169,6,141,9,212,104
1470 DATA 64,104,104,133,204,104
1480 DATA 133,203,169,0,133,205
1490 DATA 169,224,133,206,162,4
1500 DATA 160,0,177,205,145,203
1510 DATA 200,208,249,230,204,230
1520 DATA 206,202,208,240,96,104
1530 DATA 162,16,169,9,157,66
1540 DATA 3,104,157,69,3,104
1550 DATA 157,68,3,169,0,157
1560 DATA 72,3,169,4,157,73
1570 DATA 3,32,86,228,96,104
1580 DATA 162,16,169,5,76,58
1590 DATA 6,9,104,169,0,9,0,133
1600 DATA 212,169,0,133,213,96
1610 POSITION 14,0:? "Filename?";
1620 FN$="":K=0
1630 POKE 20,0
1640 IF PEEK(764)<255 AND PEEK(764)<>39
AND PEEK(764)<>60 THEN 1670
1650 IF PEEK(20)<10 THEN 1640
1660 ? CHR$(21+11*K);"(LEFT)";:K=1-K:GOT
O 1630
1670 GET #2,A
1680 IF A=155 THEN ? " ";:FOR I=1 TO LEN
(FN$)+10:? "(BACK S)";:NEXT I:RETURN
1690 IF A=126 AND LEN(FN$)>1 THEN FN$=FN
$(1,LEN(FN$)-1):? " (LEFT)";CHR$(A);:GOT
O 1630
1695 IF A=126 AND LEN(FN$)=1 THEN ? CHR$
(A);:GOTO 1620
1700 IF A=58 OR (A>=48 AND A<=57) OR (A>
=65 AND A<=90) OR A=46 THEN 1720
1710 GOTO 1630
1720 IF LEN(FN$)<14 THEN FN$(LEN(FN$)+1)
=CHR$(A):? CHR$(A);
1730 GOTO 1630
1740 END
1750 REM GET CHOICE OF CHARACTER
1760 CY=INT(MRY/32):CX=MRY-32*CY
1770 C=CX+CY*32
1780 POKE SD+CX+CY*40+4,C+128
1790 POKE ASD+CX+CY*40+4,C+128
1800 IF STRIG(0)=0 OR PEEK(764)<255 THEN
 MRY=C:GOTO 1900
1810 ST=STICK(0):IF ST=15 THEN 1800
1820 POKE 53279,0
1830 GOSUB 1900
1840 CX=CX-(ST=11)+(ST=7):CY=CY-(ST=14)+
(ST=13)
1850 IF CX<0 THEN CX=31:CY=CY-1
1860 IF CX>31 THEN CX=0:CY=CY+1
1870 IF CY<0 THEN CY=3
1880 IF CY>3 THEN CY=0
1890 GOTO 1770
1900 POKE SD+CX+CY*40+4,C
1910 POKE ASD+CX+CY*40+4,C
1920 RETURN
```

Program 2.

```
100 REM CHLOAD--CHARACTER SET LOADER
110 CHSET=PEEK(106)-8:POKE 756,CHSET
120 CHSET=CHSET*256
130 TRAP 180
140 OPEN #1,4,0,"D:FONT":REM YOUR FILENA
ME HERE
150 FOR I=0 TO 1023
160 GET #1,A:POKE CHSET+I,A
170 NEXT I
180 CLOSE #1
```

Program 3.

```
100 REM CHPRINT--CHARACTER SET PRINTOUT
110 TRAP 340
120 OPEN #1,4,0,"D:FONT":REM YOUR FILENA
ME HERE
130 OPEN #2,8,0,"P:":REM CHANGE TO "E:"
FOR SCREEN
140 PRINT "|1| HEX OR |2| DECIMAL";:INPU
T TYPE
150 DIM HEX$(16),F$(3)
160 HEX$="0123456789ABCDEF"
165 LSB=-1
```

```
170 FOR I=0 TO 1023 STEP 8
180 F$="    ":C=INT(I/8)
190 IF TYPE=2 THEN F$(1,LEN(STR$(C)))=ST
R$(C):PRINT #2;F$;":";:GOTO 250
200 LSB=LSB+1:IF LSB=256 THEN LSB=0:MSB=
MSB+1
210 PRINT #2;"$";HEX$(MSB+1,MSB+1);
230 HINYB=INT(LSB/16):LONYB=LSB-16*HINYB

240 PRINT #2;HEX$(HINYB+1,HINYB+1);HEX$(
LONYB+1,LONYB+1);": ";
250 FOR J=0 TO 7
260 GET #1,A
270 F$="    ":IF TYPE=2 THEN F$(1,LEN(STR
$(A)))=STR$(A):PRINT #2;" ";F$;:GOTO 310

290 HINYB=INT(A/16):LONYB=A-16*HINYB
300 PRINT #2;HEX$(HINYB+1,HINYB+1);HEX$(
LONYB+1,LONYB+1);" ";
310 NEXT J
320 PRINT #2
330 NEXT I
340 CLOSE #1:CLOSE #2
```

©

Word Search

Bob Jones
Cranbury, NJ

Basically this program creates two matrices. The first matrix, the A matrix, is the one we shall hide the words in. Since the ATARI and many other BASICs I have run across do not permit the use of strings in a matrix, I have found that the next best thing to do is use the ASCII value of the characters instead. In this case it seems to be even simpler. The A matrix is initialized in line 10 to a random set of numbers between 65 and 91, (the ASCII value of the letters A thru Z). The C matrix will be our control matrix and our answer key. In line 10, all locations in C are initialized to 42, (the ASCII value for the character "*").

Next the user is asked to input 12 words, (the subroutine called by line 15). Lines 3015 through 3130 simply set A$ equal to the word to be processed, selected by the variable I. Line 45 sets L equal to the length of the word and if it is too long, (greater than ten letters) asks the user to input a shorter word. In line 50 we convert letters of the word to their ASCII values and place them in the B array, (a numeric array also initialized to all zero's by line 10). This array is our workhorse. Line 60 serves two functions: first, to generate a random starting location within both matrices and, second, to generate a random direction for the word to go in.

Now comes the math. Line 70 directs the program to one of eight subroutines, each one representing a different possible direction for the word to travel in. I shall only go over the first one, (lines 500 to 550) as the others work the same way. Line 500 checks to see that the word will fit within the matrix, if not the program is directed back to line 60 to generate a new starting location and direction. In line 510 we check the position of the word against the C or control matrix for possible conflicts with words already placed within that matrix. If a conflict exists the program is again directed back to line 60. Line 520 checks for a crossover with a previous word and if there is one it sets a flag, (the variable F) equal to 1. Line 630 directs the program to lines 2000 to 2020, these lines would have been repeated 8 times, once for each direction subroutine so in order to save memory they are only listed once and called upon when needed. The use of the 'GOTO' instead of the 'GOSUB' command is necessary in order to conditionally return to other portions of the program without confusing the computer by jumping in and out of subroutines. In these lines, (2000 to 2020) we continue to process our word, if there is a crossover (F = 1), or we have tried 300 times to find one, (determined by the variable R) we continue, otherwise we go back to line 60. Line 2020 gets us back into our original subroutine. Line 550 is the last line of our subroutine, it places our word into the A and C matrix's and sends us on to get a new word.

Line 80 determines if we have processed all of our words, and if so sends us on. In line 100 we print our hidden word matrix by printing the letters represented by our ASCII values, and when we are ready, line 110 prints our C matrix which is now our answer key.

This program requires more than 8K of memory as stands to run, though it will load into 8K of memory. It is a simple matter to shorten it by cutting out some of the possible direction subroutines. Also you can ask for the words to be INPUT as they are needed rather than storing them in string arrays. This program can be run on almost any computer using BASIC as stands, the only possible modifications that might be needed are with the GOTO statements like 'GOTO D*100'. These may be changed to 'ON D GOTO 500,600, 700,800,900,1000,1100,1200'. Or you could use the 'IF...THEN' statements, though the program won't be as much fun. A '?' is simply a PRINT command. The POKE statements are not necessary: they simply speed up the program. (Thanks to Ed Stewart, **COMPUTE!** #11.)

```
1 REM WORD SEARCH  WRITTEN BY BOB JONES
5 POKE 559,0:DIM A$(11),B$(11),C$(11),D$
(11),E$(11),F$(11),G$(11),H$(11),I$(11),
J$(11),K$(11),L$(11),M$(11)
10 DIM A(13,16),B(13),C(13,16):I=0:FOR X
=1 TO 13:B(X)=0:FOR Y=1 TO 16:A(X,Y)=INT
(26*RND(0)+65):C(X,Y)=42
15 NEXT Y:NEXT X:R=300:GOSUB 3000:GOTO 4
0
20 FOR X=1 TO 13:FOR Y=1 TO 16
30 R=0
40 GOSUB 3015
45 L=LEN(A$):IF L>10 THEN 3150
50 FOR S=1 TO L:B(S)=ASC(A$(S,S)):NEXT S

60 F=0:R=R+1:X=INT(13*RND(0)+1):Y=INT(16
*RND(0)+1):D=INT(8*RND(0)+5)
70 GOTO D*100
80 I=I+1:LPRINT A$:POKE 559,34:? A$:POKE
 559,0:IF I=12 THEN 100
90 GOTO 30
100 POKE 559,34:FOR X=1 TO 13:? :LPRINT
:FOR Y=1 TO 16:? CHR$(A(X,Y));:LPRINT CH
```

```
R$(A(X,Y));:NEXT Y:NEXT X
105 ? "TO SEE ANSWERS PRESS RETURN KEY":
INPUT A$
110 LPRINT :LPRINT :FOR X=1 TO 13:LPRINT
 :? :FOR Y=1 TO 16:LPRINT CHR$(C(X,Y));:
? CHR$(C(X,Y));:NEXT Y:NEXT X
120 LPRINT :LPRINT :LPRINT :LPRINT :LPRI
NT :END
500 IF Y+L-1>16 THEN 60
510 FOR Z=0 TO L-1:IF C(X,Y+Z)>42 AND C(
X,Y+Z)<>B(Z+1) THEN 60
520 IF C(X,Y+Z)=B(Z+1) THEN F=1
530 GOTO 2000
550 C(X,Y+S)=B(S+1):A(X,Y+S)=B(S+1):NEXT
 S:GOTO 80
600 IF Y-L+1<1 THEN 60
610 FOR Z=L-1 TO 0 STEP -1:IF C(X,Y-Z)>4
2 AND C(X,Y-Z)<>B(Z+1) THEN 60
620 IF C(X,Y-Z)=B(Z+1) THEN F=1
630 GOTO 2000
650 C(X,Y-S)=B(S+1):A(X,Y-S)=B(S+1):NEXT
 S:GOTO 80
700 IF X+L-1>13 THEN 60
710 FOR Z=0 TO L-1:IF C(X+Z,Y)>42 AND C(
X+Z,Y)<>B(Z+1) THEN 60
720 IF C(X+Z,Y)=B(Z+1) THEN F=1
730 GOTO 2000
750 C(X+S,Y)=B(S+1):A(X+S,Y)=B(S+1):NEXT
 S:GOTO 80
800 IF X-L+1<1 THEN 60
810 FOR Z=L-1 TO 0 STEP -1:IF C(X-Z,Y)>4
2 AND C(X-Z,Y)<>B(Z+1) THEN 60
820 IF C(X-Z,Y)=B(Z+1) THEN F=1
830 GOTO 2000
850 C(X-S,Y)=B(S+1):A(X-S,Y)=B(S+1):NEXT
 S:GOTO 80
900 IF X+L-1>13 OR Y+L-1>16 THEN 60
910 FOR Z=0 TO L-1:IF C(X+Z,Y+Z)>42 AND
C(X+Z,Y+Z)<>B(Z+1) THEN 60
920 IF C(X+Z,Y+Z)=B(Z+1) THEN F=1
930 GOTO 2000
950 C(X+S,Y+S)=B(S+1):A(X+S,Y+S)=B(S+1):
NEXT S:GOTO 80
1000 IF X-L+1<16 OR Y-L+1<1 THEN 60
1010 FOR Z=L-1 TO 0 STEP -1:IF C(X-Z,Y-Z
)>42 AND C(X-Z,Y-Z)<>B(Z+1) THEN 60
1020 IF C(X-Z,Y-Z)=B(Z+1) THEN F=1
1030 GOTO 2000
1050 C(X-S,Y-S)=B(S+1):A(X-S,Y-S)=B(S+1)
:NEXT S:GOTO 80
1100 IF Y-L+1<1 OR X+L-1>13 THEN 60
1110 FOR Z=0 TO L-1:IF C(X+Z,Y-Z)>42 AND
 C(X+Z,Y-Z)<>B(Z+1) THEN 60
1120 IF C(X+Z,Y-Z)=B(Z+1) THEN F=1
1130 GOTO 2000
1150 C(X+S,Y-S)=B(S+1):A(X+S,Y-S)=B(S+1)
:NEXT S:GOTO 80
1200 IF Y+L-1>16 OR X-L+1<1 THEN 60
1210 FOR Z=L-1 TO 0 STEP -1:IF C(X-Z,Y+Z
)>42 AND C(X-Z,Y+Z)<>B(Z+1) THEN 60
1220 IF C(X-Z,Y+Z)=B(Z+1) THEN F=1
1230 GOTO 2000
1250 C(X-S,Y+S)=B(S+1):A(X-S,Y+S)=B(S+1)
:NEXT S:GOTO 80
2000 NEXT Z:IF F>0 OR R>300 THEN 2020
2010 GOTO 60
2020 FOR S=0 TO L-1:GOTO D*100+50
3000 POKE 559,34:? "TYPE WORD AND THEN H
IT RETURN":INPUT B$,C$,D$,E$,F$,G$,H$,I$
,J$,K$,L$,M$:POKE 559,0:RETURN
3015 IF I=0 THEN A$=B$
3020 IF I=1 THEN A$=C$
3030 IF I=2 THEN A$=D$
3040 IF I=3 THEN A$=E$
3050 IF I=4 THEN A$=F$
3060 IF I=5 THEN A$=G$
3070 IF I=6 THEN A$=H$
3080 IF I=7 THEN A$=I$
3090 IF I=8 THEN A$=J$
3100 IF I=9 THEN A$=K$
3110 IF I=10 THEN A$=L$
3120 IF I=11 THEN A$=M$
3130 RETURN
3150 POKE 559,34:? A$;" TOO LONG MUST BE
 NO GREATER THAN 10 LETTERS TRY ANOTHER
WORD";:INPUT A$:POKE 559,0:GOTO 45
```

©

Review:

Screen Printer Interface (Version 2.0) From Macrotronics

David D Thornburg
Innovision
Los Altos, CA

More than anything else, I use my Atari computer for the creation of pictures. For various reasons, it is not enough for me to see these pictures on a TV screen — I also need copies of them on plain paper. Fortunately, there is an exceptionally well designed product which makes this a very simple task. That product is the screen printer Interface from Macrotronics. This program allows the user to transfer any image from the display screen to a suitable graphics printer with a single keystroke. The printed image can (if you choose) preserve grey scales, and can be printed in any size from a single sheet to a poster which would cover a wall. The user can choose among several printers (Trendcom, IDS, Centronics, Epson), and does not need the Atari 850 interface unit. Instead of the 850, Macrotronics provides a printer interface cable which connects to joystick ports 3 and 4. The screen printer software comes on a disk containing DOS 1, and they also provide a copy of the utility which is compatible with DOS 2.

The manual is clearly written and contains many examples showing the use of this interface with all language environments presently supported by Atari (BASIC, Assembler, PILOT).

Setting It Up

To use the system, one first connects the printer to the joystick ports with the cable provided and then boots the system from the disk supplied. During the boot process, the screen prompts the user for information on the printer being used. Once this is done, the rest of the program is loaded (the total utility occupies less than 3K bytes) and the familiar blue screen appears.

From this point on, the printer driver software is tucked safely inside the computer where it remains to do your bidding until the computer is turned off. Any command which sends information to device P: will cause this information to be printed. BASIC commands such as LPRINT behave just as they would for an Atari printer connected through the serial port.

While this system supports all text printing functions, the real value of this interface package is the power it gives as a graphics printing tool. Any time this system is in the computer you can get a dot-by-dot copy of the screen image by simply typing CTRL-P. Macrotronics has created some default printer conditions which cause most images to be printed quite nicely. The user has total control over the system parameters and can change the settings of various registers to create many different effects.

For example, the printed image can be scaled independently in both axes by POKEing a number between 1 and 16 in each of two memory locations. The default scale (16) produces a figure which fits nicely on 8.5" wide paper. As the scale values are decreased, the image size increases by 16/n where n is the scale value. Wide images are printed in multiple strips which can then be glued together. On multiple strip printouts, each strip overlaps the previous one by a little bit to make strip alignment simple. This attention to making life simple for the user is beautiful!

In addition to using the scale variables to make large pictures, they can also be used to adjust for the fact that most dot matrix printers have different inter-dot spacings on each axis. To get an accurate square on the Epson MX-100, for example, the vertical scale should be set to 14 (with the horizontal scale left at 16). The result is almost perfect.

In addition to scaling, the user can select positive or negative images, grey scale or black and white, determine grey scale from either hue or luminance data, print data which has been "fine scrolled," and print players and missiles.

In short, if your Atari computer can generate it, the Macrotronics screen printer can print it.

I use this software almost every day. So far I have used it to print the illustrations for three book manuscripts, numerous articles, several large posters and some custom bumper stickers.

The Only Error

The only error I have uncovered is that the default grey scale setting uses *hue* data rather than *luminance* data, but this is just a documentation error — the software works perfectly.

To see more examples of printouts made with this utility, look at any "Friends of the Turtle" column in **COMPUTE!**, or at the book *Picture This!*, soon to be published by Addison Wesley.

Figure 1.

Figure 2.

Figure 3.

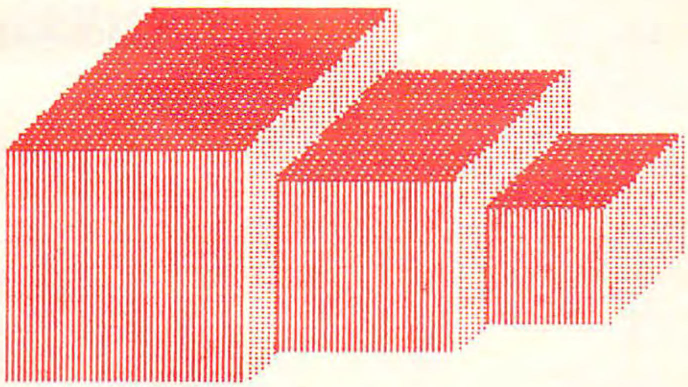
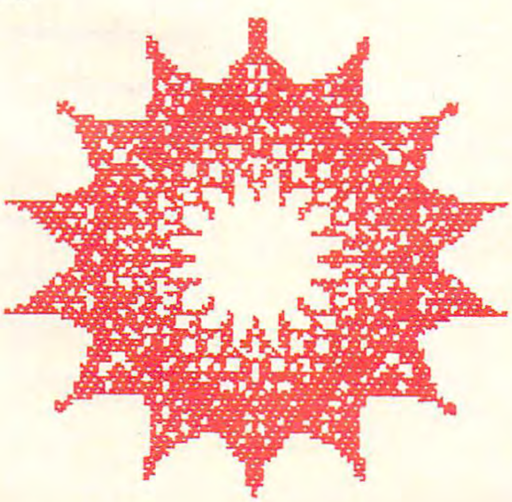


Figure 4.



©

INSIGHT: ATARI

Bill Wilkinson
Cupertino, CA

I have recently seen a copy of the complete *De Re Atari* (by Atari's own Chris Crawford, author of SCRAM and EASTERN FRONT, et al). Since two out of three people I talk to say "Huh?" when I mention the name, I have personally subtitled it *Everything You Ever Wanted to Know About the Atari Computers But Didn't Know Enough to Ask.* The book concerns itself with foibles, tricks, innards, hardware, software, and everything in between: there are even tricks using Atari BASIC (that are "obvious" upon discovery) which we never thought about when we designed the thing! I must heartily recommend that every serious Atari programmer trade in his or her left thumb, if necessary, for a copy of this book.

"De Re" (the insiders' appellation) is currently being serialized in BYTE magazine (I guess Atari's trying to impress the non-Atari world), but seeing the book in one piece is somehow more instructive. "De Re" is generally a fantastic resource, but it does often assume that the reader has intimate knowledge and understanding of the Atari Hardware Reference Manuals, etc. This is *not* a fault (the authors forewarn the reader); and, besides, it does leave room for columns like this. I don't intend to duplicate material in either Atari's manuals or "De Re", but there is bound to be some overlap. I intend to present the "hows" and "whys" to supplement Atari's "whats."

I try to write this column for the programmer: the person who knows software, but is unfamiliar with Atari hardware and/or Atari's system level software. If this column stretches your understanding of the Atari and/or its software, that's probably good. And I am constantly amazed at the questions which beginners on the Atari come up with; they often show "insights" to solution methods that I wouldn't dream of. The first questions are arriving in my mailbox. Send more!

This month's column is part three of the series on the Atari Operating System. Next month we will cover screen output, including graphics, to formally end the series. I have a few ideas on what should come next for you non-BASIC Atari users, but I would welcome some input. Also, this month, we begin a series which will explore the inner workings of Atari BASIC.

Atari I/O, Part 3: Device Handlers

As we noted before, Atari's OS is actually a very small program (approximately 700 bytes). Even so, it is able to handle the wide variety of I/O requests detailed in the first two parts of this series with a surprisingly simple and consistent assembly language interface. Perhaps even more amazing is the purity and simplicity of the OS interface to its device handlers.

Admittedly, because of this very simplicity, Atari's OS is sometimes slower than one would wish (probably only noticeably so with PUT BINARY RECORD and GET BINARY RECORD) and the handlers must be relatively sophisticated. But not overly so, as we will show.

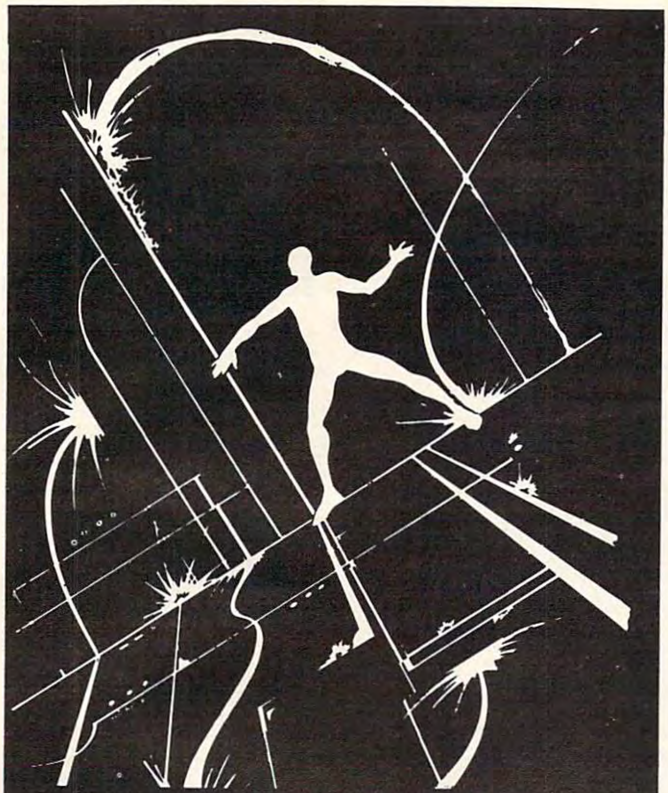
The Device Handler Table

Atari OS has, in ROM, a list of the standard devices (P:,C:,E:,S:, and K:) and the addresses thereof. So far, so good. But notice that, for example, the disk handler (D:) is not listed there; how does OS know about other devices? Simple. On SYSTEM RESET, the list is moved from ROM to RAM, and OS then utilizes only the RAM version. To add a device, simply tack it on to the end of the list: you need only specify the device's name (one character) and the address of its handler table (more on that in a moment). To reassure you that it is this simple, let me point out that this is exactly how the "D:" (Disk) handler is attached when the disk is booted.

In theory, all named device handlers under Atari OS may handle more than one physical device. Just as the disk handler understands "D1:" and "D2:", so could a printer handler understand "P1:" and "P2:". In practice, of all the standard Atari handlers only the Disk and Serial Port handlers can utilize the sub-device numbers. Incidentally, Atari OS supplies a default sub-device number of "1" if no number is given (thus "D:" becomes "D1:"). A project for those of you with two printers (there

```
        *=      $031A
HTABS                   ; the Printer device
        .WORD   PDEVICE ; and the address of its
                          driver
        .BYTE   'C'     ; the Cassette device
        .WORD   CDEVICE
        .BYTE   'E'     ; the screen Editor device
        .WORD   EDEVICE
        .BYTE   'S'     ; the graphics Screen
                          device
        .WORD   SDEVICE
        .BYTE   'K'     ; the Keyboard device
        .WORD   KDEVICE
        .BYTE   0       ; zero marks the end of
                          the table
        .WORD   0       ; ...but there's room for
                          up to
        .BYTE   0       ; ...9 more devices
        et cetera
```

Figure 1.



MICROWORLD, an adventure within your computer, is available on the ATARI and the TRS-80. You are transformed into an electroid, and must explore the circuits of your computer. Over 80 locations and many original problems exist within the maze of transformers and transistors. We dare you to explore the maze of bit cells! Each version of Microworld explores the workings of its respective computer, Atari or TRS-80. Microworld comes with a booklet defining terms and describing the function of the mystifying inner workings of home computers. Come face to face with a staticon! Explore the Microworld!

SATISFACTION GUARANTEED!
If for any reason you are not satisfied with our products, return your order within 14 days for a prompt and cheerful refund.

ORDERING INFORMATION
Orders are processed within five working days. Shipping and handling charge of $1.00 will be added to all orders within the U.S. and Canada. Overseas orders please add $3.00 for air post.

Atari Microworld Atari 400 and 800
32K Cassette $19.95
Atari 400 and 800
32K Diskette $22.95
TRS-80 Microwrold TRS-80 Model I and Model III
Level II 16K Cassette $19.95
TRS-80 Model I and Model III
Level II 32K Diskette $22.95

- MICROWORLD $ ______
- SHIPPING 1.00
- TOTAL $ ______

■ TRS-80 ■ ATARI
■ CASSETTE ■ DISKETTE
NAME ______
STREET ______
CITY ______ STATE ______ ZIP ______
■ CHECK ■ VISA ■ MASTERCARD
\# ______ EXP. DATE ______

MED SYSTEMS SOFTWARE
P.O. Box 2674, Chapel Hill, NC 27514
1-800-334-5470

Fantasy for your ATARI

Ali Baba and the forty thieves

By Stuart Smith

Guide your alter ego, Ali Baba, through the thieves mountain den in an attempt to rescue the beautiful princess. Treasure, magic, and great danger await you! One or more human players can guide up to seventeen friendly characters through the many rooms, halls, and caves. Some characters wander around randomly, making each adventure a little different.

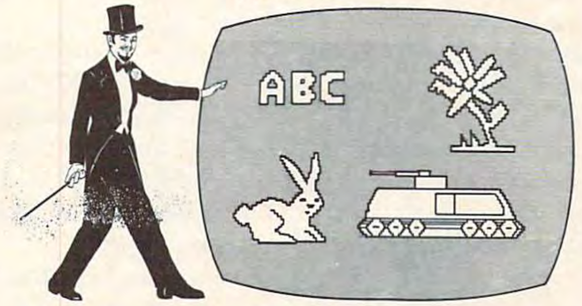
ALI BABA AND THE FORTY THIEVES is written in high resolution color graphics and includes music and sound effects. Adventures can be saved to disk and resumed at a later time. Requires 32K.

On Diskette Only — $32.95

Graphics for your ATARI

Character Magic

By Chris Hull



It's easy to create your own character sets and save them to diskette or cassette with CHARACTER MAGIC. But this is not just another character editor. CHARACTER MAGIC helps you use all the character types that the Atari is capable of, including descending characters (8 x 10 dots) and two types of five-color character graphics not supported by Atari's Operating System. Documentation includes examples of display lists that let you use these "secret" graphics modes. Requires 32K.

Cassette or Diskette - $29.95

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

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.

QS QUALITY SOFTWARE

6660 Reseda Blvd., Suite 105, Reseda, Ca. 91335 (213) 344-6599

*Indicates trademarks of Atari

must be one or two of you): presumably one of them is connected via the MacroTronics interface; if so, try modifying the MacroTronics handler so that "P1:" refers to the Atari 850 interface while "P2:" refers to the MacroTronics. It's really a fairly easy project, presuming you have the listings of Atari's OS (which are available from Atari).

Rules For Writing Device Handlers

Each device which has its handler address placed into the handler address table (above) is expected to conform to certain rules. In particular, the driver is expected to provide six action subroutines and an initialization routine. (In practice, I believe the current Atari OS only calls the initialization routines for its own pre-defined devices. Since this may change in future OS's and since one can force the call to one's own initialization routine, I must recommend that each driver include one, even if it does nothing.) The address placed in the handler address table must point to, again, another table, the form of which is shown in Figure 2.

Notice the six addresses which must be specified; and note that, in the table, one must subtract one from each address (the "-1" simply makes CIO's job easier...honest). A brief word about each routine is in order.

The OPEN routine must perform any initialization needed by the device. For many devices, such as a printer, this may consist of simply checking the device status to insure that it is actually present. Since the X-register, on entry to each of these routines, contains the IOCB number being used for this call, the driver may examine ICAX1 (via LDA ICAX1,X) and/or ICAX2 to determine the kind of OPEN being requested. (Caution: Atari OS preempts bits 2 and 3, $04 and $08, of ICAX1 for read/write access control. These bits may be examined, but should normally not be changed.)

The CLOSE routine is often even simpler. It should "turn off" the device if necessary and if possible.

The PUTBYTE and GETBYTE routines are just what are implied by their names: the device handler must supply a routine to output one byte to the device and a routine to input one byte from the device. *However,* for many devices, one or the other of these routines doesn't make sense (ever tried to input from a printer?). In this case the routine may simply RTS and Atari OS will supply an error code.

The STATUS routine is intended to implement a dynamic status check. Generally, if dynamic checking is not desirable or feasible, the routine may simply return the status value it finds in the user's IOCB. However, it is *not* an error under Atari OS to call the status routine for an unOPENed device, so be careful.

The XIO routine does just what its name implies: it allows the user to call any and all special and wonderful routines that a given device handler may choose to implement. OS does nothing to process an XIO call except pass it to the appropriate driver.

Note: In general, the AUXilliary bytes of each IOCB are available to each driver. In practice, it is best to avoid ICAX1 and ICAX2, as several BASIC and OS commands will alter them at their will. Note that ICAX3 through ICAX5 may be used to pass and receive information to and from BASIC via the NOTE and POINT commands (which are actually special XIO commands). Finally, drivers should not touch any other bytes in the IOCBs, especially the first two bytes.

Notice that handlers need not be concerned with PUT BINARY RECORD, GET TEXT RECORD, etc.: OS performs all the needed housekeeping for these user-level commands.

```
HANDLER
        .WORD   <address of OPEN routine>-1
        .WORD   <address of CLOSE routine>-1
        .WORD   <address of GETBYTE routine>-1
        .WORD   <address of PUTBYTE routine>-1
        .WORD   <address of STATUS routine>-1
        .WORD   <address of XIO routine>-1
        JMP     <address of initialization routine>
```

Figure 2.

Rules For Adding Things To OS

We touched on this subject last month, in the section titled "The Easiest Way of Making Room?", but a review and an addition are in order. Both Atari FMS (File Manager System, otherwise known as DOS and/or the Disk Device Driver) and the serial port handlers follow the same scheme when they add themselves to OS, so it is safe to assume that this method may be considered the *de facto* Atari standard. We enumerate:

1. Inspect the system MEMLO pointer (at $2E7, I called it LOMEM last month, which is BASIC's name for it).

2. Load your routine (including needed buffers) at the current value of MEMLO.

3. Add the size of your routine to MEMLO.

4. Store the resultant value back in MEMLO.

5. Connect your driver to OS by adding its name and address into the handler address table.

6. Fool OS so that if SYSTEM RESET is hit steps 3 through 5 will be re-executed (because SYSTEM RESET indeed resets the handler address table and the value of MEMLO).

In point of fact, step 2 is the hardest of these to accomplish. In order to load your routine at wherever MEMLO may be pointing, you need a relocatable (or self-relocatable) routine. Since there is currently no assembler for the Atari which produces relocatable code, this is not an easy task. (*However*, I just happen to have a method which works. But it will have to wait for a later article.)

Step 6 is accomplished by making Atari OS think that your driver is the Disk driver for initialization purposes (by "stealing" the DOSINI vector) and then calling the Disk's initializer yourself when steps 3 through 5 are performed. This is a fairly simple process, but again, details must await a future article.

Yet Another Real Live Example

I promised last month that we would present a driver for a "peripheral" device found in every Atari, yet not supported by any Atari device handlers. I could have been cagey and presented a driver for a "Null" device. (A handy thing to have, actually: One can throw away one's output *very* fast when trying to debug a program. See *De Re Atari* for a simple implementation of one. Better yet, try to write one from the information presented herein.) Being a glutton for punishment, I undertook to write a truly useful handler for Atari's overlooked device: RAM memory!!

After the snickers and sarcastic comments die down, let me point out how truly useful such a device is to BASIC programs: program one can "write" data to RAM and then chain to program two, which then "reads" the same data back. Voila! Chaining with COMMON in Atari BASIC. So herewith the "M:" (Memory) driver, presented in its entirety in Figure 3.

Does It Work?

Some words of caution are in order. This driver does *not* perform step 6 as noted in the last section (but it may be reinitialized via a BASIC USR call). It does *not* perform self-relocation: instead it simply locates itself above all normal low memory usage (except the serial port drivers, which would have to be loaded *after* this driver). If you assemble it yourself, you could do so at the MEMLO you find in your normal system configuration (or you could improve it to be self-modifying, of course).

Other caveats pertain to the handler's usage: it uses RAM from the contents of MEMTOP ($2E5) downward. It does *not* check to see if it has bumped into BASIC's MEMTOP ($90) and hence could conceivably wipe out programs and/or data. To be safe, don't write more data to the RAM than a FRE(0) shows (and preferrably even less).

In operation, the M: driver reinitializes upon an OPEN for write access (mode 8). A CLOSE followed by a subsequent READ access will allow the data to be read in the order it was written. More cautions: don't change graphics modes between writing and reading if the change would use more memory (to be safe, simply don't change at all). The M: will perform almost exactly as if it were a cassette file, so the user program should be data sensitive if necessary: the M: driver will *not* itself give an error based on data contents. Note that the data may be re-READ if desired (via CLOSE and re-OPEN).

Installing The M: Driver

The most obvious way to install this driver (Program 1) is to type in the source and assemble it directly to the disk. Then simply loading the object file from DOS 2 (or OS/A+) will activate the driver and move LOMEM as needed. You could even name the resulting file "AUTORUN.SYS" so that it would be automatically booted on power up.

If you don't have an assembler and/or disk, the problem is a little more difficult. If you are comfortable writing BASIC programs that load assembly language data to memory, you migth use the techniques described in last month's "Make Room?" to reserve the required memory. Then a simple POKEr program which uses DATA statements would suffice.

But the assembly listing given here is designed for a disk system and would waste 5K bytes or so in a cassette system. So, if you can't reassemble it and/or write that POKEr program, you will just have to be patient: I will try to give you a simplified BASIC POKEr program next month.

A suggested set of BASIC programs is presented:

Ending of Program 1:

```
9900 OPEN #2,8,0,"M:"
9910 PRINT #2; LEN(A$)
9920 PRINT #2; A$
9930 CLOSE #2
9940 RUN "D:PROGRAM2"
```

Beginning of Program 2:

```
100 JUNK = USR( 7984 )
    [ to insure the M: driver is linked, in case of
    RESET ]
110 OPEN #4,4,0,"M:"
120 INPUT #4, SIZE
130 DIM STRING$(SIZE)
140 INPUT #4, STRING$
150 CLOSE #4
```

BASIC A+ users might find RPUT/RGET and BPUT/BGET to be useful tools here instead of PRINT and INPUT. And, of course, users of any other language(s) might find this a handy inter-program communications device.

BASIC, Part 1: Why?

The first "Why?" I usually hear is "Why not Microsoft BASIC?" After a little probing, I find that the question really boils down to "Why not string arrays?" There is no simple answer to that question, so I hope to save myself time in the future by pointing toward these articles. Because I intend to give the true and not-so-simple answer, along with some (hopefully) very interesting information.

Believe it or not, Atari BASIC pretty much works the way it was designed and specified. And yours truly must take a large part of the brickbats or roses you might throw because of those specifications. We (that is, at the time, Shepardson Microsystems) were just finishing the highly successful and very powerful Cromemco 32K Structured BASIC. And, while a few Cromemco users had carped about the lack of string arrays, on the whole the real power of the language is extraordinarily impressive. All this "power" probably went to our head(s), so of course we had to duplicate the feat for Atari.

Oops. A small problem: Cromemco gave us 32K bytes for Structured BASIC; Atari gave us 10K bytes. What comes out? Wrong question! What can stay in?! Of course, Atari had some ideas, too, and the important features that we ended up with include (in my opinion):

Decimal Arithmetic
Long Variable Names
Long Strings (more than 255 bytes)
Flexible I/O
Reasonable Assembly Language Interface
Syntax Check at entrh time

That last item won't be appreciated by those of you who haven't used a BASIC that doesn't do it, so I will try to describe the horrors to you: You type in a long program which includes a line such as:

```
3034 IF SYSTEMERROR THEN
     PINT "Bad Disk Drive":
     GOTO 4090
```

Did you catch it? It says 'PINT' where it should say 'PRINT'. Most microcomputer BASICs will happily gulp that line in with nary a burp. Now, 13 months later, when that dreaded 'systemerror' actually occurs, your user (who lives in Hong Kong, naturally) sees the helpful message

```
***SYNTAX ERROR at LINE 3037
```

When you have fathomed the implications of that, calm your nerves so we can continue.

Needless to say, we were more than happy to include the Syntax Check feature. However, this inclusion had implications that rippled throughout the rest of the design of BASIC. First, you don't get something for nothing: such syntax checking uses memory, perhaps one to two kilobytes. Second, pre-syntaxing implies that the user program will be "tokenized": that is, the user's source will be converted into internal tokens for ease of execution and efficiency. Even Microsoft BASICs tokenize the keywords of the language; Atari BASIC tokenizes *everything:* keywords, variables, constants, operators, etc. Thirdly, the decision to have strings longer than 255 characters (coupled with the tight memory requirements) simply precluded any implementation of string arrays. (In fact, I do not know of *any* small-machine BASIC that supports string arrays with elements longer than 255 characters.)

Before perusing some quickie programs to show the effects of tokenizing, I should like to give some credit where it is due. Though I participated in the specifications for Atari BASIC, I had little to do with the actual implementation. More history: Atari asked us (in September, 1978) to bid on producing a custom "consumer-oriented" BASIC for them. Sometime in October, the specifications were finalized and Paul Laughton and Kathleen O'Brien (with a very little help from three more of us) began to work in earnest. The contract called for delivery by April 6, 1979, and included delivery of a File Manager System (DOS 1). Atari planned to take an early, 8K Microsoft BASIC to the Consumer Electronics Show (in Las Vegas) in January, 1979, and then switch later. The actual purchase order took a while to get through Atari's red tape, and the final version thereof is dated 12/28/78 — about one week *after* both BASIC and DOS were delivered to Atari! Atari took Atari BASIC to CES.

Investigating BASIC's Tokens

There are three fundamental types of tokens in Atari BASIC, each of which occupies exactly one byte of RAM memory, with only two special cases. The token types are statement name tokens, operator name tokens (which include function names and some other miscellany), and variable name tokens. The special cases are numeric and string constants, which begin with an operator name token, but are followed by the actual value of the constant.

Statement name tokens can *only* occur as the first item of a statement and, thus, have their own keyword and tokenizing table. In theory, Atari BASIC's structure could support up to 256 types of statements. Variable name tokens and operator name tokens are intermixed throughout the rest of a statement and are distinguished by the state of their upper bit: variable name tokens have their upper bit on, operators don't.

A few of the statement types are also special cased in that they are not followed by operator and variable tokens. These special cases include the obvious REM and DATA and the not-so-obvious ERROR (the statement name given to lines containing a syntax error).

Since each variable is reduced to a single byte (with its upper bit set), there are a maximum of 128 different variable names per program. There is the further implication that BASIC must remember the association of name to token in order to LIST your program back to you. The actual ATASCII names are stored in the "Variable Name Table," and we investigated its structure in **COMPUTE!** #17 under the heading of "VARIABLE, VARIBLE, VARABLE." (Briefly, the names are simply stored one after the other, with the upper bit of the last character of each name turned on.)

The statement and operator names are obviously predefined in the BASIC ROM cartridge, and we offer herewith a program (Program 2) which prints out the token numbers and corresponding keywords. When you run the program, you will notice that some operators (especially the left parenthesis) appear to be repeated. They are. We will find out why next month.

Program 1.

```
A sample device driver for Atari's OS
---- general remarks ----

0000         1000          .PAGE  "---- general remarks ----"
             1010 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             1020 ;
             1030 ; The "M:" driver --
             1040 ;     Using memory as a device
             1050 ;
             1060 ; Includes installation program
             1070 ;
             1080 ; Written by Bill Wilkinson
             1090 ;   for January, 1982, COMPUTE!
             1100 ;
             1110 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

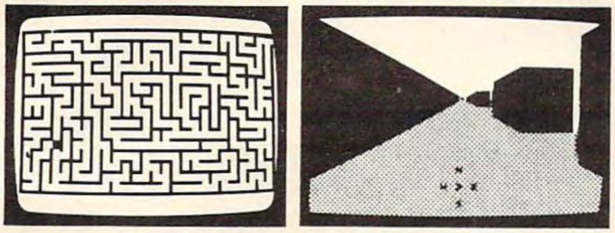
```
              1120 ;
              1130 ; EQUATES INTO ATARI'S OS, ETC.
              1140 ;
034A          1150 ICAUX1  =   $34A      ; The AUX1 byte of IOCB
              1160 ;
0008          1170 OPOUT   =   8         ; Mode 8 is OPEN for OUTPUT
              1180 ;
02E7          1190 MEMLO   =   $2E7      ; pointer to bottom of free RAM
02E5          1200 MEMTOP  =   $2E5      ; pointer to top of free RAM
              1210 ;
00E0          1220 FR1     =   $E0       ; Fltg Pt Register 1, scratch
              1230 ;
0001          1240 STATUSOK =  1         ; I/O was good
0088          1250 STATUSEOF = $88       ; reached an end-of-file
              1260 ;
031A          1270 HATABS  =   $31A
              1280 ;
0100          1290 HIGH    =   $100      ; divisor for high byte
00FF          1300 LOW     =   $FF       ; mask for low byte
              1310 ;

A sample device driver for Atari's OS
The installation routine

0000          1320         .PAGE "The installation routine"
              1330 ;
0000          1340         *=   $1F00
              1350 ; This first routine is simply
              1360 ; used to connect the driver
              1370 ; to Atari's handler address
              1380 ; table.
              1390 ;
              1400 LOADANDGO
1F00 A200     1410         LDX  #0           ; We begin at start of table
              1420 SEARCHING
1F02 BD1A03   1430         LDA  HATABS,X     ; Check device name
1F05 F00A     1440         BEQ  EMPTYFOUND   ; Found last one
1F07 C94D     1450         CMP  #'M        ' ; Already have M: ?
1F09 F01A     1460         BEQ  MINSTALLED   ; Yes, don't reinstall
1F0B E8       1470         INX
1F0C E8       1480         INX
1F0D E8       1490         INX               ; Point to next entry
1F0E D0F2     1500         BNE  SEARCHING    ; and keep looking
1F10 60       1510         RTS               ; Huh? Impossible!!!
              1520 ;
              1530 ; We found the current end of the
              1540 ; table...so extend it.
              1550 ;
              1560 EMPTYFOUND
1F11 A94D     1570         LDA  #'M        ' ; Our device name, "M:"
1F13 9D1A03   1580         STA  HATABS,X     ; is first byte of entry
1F16 A93B     1590         LDA  #MDRIVER&LOW
1F18 9D1B03   1600         STA  HATABS+1,X   ; LSB of driver addr
1F1B A91F     1610         LDA  #MDRIVER/HIGH
1F1D 9D1C03   1620         STA  HATABS+2,X   ; and MSB of addr
1F20 A900     1630         LDA  #0
1F22 9D1D03   1640         STA  HATABS+3,X   ; A new end for the table
              1650 ;
              1660 ; now change LOMEM so BASIC won't
              1670 ; overwrite us.
```

```
                ;
                MINSTALLED
1F25 A900              LDA   #DRIVERTOP&LOW
1F27 8DE702            STA   MEMLO       ; LSB of top addr
1F2A A920              LDA   #DRIVERTOP/HIGH
1F2C 8DE802            STA   MEMLO+1     ; and MSB therof
                ;
                ; and that's all we have to do!
                ;
1F2F 60                RTS
                ;
                ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ; This entry point is provided
                ; so that BASIC can reconnect
                ; the driver via a USR(RECONNECT)
                ;
                RECONNECT
1F30 68                PLA
1F31 F0CD              BEQ   LOADANDGO ; No parameters, I hope
1F33 A8                TAY
                PULLTHEM
1F34 68                PLA
1F35 68                PLA              ; get rid of a parameter
1F36 88                DEY
1F37 D0FB              BNE   PULLTHEM  ; and pull another
1F39 F0C5              BEQ   LOADANDGO ; go reconnect
                ;
```

```
A sample device driver for Atari's OS
The driver itself

1F3B                    .PAGE "The driver itself"
              ;
              ; Recall that all drivers must
              ; be connected to OS through
              ; a driver routines address table.
              ;
              MDRIVER
1F3B 4C1F               .WORD MOPEN-1   ; The addresses must
1F3D 6F1F               .WORD MCLOSE-1  ; ...be given in this
1F3F 921F               .WORD MGETB-1   ; ...order and must
1F41 851F               .WORD MPUTB-1   ; ...be one (1) less
1F43 9F1F               .WORD MSTATUS-1 ; ...than the actual
1F45 491F               .WORD MXIO-1    ; ...address
1F47 4C4A1F             JMP  MINIT      ; This is for safety only
              ;
              ; For many drivers, some of these
              ; routines are not needed, and
              ; can effectively be null routines
              ;
              ; A null routine should return
              ; a one (1) in the Y-register
              ; to indicate success.
              ;
              MXIO
              MINIT
1F4A A001               LDY  #1         ; success
1F4C 60                 RTS
              ;
              ; If a routine is omitted because
              ; it is illegal (reading from a
              ; printer, etc.), simply pointing
              ; to an RTS is adequate, since
              ; Atari OS preloads Y with a
              ; 'Function Not Implemented' error
              ; return code.
              ;
```

```
A sample device driver for Atari's OS
The driver function routines

1F4D                    .PAGE "The driver function routines"
              ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              ;
              ; Now we begin the code for the
              ; routines that do the actual
              ; work
              ;
              MOPEN
1F4D BD4A03             LDA  ICAUX1,X    ; Check type of open
1F50 2908               AND  #OPOUT      ; Open for output?
1F52 F00D               BEQ  OPENFORREAD ; No...assume for input
1F54 ADE502             LDA  MEMTOP
1F57 8DD21F             STA  MSTART      ; We start storing
1F5A ACE602             LDY  MEMTOP+1    ; ...the bytes
1F5D 88                 DEY              ; ...one page below
1F5E 8CD31F             STY  MSTART+1    ; the supposed top of mem
```

CAPTIVITY



the most advanced maze game available to the ATARI® owner

A new game by Craig Patchett which uses machine language routines, 5-color, high-resolution graphics and sound effects.

The player's goal is to guide an imaginary robot through and out of a 3-dimensional maze.

20 game variations insure countless hours of enjoyment.

CAPTIVITY can be used with any **ATARI®** computer with at least 24K. It is available in cassette format only. Price is $24.95 plus $2.00 shipping and handling.

Available at fine computer stores or write us directly.

PDI

Program Design, Inc. 11 Idar Court Greenwich, CT 06830
203-661-8799

ATARI® is the registered trademark of **ATARI**, Inc.

ATARI 400/800 OWNERS: THERE'S A MICROCONNECTION™ FOR YOU! —

Now you can direct-connect to the telephone with or without the 850™ interface, there's also a serial port to drive a printer, optional autodial and autoanswer, and smart terminal software! Prices start at $199.50 For more details write or phone:

the microperipheral corporation

2643A - 151st Pl. N.E.
Redmond, WA 98052
(206) 881-7544

master charge
VISA

SOFTWARE STREET
PRESENTS

ATARI

800 Computer 16k......$729.00
800 Computer 32k......$769.00
800 Computer 48k......$839.00
825 Printer(80 col)..$599.00
810 Disk Drive.......$429.00
850 Intf. Mod........$155.00
822 Thermal Printer..$329.00
820 Printer..........$259.00
410 Recorder..........$65.00
830 Modem............$145.00

Special Software Packages

The Entertainer........$69.00
Star Raiders, Missile Command, and Joysticks.

The Programmer.........$54.00
BASIC Programming language, Self teaching guide to BASIC, and BASIC reference manual

The Educator...........$99.00
410 Recorder, BASIC programming language, and States and Capitals.

TOP SELLING PROGRAMS

Missile Command.........$33.96
Asteroids...............$33.96
Invitation to Prog. 2..$16.96
Invitation to Prog. 3..$16.96
Assembler/Editor........$48.96
Jaw Breaker.............$25.95
Cranston Manor..........$29.95
Dodge Racer.............$19.95
Atari Word Processor..$129.95
Atari World.............$59.95
Star Raiders............$39.96
Intruder Alert..........$14.95
Sammy the Sea Serpent..$14.95

savings from:
Atari
Atari Program Exchange
Adventure International
Crystal
P.D.I.
L.J.K.
Dynacomp
Quality Software
Avalon Hill
Epyx
Computer consultants
Synapse
Datasoft
United Software
On-Line
O.S.S.
Software Street

ATARI IS A REGISTERED TRADEMARK

Software Street
3392 Clipper Dr.
Chino, CA 91710
(714) 591-3061

Software Street is your mail order Atari discount center.

Shipping costs:
Software-minimum $2.00
Hardware-prices will vary (please call)

prices subject to change

```
                2490 ;
                2500 ; now we join up with mode 4 open
                2510 ;
                2520 OPENFORREAD
1F61 ADD21F     2530        LDA   MSTART      ; simply move the
1F64 8DCE1F     2540        STA   MCURRENT    ; ...start pointer
1F67 ADD31F     2550        LDA   MSTART+1    ; ...to the current
1F6A 8DCF1F     2560        STA   MCURRENT+1 ; ...pointer, both bytes
                2570 ;
1F6D A001       2580        LDY   #STATUSOK
1F6F 60         2590        RTS               ; we don't acknowledge failure
                2600 ;
                2610 ;
                2620 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                2630 ;
                2640 ; the routine for CLOSE of M:
                2650 ;
                2660 MCLOSE
1F70 BD4A03     2670        LDA   ICAUX1,X    ; check mode of open
1F73 2908       2680        AND   #OPOUT      ; was for output?
1F75 F00C       2690        BEQ   MCLREAD     ; no...close input 'file'
                2700 ;
1F77 ADCE1F     2710        LDA   MCURRENT    ; we establish our
1F7A 8DD01F     2720        STA   MSTOP       ; ...limit so that
1F7D ADCF1F     2730        LDA   MCURRENT+1 ; ...next use can't
1F80 8DD11F     2740        STA   MSTOP+1     ; ...go too far
                2750 ;
                2760 MCLREAD
1F83 A001       2770        LDY   #STATUSOK
1F85 60         2780        RTS               ; and guaranteed to be ok
                2790 ;
                2800 ;
                2810 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                2820 ;
                2830 ; This routine puts one byte
                2840 ; to the memory for later
                2850 ; retrieval.
                2860 ;
                2870 MPUTB
1F86 48         2880        PHA               ; save the byte to be PUT
1F87 20B51F     2890        JSR   MOVECURRENT ; get ptr to zero page
1F8A 68         2900        PLA               ; the byte again
1F8B A000       2910        LDY   #0
1F8D 91E0       2920        STA   (FR1),Y     ; put the byte, indirectly
1F8F 20C01F     2930        JSR   DECCURRENT ; point to nxt byte
1F92 60         2940        RTS               ; that's all
                2950 ;
                2960 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                2970 ;
                2980 ; routine to get a byte put
                2990 ; in memory before.
                3000 ;
                3010 MGETB
1F93 20A01F     3020        JSR   MSTATUS     ; any more bytes?
1F96 B007       3030        BCS   MGETRTS     ; no...error
1F98 A000       3040        LDY   #0
1F9A B1E0       3050        LDA   (FR1),Y     ; yes...get a byte
1F9C 20C01F     3060        JSR   DECCURRENT ; and point to next byte
                3070 MGETRTS
```

```
1F9F 60      3080         RTS
             3090 ;
             3100 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             3110 ;
             3120 ; check the status of the driver
             3130 ;
             3140 ; this routine is only valid
             3150 ; when READing the 'file'...
             3160 ; "M:" never gets errors when
             3170 ; writing.
             3180 ;
             3190 MSTATUS
1FA0 20B51F  3200         JSR  MOVECURRENT ; current ptr to zero page
1FA3 CDD01F  3210         CMP  MSTOP       ; any more bytes to get?
1FA6 D009    3220         BNE  MSTOK       ; yes
1FA8 CCD11F  3230         CPY  MSTOP+1     ; double chk
1FAB D004    3240         BNE  MSTOK       ; yes, again
1FAD A088    3250         LDY  #STATUSEOF ; oops...
1FAF 38      3260         SEC              ; no more bytes
1FB0 60      3270         RTS
             3280 ;
             3290 MSTOK
1FB1 A001    3300         LDY  #STATUSOK ; all is okay
1FB3 18      3310         CLC              ; flag for MGETB
1FB4 60      3320         RTS

A sample device driver for Atari's OS
Miscellaneous subroutines

1FB5         3330         .PAGE "Miscellaneous subroutines"
             3340 ;
             3350 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             3360 ;
             3370 ; finally, we have a couple of
             3380 ; short and simple routines to
             3390 ; manipulate MCURRENT, the ptr
             3400 ; to the currently accessed byte
             3410 ;
             3420 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             3430 ;
             3440 ; MOVECURRENT simply moves
             3450 ;  MCURRENT to the floating
             3460 ;  point register, FR1, in
             3470 ;  zero page.  FR1 is always
             3480 ;  safe to use except in the
             3490 ;  middle of an expression.
             3500 ;
             3510 MOVECURRENT
1FB5 ADCE1F  3520         LDA  MCURRENT
1FB8 85E0    3530         STA  FR1         ; notice that we use
1FBA ACCF1F  3540         LDY  MCURRENT+1 ; both the A and
1FBD 84E1    3550         STY  FR1+1       ; Y registers...this
1FBF 60      3560         RTS              ; is for MSTATUS use
             3570 ;
             3580 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
             3590 ;
             3600 ; DECCURRENT simply does a two
             3610 ;  byte decrement of the MCURRENT
             3620 ;  pointer and returns with the
```

```
              3630 ;  Y register indicating OK status.
              3640 ; NOTE that the A register is
              3650 ;  left undisturbed.
              3660 ;
              3670 DECCURRENT
1FC0 ACCE1F  3680        LDY  MCURRENT   ; check LSB's value
1FC3 D003    3690        BNE  DECLOW     ; if non-zero, MSB is ok
1FC5 CECF1F  3700        DEC  MCURRENT+1 ; if zero, need to bump MSB
              3710 DECLOW
1FC8 CECE1F  3720        DEC  MCURRENT   ; now bump the LSB
1FCB A001    3730        LDY  #STATUSOK  ; as promised
1FCD 60      3740        RTS

A sample device driver for Atari's OS
RAM usage and clean up

1FCE          3750        .PAGE "RAM usage and clean up"
              3760 ;
              3770 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              3780 ;
              3790 ; END OF CODE
              3800 ;
              3810 ;
              3820 ; Now we define our storage
              3830 ; locations.
              3840 ;
              3850 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              3860 ;
              3870 ;
              3880 ; MCURRENT holds the pointer to
              3890 ; the next byte to be PUT or GET
1FCE 0000     3900 MCURRENT .WORD 0
              3910 ;
              3920 ; MSTOP is set by CLOSE to point
              3930 ; to the last byte PUT, so GET
              3940 ; won't try to go past the end
              3950 ; of data.
1FD0 0000     3960 MSTOP  .WORD 0
              3970 ;
              3980 ; MSTART is derived from MEMTOP
              3990 ; and points to the first byte
              4000 ; stored.  The bytes are stored
              4010 ; in descending addresses until
              4020 ; MSTOP is set by CLOSE.
1FD2 0000     4030 MSTART .WORD 0
              4040 ;
              4050 ; DRIVERTOP becomes the new
              4060 ; contents of MEMLO
2000          4070 DRIVERTOP = *+$FF&$FF00
              4080 ; (sets to next page boundary)
              4090 ;
              4100 ;
              4110 ; The following is how you make
              4120 ; a LOAD-AND-GO file under
              4130 ; Atari's DOS 2
              4140 ;
1FD4          4150        *=    $2E0
02E0 001F     4160        .WORD LOADANDGO
              4170 ;
```

```
           4180 ;
02E2       4190        .END
```

A sample device driver for Atari's OS
RAM usage and clean up

=034A	ICAUX1	=0008	OPOUT	=02E7	MEMLO	=02E5	MEMTOP
=00E0	FR1	=0001	STATUSOK	=0088	STATUSEOF	=031A	HADABS
=0100	HIGH	=00FF	LOW	1F00	LOADANDGO	1F02	SEARCHING
1F11	EMPTYFOUND	1F25	MINSTALLED	1F3B	MDRIVER	=2000	DRIVERTOP
1F30	RECONNECT	1F34	PULLTHEM	1F4D	MOPEN	1F70	MCLOSE
1F93	MGETB	1F86	MPUTB	1FA0	MSTATUS	1F4A	MXIO
1F4A	MINIT	1F61	OPENFORREAD	1FD2	MSTART	1FCE	MCERRENT
1F83	MCLREAD	1FD0	MSTOP	1FB5	MOVECURRENT	1FC0	DECCURRENT
1F9F	MGETRTS	1FB1	MSTOK	1FC8	DECLOW		

Program 2.

```
100 REM listing of a program to print token values
101 REM and their ATASCII equivalents
200 ? "The STATEMENT Token List" : ?
210 ADDR = 42161 : SKIP = 2 : TOKEN = 0
220 GOSUB 1000 : REM call the token printer
300 ? "The OPERATOR Token List" : ?
310 ADDR = 42979 : SKIP = 0 : TOKEN = 16
320 GOSUB 1000 : REM again call to print tokens
400 END

1000 REM Subroutine to print a keyword table
1001 REM On entry:
1002 REM   ADDR = the address of the keyword table
1003 REM   SKIP = number of bytes to skip
1004 REM          between keyword strings
1005 REM  TOKEN = the starting token number for
1006 REM          this table
1007 REM
1050 IF NOT PEEK(ADDR) THEN ?:?:RETURN

       [note: both tables end with a zero byte]

1060 PRINT TOKEN, : REM the token number
1100 REM Print the ATASCII string for this token
1110 BYTE = PEEK(ADDR) : ADDR = ADDR+1
1120 IF BYTE < 128 THEN ? CHR$(BYTE); : GOTO 1100
1130 PRINT CHR$(BYTE-128) : REM last character
       in keyword has upper bit on
1140 ADDR = ADDR + SKIP : REM an address for stmts
1150 TOKEN = TOKEN + 1  : REM to next keyword
1160 GOTO 1000
```

©

THE OSI® GAZETTE

Part I:

A Small Operating System: OS65D The Disk Routines

T. R. Berger
Coon Rapids, MN

Editor's Note: In this first part of a two-part series, Mr. Berger presents valuable information for all disk drive users. The article concludes next month with a memory map of the disk routines and flowcharts for all the major subroutines. — RTM

In this article I will examine the disk routines in OS65D (V3.2 NMHZ). To understand these subroutines, it is neither necessary to know precise details about the physical functioning of a disk drive, nor to know about various methods of storing data on a diskette. However, such background makes it easier to understand what is involved in an operating system, and why certain processes are done as they are. There are several articles [1-3] which offer very good general descriptions of disk drives. Further, manufacturers' drive manuals usually give fairly complete descriptions of individual drives. I only discuss those aspects which are immediately applicable to the functioning of OS65D.

The Disk Drive

The typical diskette looks as in Figure 1. A magnetically coated round piece of plastic is enclosed in a protective cardboard envelope which has an inner, slippery plastic liner. The hub of the disk drive engages the large hole in the middle of the plastic diskette causing it to spin very rapidly inside its envelope. There is a long slot in the envelope enabling the head of the drive to make contact with the plastic diskette.

Imagine a large number of concentric circles drawn on the plastic diskette so that part of each circumference is visible through the slot. We call each circle a track on the diskette. When the diskette is in the drive, the head is precisely positioned under one of these circular tracks, and contact is made with the diskette. The spinning of the diskette causes this track to continually pass over the head. If we imagine the track to be a continuous loop of magnetic cassette recorder head, we can appreciate how one might store data on the disk. If we envision each track as being a different loop of tape then we can begin to see the power of a disk drive.

In some minifloppies, inserting the diskette and closing the drive door brings the head into contact with the diskette. On other drives, there is a little lever with a soft pad attached directly above the head which is below the diskette. On drives with such a lever, there is a switch which causes this slapping. Turn the switch on and the head engages the diskette; turn it off and the head loses contact with the diskette.

The head can slide back and forth along the long slot in the diskette accessing all the concentric tracks on the diskette. This sliding motion is generally accomplished in one of two ways. The head may be on a screw. Spinning the screw one way or another moves the head in or out. The head may be on a flat metal band which is looped over some shafts, or it may be on a wire which is wound around some shafts. Spinning a shaft causes the head to move. The slide rule dial on most radios works by a similar principle: i.e. the dial pointer is mounted on a string strung over pulleys and wound around the tuning knob shaft. Twisting the knob moves the pointer across the dial. Thus the back and forth motion of the disk head is caused by the turning motion of a motor shaft.

Since the tracks on a disk are very close together, the motor only needs to spin a small fraction of a revolution in order to move the head one track. Very special motors called stepper motors are used for this purpose. When the motor is pulsed, it spins a fixed fraction of a revolution then stops. If pulsed again, it will spin that same fraction again. Clockwise or counterwise motion of the motor shaft translates into back or forth motion of the disk head. Consequently, there are two switches which control this motor: one to determine direction, the other an ON/OFF switch. If we set the direction switch as desired then flick the ON/OFF switch first to on then to off, the disk will move one track.

If we have a memory location in the computer which tells us the track number (say, Track 27) on

OSI TRS-80 COLOR-80 OSI

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 OSI

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. OSI

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) 4 back issues and subscription for 2 additional issues - $9.00.

ADVENTURES!!!

For OSI, TRS-80, and COLOR-80. These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch".)

Adventures require 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 each.

ESCAPE FROM MARS (by Rodger Olsen)

This ADVENTURE takes place on the RED PLANT. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)

This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

TREK ADVENTURE (by Bob Retelle)

This one takes place aboard a familiar starship. The crew has left for good reasons - but they forgot to take you, and now you are in deep trouble.

DEATH SHIP (by Rodger Olsen)

Our first and original ADVENTURE, this one takes place aboard a cruise ship - but it ain't the Love Boat.

VAMPIRE CASTLE (by Mike Bassman)

This is a contest between you and old Drac - and it's getting a little dark outside. $14.95 each.

OSI NEW-NEW-NEW OSI
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 OSI

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 (5¼") OSI

BARE BOARDS FOR OSI C1P

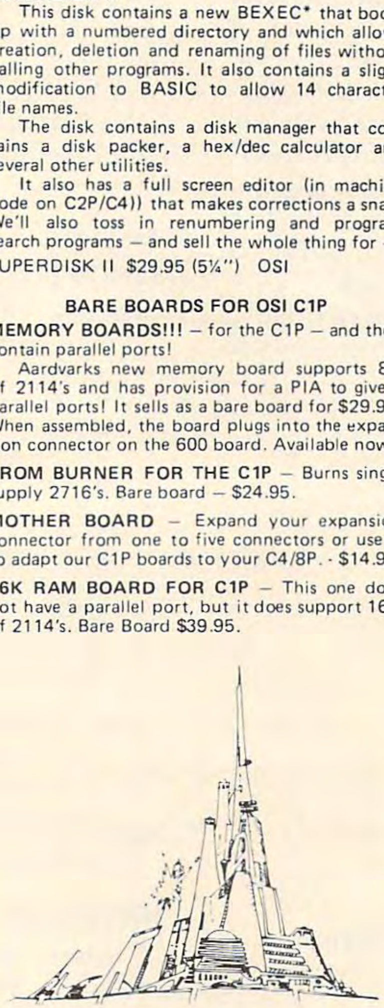
MEMORY BOARDS!!! — for the C1P — and they contain parallel ports!

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

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

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

16K RAM BOARD FOR C1P — This one does not have a parallel port, but it does support 16K of 2114's. Bare Board $39.95.



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. Specify 5¼ or 8" disk.

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 OSI

ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)

TIMETREK - A REAL TIME, REAL GRAPHICS STARTRECK. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! $14.95.

STARFIGHTER - This one man space war game pits you against spacecruisers, battlewagons, and one man fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Another real time goody. $9.95

BATTLEFLEET - This grown up version of Battleship is the toughest thinking game available on OSI or 80 computers. There is no luck involved as you seek out the computers hidden fleet. A topographical toughie. $9.95

QUEST - A NEW IDEA IN ADVENTURE GAMES! Different from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gather men and supplies by combbat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time.

16K COLOR-80 OR TRS-80 ONLY. $14.95

Please specify system on all orders

This is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSI systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.

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

OSI COLOR-80

which the head is currently positioned, and we move the head outward one track on the diskette (the outermost track is Track 0) then we may decrease the number in memory by one (to Track 26). In other words, we may move from any track to any desired track just by stepping and counting. A single step occurs very rapidly, more than 100 steps per second are usually possible. Of course, this stepping method will only work if we know the current track number on which the head is located.

Most drives have a special indicator to tell when the head is positioned over the outermost track (Track 0). Moving the head out until this indicator comes on allows us to set a track counter to 0 precisely when the head is at Track 0.

When the head is down on a particular track, several operations are possible. The head can read (playback) data from the diskette, or it can write (record) data on the diskette. In addition, an erase function can be switched on. If we erase only, the track will be erased. However, if we erase and write at the same time, the erase function narrows the data stream keeping it from widening into neighboring tracks. The disk has a switch which causes the head to write if on and read if off. An additional switch turns the erase function on and off.

If you look down on some spinning circular object (e.g. a turning phonograph record), you will see that the outer edge is moving much more quickly than any inner part. In particular, on a diskette, each track moves at a different speed past the head. These radical changes in head speed from inner to outer tracks pose difficult problems in obtaining uniform recordings on all tracks of the diskette. Some drives compensate by having two possible recording levels: one for inner tracks, the other for outer tracks. A switch is needed to move between these two modes.

If you own more than one drive, there are switches which allow the computer to select any one of these drives.

In Table 1, under CONTROL LINES, you will see that the computer has a bit to control each of the switches just described. Other than a serial port through which data flows and its associated control location, these are all the control lines used by OS65D to run the disk.

As already mentioned, there are also STATUS LINES to the computer which indicate current conditions at the disk. There is an indicator to tell

Table 1.

DISK STATUS LINES

PA0	DRIVE 1 READY
PA1	HEAD AT TRACK 0
PA2	FAULT INDICATOR
PA3	SECTOR HOLE
PA4	DRIVE 2 READY
PA5	DISK WRITE PROTECTED
PA7	INDEX HOLE

DISK CONTROL LINES

PB0	ENABLE WRITE FUNCTION
PB1	ENABLE ERASE FUNCTION
PB2	STEP MOTOR DIRECTION (IN)
PB3	STEP MOTOR ON (OFF)
PB4	FAULT RESET
PA6	DRIVE ½ SELECT
PB5	DRIVE ½ SELECT
PB6	SET HEAD RECORD CURRENT TO LOW
PB7	PUT HEAD ONTO DISKETTE

The disk PIA has two ports 'A' and 'B'.

PORT A $C000 (with bits PA0-PA7)
PORT A CONTROL REGISTER $C001
PORT B $C002 (with bits PB0-PB7)
PORT B CONTROL REGISTER $C003

The disk has an ACIA

SERIAL PORT $C011
STATUS/CONTROL REGISTER $C010

OS65D configures this port for 8 bit bytes with even parity and 1 stop bit ($58).

Table 2.

8 INCH FLOPPY TIMING

#Sectors	Total Pages	Pages Last Sector	Time	DT
1	13	13	162768	3900
2	13	10	166203	464
3	13	10	166638	29
4	12	1	163209	3458
5	12	1	163144	3023
6	12	1	164079	2588
7	12	1	164514	2153
8	12	1	164949	1718
9	12	1	165384	1283
10	12	1	165819	848
11	12	1	166254	413
12	12	1	166689	-22

t(us.) = 8101 + 12864xp - 1000xr + 435xn

p = number of pages in track
r = number of pages in last track
n = number of sectors
166667 us. = time on one track
DT = time left on track

MINIFLOPPY TIMING

# Sectors	Total Pages	Pages Last Sector	Time	DT
1	8	8	193986	6014
2	8	3	199641	359
3	8	4	199296	704
4	8	4	199951	49
5	7	1	179478	20522
6	7	1	180133	19867
7	7	1	180788	19212
8	8	1	205571	-5571

t(us.) = 8307 + 24128xp - 1000xr + 435xn

200000 ux. = time on one track

if a drive is ready (i.e. if the drive door is closed indicating a diskette is mounted and ready). There may be an indicator to tell if a diskette is write protected. Finally there is an index hole detector. These indicators are all listed in Table 1 under STATUS LINES. You will see a few more than mentioned here. These are not used by OS65D.

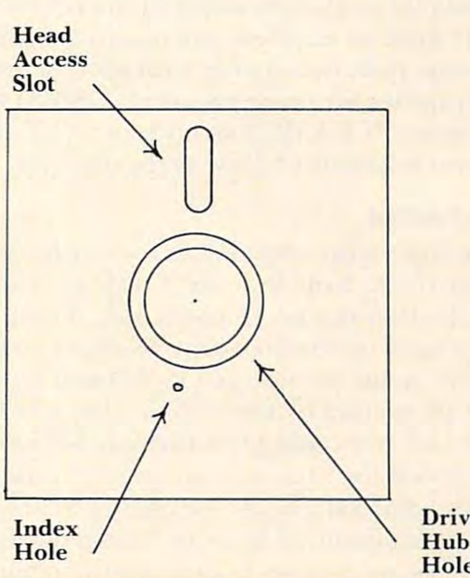
Let's examine the function of the index hole a little more closely. In Figure 1 you will see a small, off-center hole punched in the diskette. (It is off center to prevent functioning if the diskette is inserted into the drive wrong side up.) As the diskette spins, the drive detects when this hole passes over a special indicator. This passage marks the beginning of a track. To find the beginning of a track, the computer moves the head to a track, puts the head on the diskette, and waits for the index hole to flash by.

Once the index hole has passed, the data format on the diskette becomes important. The few methods for encoding data magnetically on the diskette are standardized and adhered to by almost all drive manufacturers. Thus one bit sent to BRAND X drive will be recorded in just about the same way as one bit sent to BRAND Y (i.e. the bit will be recorded in one of about three standard ways). There are a few exceptions to this rule.

This standardization allows computer manufacturers to use drives from different disk manufacturers on the same computer. OSI supplies computers with Shugart, Siemans, and other drives without explicitly telling the buyer which drives he is getting.

Most computer manufacturers send bits from their computers to disk drives as a steady stream of bits, eight bits per byte, and a fixed number of bytes per stream. At the end of a stream are two more bytes called a checksum of cyclic reduncancy

Figure 1.



check (CRC). These two bytes are usually the sum of all the previous bytes in the stream. On reading the stream, the checksum can be recomputed from the stream and compared with the checksum recorded on the diskette at the end of the stream. If there is a mismatch, an error has occurred somewhere in the stream.

OSI does not follow this format. They treat disk communication as an asynchronous communication line. In other words, except for the speed of the bits, the computer sends bits to the disk drive in the same way it sends bits to a modem: through a special serial port called an Asynchronous Communication Interface Adapter (for short, an ACIA, UART, or just serial port). OS65D requires 11 bits to be recorded on the diskette for each eight-bit byte. The first bit is a start bit indicating that the byte is beginning. The next eight bits (bits 2-9) are the actual data byte. The tenth bit is a parity bit indicating whether the byte contains an even or an odd number of value one bits. The last bit is a stop bit indicating the end of the byte.

The disadvantages of this method are twofold. First, it is nonstandard. OSI owners cannot interchange disks made by computers of other manufacturers. Second, OSI can store only 8/11 as much on a disk as other manufacturer's computers.

The advantages are reliability and simplicity. An inexpensive ACIA performs many chores simplifying software and hardware. No cyclic redundancy checks are needed. Each byte can be individually checked for an error by the ACIA. If there is a disk error, usually all but a few bytes can be recovered correctly using the EXAMINE command of OS65D. Other systems make recovery much more difficult. A bit error can cause all bits in a stream to shift by one. In other words, bit two of a byte may be read as bit one, and bit zero of a byte may be read as bit seven in the previous byte. OS65D does an excellent job of error detection. It is a shame that, in a system with such excellent opportunities for error recovery, OS65D has absolutely none. If BASIC encounters a disk error, a program stops with a terse error message.

Track Format

Figure 2 gives the actual data format for an OS65D diskette track. Note that the Track 0 format differs from all other tracks. In particular, Track 0 can only be used by the bootstrap ROM. Track 0 contains the major portion of OS65D and is given added protection by this scheme, but I believe OSI blundered in choosing this format. All tracks should be recorded the same way to maximize flexibility.

The data on a track commences 1 ms. past the index hole (about 23 bytes in time at 44 us./byte). Two bytes are written to indicate the beginning of a track. The bytes should be carefully chosen so as to be an unusual combination. OS65D always writes $43 then $57. When the track is read, reading does not commence until the $43 and $57 have been found. A simple encryption method would be to change these bytes. Since the EXAMINE command will even read such a track, this encryption is not terribly secure. OS65U uses different bytes, so OS65U tracks cannot be read by OS65D without minor changes to the operating system.

Figure 2.

FORMAT FOR TRACKS (>0)

Index Hole | 1ms. | $43 | $57 | Track # | $58 | 6615 us. ...

... $76 | Sector # | # Pages in Sector | That many Pages of Data ...

... $47 | $53 | Intersector Wait Time ... Repeat for each Sector

FORMAT FOR TRACK 0

Index Hole | 1ms. | Load vector high | Load vector low | #Pages ...

... That many Pages of Data ...

Next the track number is written in binary coded decimal (BCD). This recorded value is always compared with the stored track number in memory to make certain the head is positioned on the correct track. Then a stop byte ($58) is recorded on the disk (this byte is never checked on a read).

This data constitutes the Track Header. On Initialization, a track is erased then the Track Header is written on the Track. This Track Header is not rerecorded at any future read or write.

There is a lull after the Track Header of just under 6.6 ms. (about 149 bytes). This time differs greatly from the time given in the OS65D GUIDE. You will see why in the following discussion.

During a sector seek operation, a "previous sector" length number p is saved. This value is set to four if we seek Sector 1 (otherwise the "previous sector" length number would be zero, which is not allowable). Then a subroutine waits px800 µs. The OS65D GUIDE says that between Sector N and Sector n + 1 there is a gap of px800 µs. This is not quite correct. After the end of a sector, OS65D waits quietly for px800 µs. The write function is then switched on. A further 185 µs. is allowed to pass. Then the erase function is switched on. We now wait an additional px800 µs. before starting to write data. In other words, the time from the last byte of sector n to the first byte of sector n + 1 is about px1600 + 185 µs. For Sector 1, p is taken to be four. In all other cases, p is the length (in pages, i.e. multiples of 256 bytes) of the "previous sector."

This description requires modification. It applies to systems with a 1 MHZ clock. On cold start, OS65D measures the timing on a serial port to calculate the clock speed. (Remember, a 300 baud port must remain 300 baud no matter what the clock speed.) Then a timing constant in the 1 ms. subroutine is set. However, this calculation does not affect the 100 us. routine used in sector spacing. (I assume this clock versatility is the reason for the NMHZ in the title of this version of OS65D.) In other words, the 100 us. routine is really a 100/T us. routine where T is the clock speed in MHZ.

This calculation accounts only for the wait loops in intersector timing. In addition, there is quite a bit of inline code which adds to intersector timing. This timing can be calculated. A crude estimate would be to add an additional 30 µs. after each sector. In other words, the sector spacing is

(px1600 + 215)/T

where T is the clock speed in MHZ, and p is the number of pages in the preceding sector. Your disk does not necessarily write diskettes identically with mine, though either computer should read the other's diskettes.

All of this says there is some kind of empty space between the end of the Track Header and the start of Sector 1. Each sector is completely rewritten each time it is addressed in a write operation. A sector is written as follows.

We put a sector start code ($76) on the disk. Next comes the sector number s, then the sector length p in pages (each page is 256 bytes). The smallest unit of disk storage in OS65D is one page. The sector number s is verified on a read operation with the value in memory. The sector length is used on read to calculate the number of bytes to load from the disk.

Now comes the actual data. The amount of data is px256 bytes where p is the number of pages in the sector. After this data comes two end check bytes ($47, $53) marking the end of a sector. Thus the sector if 5 + px256 bytes long. The gap between sectors has already been described. Each succeeding sector follows the same format. This format is pictured in Figure 2. This discussion does not apply to Track 0.

Before discussing Track 0, let's make a few calculations. We assume we have 8" floppies and a 1 MHZ clock (this latter enters in only for the timing between sectors). We discuss how many and what kind of sectors may be put on a track. The discussion is important for the following reason: on a write operation, OS65D checks for the index hole when seeking a Track Header. This keeps the computer from "hanging" on uninitialized tracks (i.e. tracks without a Track Header). In writing sector n, the computer must read the preceding sectors 1,2,...,n-1. For each of these, while the computer is searching for the sector start code, it also watches for the index hole to come around again (also to avoid "hanging" on a sector seek). After the start of the preceding sector, the computer no longer checks to see if we pass the index hole. The reason for this is simple. At 1 MHZ with 8" floppies there is just not enough time between input or output bytes from the disk to check for the index hole and to do all the other operations required during a read or write operation.

If the index hole passes, we are back to the beginning of the track. If 1 ms. passes, we're over the Track Header again. Obliterating the Track Header destroys the readibility of the Track. Experienced programmers may salvage matters using the EXAMINE command, but this is not a task you want to face. *Moral:* Don't pass the index hole a second time on a write operation.

If you're not a whiz at algebra, skim over this part until we start drawing conclusions.

We wish to derive a formula for the time from the index hole to the time the head stops writing on the diskette after sector n. If this time occurs before a second appearance of the index hole, then n sectors will fit on a track. We must account for all the time from the first appearance of the index hole until the write function is switched off after the last sector.

The disk spins at 360 rpm. Thus one revolution takes 166,667 µs. The disk data clock runs at 250 KHZ. In particular, each bit takes four µs. Since an OSI byte uses 11 bits, 44 µs. are required per byte. If we could pack a track, this means we could fit 3,787 bytes on a track. But a track is not packed. It is formatted, and we must calculate the formatting time.

We use 1000 us. from the index hole to the Track Header. The Header is four bytes long using 176 us. more. As we have seen, from the Track Header to the start of Sector one, we use 4x1600 + 215 or 6615 us. In particular, 7791 µs. are spent between the index hole and the start of Sector one.

Each sector contains an integral number of pages. Thus, all sectors contain, as an aggregate, *p* pages. Each byte takes 44 us. and there are 256 bytes per page. Thus all these pages account for 11264xp us.

Each sector has five extra bytes. Thus, for *n* sectors, we have 220xn µs.

Next we must account for all the wait time after each of the *n* sectors. Recall that the wait from one sector to the next is qx1600 + 215 µs. where *q* is the number of pages in the preceding sector. Since we assume *n* sectors are on a track, there are only *n*-1 spaces between *n* sectors. If the last sector has *r* pages, then the preceding *n*-1 contain *p-r* pages altogether. Thus, the total inter-sector wait time is 1600x(p-4) + 215x(n-1) µs.

Finally, we must account for the time after the last sector is written until the write and erase functions are switched off. Write and erase continue for 600xr us. after the last byte is written. Then write is switched off and erase continues for 525 µs. more before it too is switched off. This total trailing time is 525 + 600xr µs.

By adding all our derivations, we can make the following statement. For 8" floppies with a one MHZ clock, the total recording time for *n* sectors is

$$t(\mu s.) = 8101 + 12864xp - 1000xr + 435xn$$

where *p* is the total number of pages of data in the sectors and *r* is the number of pages in the last sector.

Remember, OS65D must run on all OSI machines, so this formula gives the "worst case" which must always be satisfied. In Table 2 you will see a few 'upper limit' values tabulated (dt gives the 'time remaining' in the track).

Recalculate *t* for your system. A minifloppy spins at 300 rpm. and the data clock is 125 KHZ. Experiment with a few values for *n* and *p* in the formula. Try actually recording this amount on a disk. Be sure to use an empty diskette track. What is wrong with filling the blank space between the index hole and the Track Header with data? (Think about $43, $57.)The maximum allowable number (plus one) of pages per sector in OS65D is stored in $27ED. You may wish to change this for your experiments.

Notice that OSI recommends a maximum of 13 sectors when only one sector is written on a track, and eight sectors (12 sectors in early GUIDES) if more than one sector is written on a track. The early GUIDE value is "just barely wrong." The later value is obviously a shot in the dark meant to be conservative. It is probably the case that many drives would accept 12 single page sectors in a track. But even 11 sectors, including 12 pages, leaves very little room for errors. ©

THE PET® GAZETTE

A Yuletide Tale

Editor's Note: We recently received the following query letter from a Mr. C. Pickins. While we will not be able to accept C's fictional endeavors, we felt his timely outline might be of interest to our PET/CBM readers. We have put C (not his real name surely) in touch with Dr. Chip. Perhaps we'll see the fruit of later joint collaboration. — RCL

Dear Mr. Lock,

I have this great story outline that I thought **COMPUTE!** might like to follow up. It's just that I don't know if you publish fiction submissions.

It's this heartwarming story about a fellow called Scrooge Tramiel, who runs a pet shop in old London, or California, or Philadelphia or somewhere (funny, the location seems to shift every time I think of it). Anyway, he exhorts his lowly clerk, a fellow by the name of Cratchit Finke, to work through the holidays on a new computer system to be called the Humbug III, and leaves to go home.

Well, what should happen but this guy runs across the ghost of his former partner, Jacob Peddle, who rattles chains and chips and emits fearful moans in all directions. The upshot of this visitation is that Scrooge is going to be visited by three more spectres: the barrister of Christmas past, the solicitor of Christmas present, and the lawyer of Christmas future.

Faster than you can say, "restraining order", the Christmas past fellow pops up and reminds Scrooge of how helpful he used to be to others. "Spirit, why do you torture me so?" wails Scrooge. "Yes, I gave that young fellow a chip to play with ... and he promptly founded Apple Computers with it. Indeed, Radio Shack got its start in the time period between when I announced my computer and started delivering it."

Just then, the digital clock beeps and along comes Christmas present. Not a Christmas present, you understand; just the Ghost of a Christmas present, who shows retailers warming themselves over the glow of their CRT screens. "Everybody else's model three has failed," they seem to be saying, "will Scrooge come through for us?"

An announcement of the digital cuckoo clock heralds visitor number three, the spirit of Christmas future. The screen swirls uncertainly ... coughing and gasping, Scrooge peers through the orange smoke ... is that IBM gaining credibility? What's going on here? The Spirit intones, "I see unused joysticks by the fireplace..." But begging and pleading

and threats of countersuits reveal that it ain't necessarily so ... there's still time to reform.

Dawn is breaking. Maybe the light is dawning, too, for Scrooge rushes over to the window and shouts at a passing boy, "Bring me the biggest goose you can find! My competitors have all the turkeys!"

And the story ends with a traditional Christmas scene ... as Tiny Tim says, "God help us, every one!".

Whaddya say, Mr. Lock? Do you think you can use the story? ©

Renumbering An Appended Routine Only

Elizabeth Deal
Malvern, PA

There is a way to append a program to another in the PET even if the line numbers are out of order. It will be shown here for the upgrade ROM tape system. It should work on other PETs. The scheme uses the Toolkit™ or its equivalents.

Suppose that program A exists in the PET and that its line numbers range from 100 to 2000. Suppose, further, that we would like to append a program B with line numbers which are lower than (or overlap) those of program A: for instance 15 to 340. As long as program B contains no GOTOs and no GOSUBs, one renumbering of the entire A-plus-B package will set the line numbers in order. Consequently, target addresses in program A will remain meaningful.

When, however, program B contains GOTOs and GOSUBs, we are in violation of the "appended program must have higher line numbers" rule. And that means save one piece, put the other one in, renumber it, save again, load again ... ad infinitum. Disk people can do it in a jiffy. It's tough for tape owners though.

Some rules just beg to be broken and this is one of them. A simple solution consists of temporarily hiding program A from PET's view by swapping some pointers around. Just before loading program B we tell the PET that the BASIC area begins at the end of program A or *exactly* two (2) bytes back from the start of variables pointer (42-43). We do this carefully by use of the Machine Language Monitor where we replace contents of \$28-29 with contents of \$2A-2B minus 2. Or by these direct BASIC commands:

```
AD=(PEEK(42)+PEEK(43))-2: AH%=AD/256
POKE40,AD-AH%*256:POKE41,AH%
```

This has to be entered correctly the first time or things get somewhat messy.

At this point we can append program B. It will be placed, in the usual manner, at address AD. We can list this program. And we can RENUMBER it, for instance, with 3000,10 parameters sent to the TOOLKIT.

To finish the process we reset the start of BASIC pointer to its original value, decimal 1025, hex \$0401, or whatever other number we have jotted down in case of being in a partition. In BASIC, the reset can be done by:

```
POKE40,1:POKE41,4
```

Using the Monitor, the reset to 1025 decimal is done by putting \$01 into \$0028 and \$04 into \$0029.

Program A reappears on the scene and the entire package is ready for use.

Pointer addresses for various releases:

	Original		Upgrade and 4.0	
Start of Basic low byte	122	\$7A	40	\$28
high byte	123	\$7B	41	\$29
Start of variables low byte	124	\$7C	42	\$2A
high byte	125	\$7D	43	\$2B

©

chips...chips...chips...chips...chips...chips...

Limited Time: FREE Disk-O-Pro.

Special: You can get this Disk-O-Pro **absolutely free**... when you order the new Commodore 2031 single disk drive from Skyles. (See page 8.)

Command-O or Command-O-Pro?

It's called the Command-O-Pro in Europe, Command-O in the U.S.

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 disk file manipulating. (And our technical writer dug up 39 *additional commands in the course of doing a 76-page manual!*)

Extends Commodore's 8032 advanced screen editing features to the ultimate. **SCROLL** up and down, insert or delete entire lines, delete the characters to the left or right of the cursor, select **TEXT** or **GRAPHICS** mode or ring the PET's bell. Redefine the window to adjust it by size and position on your screen. And define any key to equal a sequence of up to 90 key strokes.

Resides in hexadecimal address **$9000**, the rightmost empty socket in 4016 and 4032 or the rearmost in 8032. If space conflicts, Socket-2-ME available at a very special price.

Command-O from Skyles Electric Works $75.00
Complete with **Socket-2-ME** 95.00

Disk-O-Pro ROM:

25 new commands at $75.00, that's only $3.00 a command!

The powerful chip for all "classic" PETs. In any PET with Version III (BASIC 2.0) ROMs (### **COMMODORE BASIC** ###) Disk-O-Pro 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 program appending (**MERGE**), program 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 3 commands.*

Plus softtouch key (**SET**) which allows you to define a key to equal a sequence of up to 80 keystrokes; **SCROLL** whereby all keys repeat as well as slow scrolling and extra editing features; **BEEP** which allows you to play music on your PET.

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. We have never abandoned a PET owner.)

Complete with 84-page manual written by Greg Yob.

Disk-O-Pro from Skyles Electric Works $75.00
Complete with **Interface Board** (for "Classic" PETs) 95.00

"The PET with the new PicChip...is like home movies."

The new ROM that took Europe by storm is available only from Skyles Electric Works in the U.S. and Canada.

PicChip, a ROM extension of the BASIC Version III (BASIC 2), BASIC 4.0 or BASIC 8032 interpreter. Over 40 commands that allow you to create programs with dynamic graphics displays: plots, bar graphs, pictures; and rolling, scrolling, shifting and inverting. All instantly and easily added to your BASIC program.

The address for the 2000/3000 (which would require PicChip module PC2), for the 4000 (PC4) and for the 8000 (PC8) is **$A000**...unless you have a Mikro, Word Pro 3 or 4, or Jinsam, which occupy that same address. In those cases, you will need the PicChip on an interface board... for the 2000/3000 series, PCB2, for the 4000, PCA4, and for the 8000, PCA8. In all cases, the Mikro or WordPro would be switchable manually using the Skyles Socket-2-ME...available at a very special price.

PicChip from Skyles Electric Works $60.00
(Please indicate PC2, PC4, PC8)
Complete with Socket-2-ME 80.00
(Please indicate PCB2, PCA4, PCA8)

The New Mikro:

Now you can program in machine language as easily as writing BASIC.

4K machine language assembler ROM plugs into your main board. Does all the machine language work for you; all you have to do is start laying down the code.

Retains all the great screen editing features of the PET... even all the Command-O or Toolkit commands. Write your own machine language subroutine. The program you write is the source code you can save. Machine language monitor saves the object code. Not as professional as the Skyles MacroTeA...not as expensive, either.

A great learning experience for those new to machine language programming but who want to master it easily. Superb 50-page manual—including brief introduction to machine language programming—included. But we also recommend several of the books in the Skyles Book Barn.

Skyles Mikro Machine Language Assembler $80.00
MK 2/3 for 2000 or 3000 Series PETs
MK 4/8 for 4000 or 8000 Series PETs and CBMs
Complete with **Socket-2-ME** 95.00

Skyles Catalogue Page 7

For PET and CBM owners only:

This is just 1 of 20 pages of the newest and biggest Skyles catalog, hot off the press.

We know you'll want this page, in its full 8½ x 10 splendor, and another 19 pages of peripherals, software and books that will make your PET or CBM computer even nicer to live with. So, if we missed sending you your very own copy within the last few weeks, call us at **(800) 227-9998** (unless you live in California, in which case call **(415) 965-1735**

From Skyles Electric Works, the oldest and largest professional specialists in the business.

Skyles Electric Works
231-E S. Whisman Road

BRANCH NEVER And QUIF Assembling On SuperPET

Richard Mansfield
Assistant Editor

Ever hear of QUIF? Or HI, ISUPPER, STOI, FSEEK, TABLELOO, COMA, ORB, PULB, SEX, COMB, or BRA? These are some of the 6809 mnemonics, utilities library macros, and "structured programming" statements available to you when you assemble on the SuperPET. The Waterloo 6809 Assembler permits machine language programming which is somewhat like programming in higher level languages. Along with the Assembler is an Editor, a Linker (to connect modules), and a monitor.

Making the transition to this assembler involves two major adjustments: you are now working with a 6809 and you are using a complicated assembler. If you are accustomed to working with simple assemblers (Supermon, Extramon, Micromon, or others), you will be baffled at first by the requirements of this assembler. Before looking into the significant differences between 6809 and our familiar 6502, let's see what is required if you decide you want to place the letter "a" in the upper left corner of your screen.

Simple 6502 Version:

```
0360 LDA #$41
0362 STA $8000
0365 BRK
```

Waterloo 6809 Assembler Version:

```
lda #'a
sta $8000
swi
end
```

SWI means software interrupt and resembles BRK on the 6502. (There are three software interrupts available: SWI, SWI2, and SWI3.) The apostrophe allows you to enter the actual letter which will be translated into the correct value for you. Otherwise, it's fairly simple at this point. You are in the Editor here (no need for addresses yet — they will be created later). The creation of your final, "object" code takes several steps: you must save this "file" to disk by typing p (for PUT) *name.asm*. Then, when the ASM file is on disk, you type BYE to get into the menu and select *a* (assemble) and you are asked for the filename, so you type: *name*. (It adds the ".asm" for you.)

The assembler makes two new files on the disk: *name.4 st* and *name.b09*. The first is a fairly straightforward listing of the source code with line numbers, object codes, mnemonics, and any comments separated into appropriate fields on screen. *Name.b09* is a file containing the object code to be used later by the Linker.

Your next step is to return to the Editor and make a fourth file:

```
"name"
org $1000
"name.b09"
```

and PUT it to disk under the title "name.cmd." The first line here names the "load module," the second line defines the starting address of the object code, and the third line names the object code file to be used in the linking process.

Then you type BYE again, select Linker from the menu, and type: *name*. (The linker will add ".cmd" to the name.) The linker creates two more files (for a total of six): *name.mod* (executable load module) and *name.map* (tells how *name.b09* was mapped into *name.mod*).

Now you are ready to run your program. You enter the monitor by typing "M" from the menu and then type: l name.mod (to load the "module"). You can then type : g 1000 and, voila!, an "a" appears on your screen.

The Monitor And Linker

Like TIM (the resident monitor on PET/CBM computers) the SuperPET monitor has several commands which are useful for debugging (Bank, Clear, Dump, Fill, Go, Modify, Passthrough, Quit, Registers, Stop, and Translate). "Bank" allows you to access any of the 16 banks of upper RAM for reading or writing. "Stop" sets breakpoints and "Clear" clears them. "Dump" is equivalent to "M" on TIM. "Modify" permits the same changes as "Dump," but in the form M ff 12 33 (where the byte at $00ff now becomes $12, $0100 becomes $33). "Quit" is like TIM's "x." "Passthrough" sends all input to a host computer and permits all output from the host to appear on screen.

"Translate" is a disassembly. Curiously, there is no provision for single-stepping or for SAVEing from the monitor. A single-step program exists (it was used at Waterloo to create the SuperPET languages), but it was not included in the monitor. As for SAVE, it was planned, evidently, that modules should be only created from the upper levels of the development system, following the steps

outlined above which result in six files per module.

The linker knits the relocatable object modules (*name.b09*) into longer executable load modules. The linker is invoked by creating the *name.cmd* file mentioned above and including various commands in this file. "Org" specifies the desired starting address for the code. "Banksize" defaults to $1000 if not specified and "Bankorg" defaults to $9000. Programs or modules may be loaded into specified banks with the "Bank" command. To merge external routines from the system library (or from your personal library of modules), use the "Include" command. Finally, "Export" sets aside some memory (Export bytespace = $7b00) which is named "bytespace" and reserved for tables, etc. Following its definition, "bytespace" can be referenced by any routine using the statement: xref bytespace.

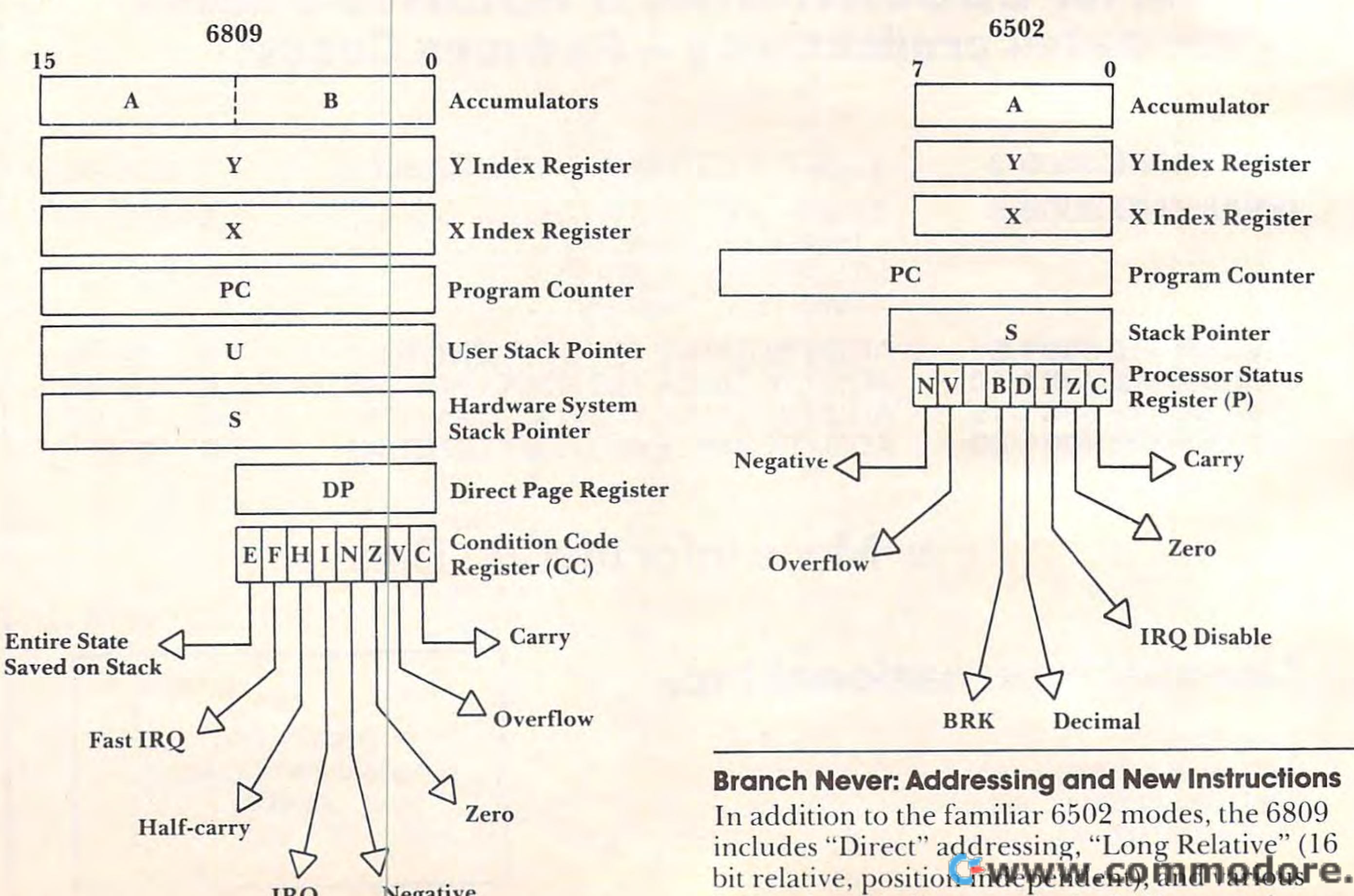
The 6809

As Figure 1 illustrates, the most obvious novelties in the 6809 are the addition of Accumulator B, the second (User) Stack, a Direct Page register, and half-carry, fast IRQ, and Entire State Saved condition flags. In addition, of course, the Y, X, and Accumulator registers and the stack pointers are expanded to 16 bits. Some of these improvements facilitate simplified addressing since a 16 bit register can address an entire 64K. Likewise, a stack can now be located anywhere in memory and be of any size desired. The A and B Accumulators can be concatenated to form Accumulator D (A is the MSB). This allows 16 bit addition, subtraction, compare, and so forth, via a single mnemonic.

The S stack pointer is used for JSRs and interrupts as expected, but the U stack pointer is controlled completely by the user and is unaffected by hardware status. This permits variables to be passed between routines.

The direct page register (normally 0) is used to form the MSB of an effective address during "direct addressing." The offset is the byte following the direct addressing mode opcode. This is like the familiar zero page addressing, but with the added ability to set "zero" at any page. A half-carry is a carry from bit three during eight-bit addition. There is a fast interrupt request line which can be masked with the fast IRQ flag. The entire-state-saved flag signals that all registers (not simply the program counter and CC) have been saved on the stack.

Figure 1.



Branch Never: Addressing and New Instructions

In addition to the familiar 6502 modes, the 6809 includes "Direct" addressing, "Long Relative" (16 bit relative, position independent), and various indexed and indirect modes including auto-increment and decrement by one or two bytes at a time. The efficiency inherent in 16 bit manipulations, new addressing modes, and new instructions permits greater programming freedom than is possible on the 6502. For example, the 6502 has approximately 56 mnemonics where the 6809 has nearly twice as many. (Mnemonic counts will vary depending on whether such instructions as ROL and ROL Accumulator are counted as distinct instructions.)

Among the more interesting new instructions is SWI (the entire machine state is saved and control is transferred through the vector at $FFFA-B. SWI2 is the same except that the IRQ masks flags are not set and the vector is $FFFA4-5). SEX means sign extended. BRA is branch always. Perhaps the most enigmatic new instruction is BRN, Branch Never. Though hundreds of uses for this spring to mind immediately, the assembler manual suggests that it can be used if you should become tired of NOP.

MUL multiplies accumulators A and B (unsigned) and stores the result in the D (A + B) accumulator. COMA and COMB complement these accumulators. ORB P inclusive ORs the value addressed by P, with B.

Assembler Expressions

The assembler provides for extensive programming options through lables, external references, libraries, macros, operators, conditional assembly, etc. QUIF? It's Quit IF, one of the structured programming statements. HI is a condition which follows QUIF and is true if the carry and zero flags are both clear. Other statements are: IF, ENDIF, ELSE, GUESS, ADMIT, ENDGUESS, LOOP, ENDLOOP, and UNTIL. Like their counterparts in other languages, these statements can be used in the assembler, if that is your preference.

Also, a library of common routines is included and can be called into a program by typing the reference name followed by an "underbar" character, an underline which is created by hitting the back-arrow key. ISDELIM checks to see if the character in question is a delimiter (not alphabetic or numeric). STOI converts a decimal string to an integer. ISUPPER sees if you have an uppercase alphabetic character. FSEEK finds a record in a random file. In all, there are 67 library modules. The first parameter is passed on D, the rest on the stack. Results come back in D.

The "structured programming" statements, 100 mnemonics, 67 library names, 17 addressing modes, 96K, two stacks, 16 memory banks. It's a bit of a transition. Nevertheless, 16 bit addressing, the freedom to MUL at will, and numerous other advantages all combine to make the 6809 option on the SuperPET exciting and promising for machine language programming. ©

PET Repairs For The Amateur

Louis F Sander
Pittsburgh, PA

My small keyboard PET has had several awful-looking symptoms over the past year, and each time I dreaded the size of the possible repair bill. But each time I was able to cure the problem myself, with no need for knowledge of digital electronics. Based on first-hand experience, and on many notes compared with others, here is what to look for when your PET is acting strange: loose connections, period.

Loose connections are probably the most frequent source of trouble in PET-like electronic equipment, and they are often the easiest to find and fix. You'll learn how I found mine, after a few words on safety. First, never look for trouble with your PET plugged in. Under normal circumstances, all lethal voltages are kept away from PET's main circuit board and other exposed parts, but when trouble comes, circumstances aren't normal. So always pull the plug when you're troubleshooting. Also, always take pains to avoid static electricity when you're poking around inside your PET. Tiny sparks that you can't see or feel can ruin some of the IC's in there, so don't take any chances. The best precaution is to ground yourself by touching bare metal on the cabinet whenever you touch an IC or the circuit board; it may look silly, but it's safe. Now for my war stories:

My first trouble was erratic operation. From time to time, I'd get a screen full of garbage, and my cassette motor would run and run. It looked like my reset button was locked down, but I knew it wasn't. On the advice of somebody who knew, I looked for an IC that was loose in its socket. When I found it, the trouble went away. With time and the flexing caused by neat, IC's all tend to walk out of their sockets. If you have symptoms of trouble, check this first. Open your PET and, with one hand touching the cabinet, firmly press down on both ends of every socketed IC, and walk them back into place. You'll be surprised how many are loose. Don't worry too much about flexing the printed circuit board itself — it can withstand a bit of bending.

My second problem came from a bad power connector. I'd lose everything on my screen, right in the middle of something important. At other times, I'd power up and not be able to get anything on the screen at all. When I found a hot power connector, I knew the cause was found. The power connector attaches your main circuit board to the wires coming from the large transformer and electrolytic capacitor at the left rear of PET's base. If you are having problems, especially ones that crop up after some length of 'on' time, run your machine for an hour or so, then feel the power connector. If it's noticeably hot, it is a candidate for replacement. I replaced both ends of my connector with Radio Shack 274-226 and 274-236, for under $3.00 total. If you're not an experienced electronics person, turn this job over to an expert — it's easy, but the new connectors are far from exact replacements.

My biggest and most mysterious problem was caused by a dirty contact on the connector between the main board and tape drive #1. For several months, I'd get strange screen messages and frequent system crashes whenever I tried to load a program that was other than the first one on a cassette. I'd say LOAD "RINKYDINK," the tape would start to move, and then I'd get some horribly misspelled version of ?ILLEGAL QUANTITY ERROR, sometimes before and sometimes after the PET had FOUND the programs preceding RINKYDINK. It got so bad that I gave up on ever being able to put more than one program on a tape. I could tell that the problem was associated with the unrecorded gaps between programs, but that's as far as it went.

I found the problem one day as I connected an audio amp to the tape READ line. The recorder was running a totally blank tape, and the noise on the READ line was tremendous. I accidentally jiggled the wire going from the recorder to the main board, and the noise stopped completely. Later I found that a poor ground contact on the PC board connector was allowing motor noise to get into the signal circuits, and that PET was trying to read the noise as data. No wonder it got an ILEGAL QUANIY ERRR! Two minutes with superfine sandpaper cured the problem, and now I can read through a whole C-60 with no system lockups. Keep your connectors clean.

By the way, I've had one minor problem unrelated to bad connections: My PET likes to read tapes a lot better without any amplifier connected to CB2. I don't know just why, but the machine definitely works better with nothing connected back there. So now I disconnect the amplifier whenever I'm through with a program that uses sound. I guess this really *is* another loose connection problem, but one of a different sort — in this one, loosest is best. But take it from one who knows more about it than he wants to — loose connections are common in your PET, and you can usually fix them yourself. ©

QUALITY ACCOUNTING SOFTWARE for the CBM COMPUTER

THE GENERAL LEDGER SYSTEM $150.00

All entries are made via formatted, fill in the blanks, screens. There is a separate check stub format disbursements entry screen and eight digit account numbers to allow sub coding as required. Up to fifty user designated journals are available. All data is verified on input with balance enforced. All journals are available for print at any point in the accounting cycle. Any printout may be printed by department. The general ledger prints: balance forward, full detail of each transaction, total credits, total debits, and end balance for each account. Available reports include: journals, disbursements register, current trial balance, audit trial balance, budget trial balance, income statement, balance sheet, cash flow analysis, and comparison of budget vs. actual amounts for year to date, or the current period.

FUND ACCOUNTING SYSTEM $200.00

The perfect accounting system for the municipal utility district, and the small city or school district. The system includes all features of the general ledger system with the added ability of printing all reports, and the general ledger, by fund as well as by department.

CLIENT ACCOUNTING SYSTEM $200.00

The accounting tool kit for the public accountant. This system includes all features of the general ledger system with the addition of a payroll check stub formatted screen for payroll check input. Also included are: a payroll disbursement register, a 1099 register, 941 reports, and W-2's.

INTERLOCKING MODULES

The following modules are available for any of the general ledger based systems:

Accounts Payable	$ 75.00
Accounts Receivable	$ 75.00
Payroll	$ 75.00
Job Cost Payroll	$100.00
Utility Billing System	$100.00

Accounting software may be reviewed at your dealer, or via mail. Full catalog, demo disc, sample operations manual, (please specify which system), and a $20. credit coupon — only $20.00

DEALERS! Please Write for Dealers Pack

Expanded Software Catalog Free on Request

BASIC SOFTWARE SERVICE
P.O. BOX 181, LA PORTE, TX., 77571
Phone Area Code 713/ 470-1857

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

Realtime Clock On Your Pet Screen

Mark L. Robinson

Editor's Note: In the version of Mr. Robinson's clock routine for 4.0 BASIC, the code has been moved up 38 (decimal) from the Upgrade version of Program 1. Add a value of 38 to his POKEs and references for the 4.0 version. — RTM

How many times have you sat down at your computer to fiddle around for a few minutes, returning to the real world hours (or days) later. This is not always a problem and I don't mind being splattered with cold suppers, missing parties, or aggravating my wife. But one night, I had some free moments to ponder the problem of losing track of time. Wouldn't it be great, I thought, if I could always have the correct time on the screen.

I knew that my PET had a 1/60 second counter which is updated during the internal interrupt cycle and some routines to print out the time. I started to study the memory map in Osborne's PET/CBM Personal Computer Guide and found the following items: jiffy memory, clock correction routine and the location of the interrupt addresses.

I figured if I could revector the interrupt through a small machine language program, I could capitalize on all three items - the jiffy clock to keep track of time, the interrupt addresses to return to the correct location and the clock correction routine to make up for the lost time of my program, if necessary.

Some other investigation showed that the routines that print the TIME$ use a lot of processing time and interact with memory locations that Basic uses. I figured it would be best to handle it completely as a separate little program. Then, the more I thought about it, I realized that once the time was set, I could follow it with a simple series of little counters rather than keep having to do long divisions. This also has the advantage of being able to jump back to the normal program whenever there is no carry up to the next most significant clock digit. This saves over 50 percent of the time penalty of the screen clock.

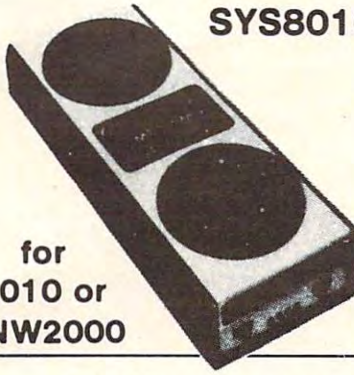
To initialize the clock and load the machine language program, I wrote a small BASIC program. You can follow the explanations in the flowchart and the symbolic listing of the machine language program along with the listing of the BASIC program. The machine language program is short enough to load with pokes rather than entering it using the machine language monitor. You can enter and run this as a normal basic program and, while the clock is running, you can use most BASIC programs. There is a small time penalty to use this while running BASIC, but if you are programming or game playing, it is not critical.

Incidentally, since the program is synchronized with the jiffy counter, you are automatically using the PET's internal correction routine. On a three hour run against a stop watch, the PET gained two seconds (so much for my stop watch). Two words of caution when you are writing programs: first, if you hit return on the line the time is on it will be entered in the listing and, second, if you have to load a program from the cassette, turn the clock off (POKE 144,46:POKE 145,230), load the program, and start the clock again (POKE 144,74:POKE 145,3). To reset the clock poke the correct time digits to locations 833-838.

Symbolic Listing
Real Time on Screen

```
DEFINITIONS
                     LOTB = Least significant time bit —
                        (Jiffy Counter)
                     TL0C1 Temporary holding location
                     TL0C2      '       '       '
                        of prior jiffy count
                     BASE 1-7 Base of count, 10 or 6
                     IMAGE 1-7 Location of time in memory
                     SCT 1-7 Screen locations of time
INITIALIZE
LDA LOTB             Initialize prior count set it ahead
ADC #05              to next .1 second. Note 1
STA TL0C2
LDA #Start           Revector interrupt to start
STA IRQ Low
START
LDA #LOTB            Check jiffies see if we've reached
STA TL0C1            next .1 sec
CMP TL0C2
ADC #05              Yes-set TL0C2 for next .1 sec, make sure
SBC TL0C2            that if more than 6 jiffs
ADC TL0C1            occurred we do not add too much
STA TL0C2
INC IMAGE,7          Increase .1 sec memory location by 1
LDX #07              Initialize counter routine
COUNTER
LDA IMAGE,X          Check to see if we've reached limit
CMP BASE,X           of base which produces a carry
BNE UPDATE INIT      No — then go to Update Init
LDA #00              Yes — place 0 in digit position
STA IMAGE,X
DEX                  Go to next number in sequence
BEQ UPDATE INIT      If we have done all 7 digits go to screen
                     update
INC IMAGE,X          Increment next digit by 1 (result of carry)
JMP COUNTER          Go back and check this digit for carry
```

SYS8010



for
8010 or
TNW2000

ROM based terminal package for CBM 8000 with 8010 or TNW 2000

SYS8010 ROM $49.95

OR

FREE with purchase of 8010 modern!

8010 modern........... $279.00

PRIVATE I

Eliminates data errors when using a modern with an extension line

$11.95

Software
by
MICROFEX
fits in 3.5K

VIC-MAN

eats dots and monsters in maze **$14.95**

VIC-20

VIC-ROIDS

shoot at asteroids as they float in space... **$14.95**

Pet Bulletin Board System

Start a bulletin board with your 16K Pet or CBM. Use 4040, 8050 or 2031 drives. See it in operation at (414) 282-8118.

See Message Number 3.

PBBS software
$49.95

We also carry and demo the entire Commodore and Jim-Pack lines of equipment.

Send check or money order with order. Allow $5.00 for shipping and handling.
PETTED microsystem

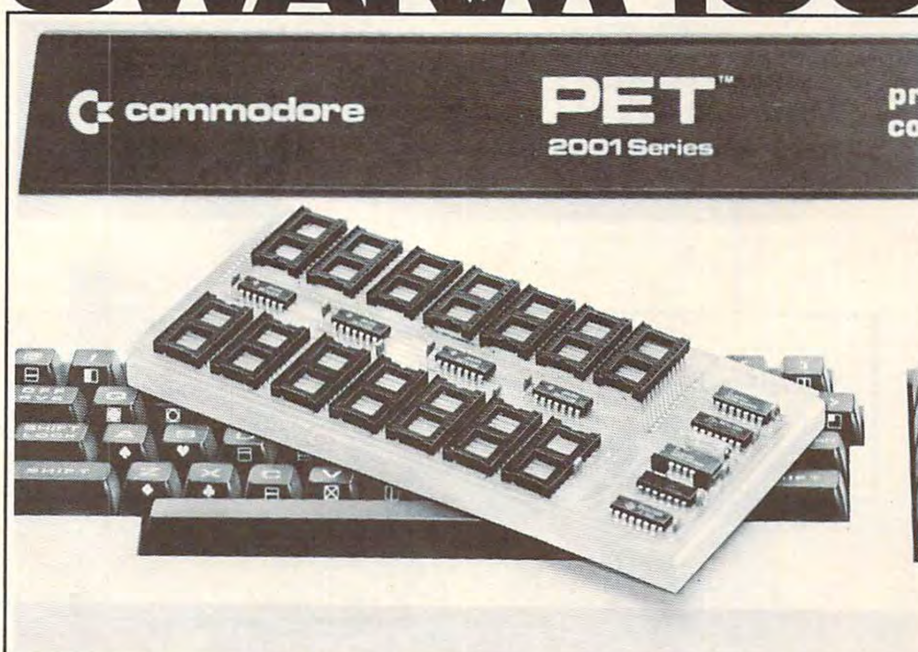
4265 W. Loomis
Milwaukee, WI 53221
under the "Shell" sign
at I-894 & Loomis Rd.
(414) 282-4181

PETTED

VISA & MASTERCHARGE ACCEPTED

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.

$125⁰⁰ (U.S.)

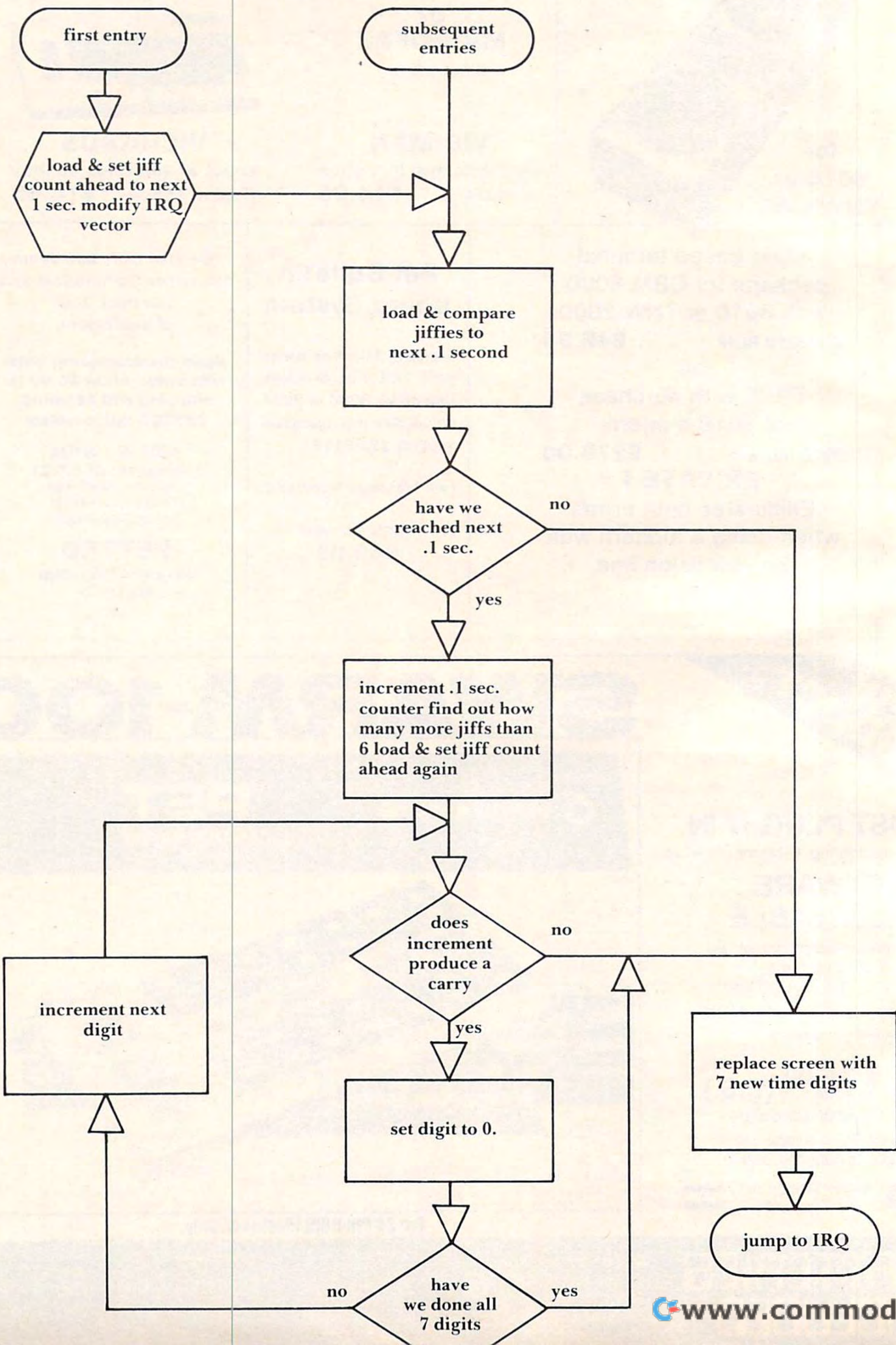
($150.00 Canadian)

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


first entry
subsequent entries
load & set jiff count ahead to next 1 sec. modify IRQ vector
load & compare jiffies to next .1 second
have we reached next .1 sec.
no
yes
increment .1 sec. counter find out how many more jiffs than 6 load & set jiff count ahead again
does increment produce a carry
no
yes
increment next digit
set digit to 0.
replace screen with 7 new time digits
jump to IRQ
have we done all 7 digits
no
yes

UPDATE INIT	
LDX #07	**Initialize the screen update routine**
UPDATE	
LDA IMAGE,X	**Load time digit**
ADC #$30	**Convert to PET number code**
STA SCT,X	**Store on screen**
DEX	
BNE UPDATE	**Have we done 7 digits? — no go back to update**
LDA #3A	**Yes — load and store colon on screen**
STA SCT,0	
JMP IRQ	**Return to PET IRQ routine**

Note 1: The reason that five is added to the jiffy count and not six (to get the next 6/60 or .1 sec) is that we are incrementing when the prior count location is less than the jiffy count. If the increment occurred on equality then you would add six. The reason for this is that I do not know if the jiffy count can count two sometimes in which case the equality would not occur for up to 12.8 seconds — when the same binary digit again occurred. This is also the reason the program checks for more than six counts.

Program 1: Upgrade Version

```
5 REM REAL TIME ON PET SCREEN
6 REM C M. ROBINSON 1980
7 REM OK FOR PERSONAL USE
10 FORA = 1 TO 100
20 READ B
30 POKE 825+A,B
40 NEXT
100 PRINT"{CLEAR}          HHMMSS"
110 INPUT"TIME";A$
120 TI$=A$
130 FORA=1 TO 6
140 D=VAL(MID$(A$,A,1))
150 POKE832+A,D
160 NEXT
200 POKE144,74:POKE145,3
250 NEW
1000 DATA10,10,6,10,6,10
1001 DATA10,0,0,0,0,0
1002 DATA0,0,87,90,165,143
1003 DATA105,5,141,73,3,169
1004 DATA85,133,144,165,143,141
1005 DATA72,3,205,73,3,48
1006 DATA38,105,5,237,73,3
1007 DATA109,72,3,141,73,3
1008 DATA238,71,3,162,7,189
1009 DATA64,3,221,57,3,48
1010 DATA14,169,0,157,64,3
1011 DATA202,240,6,254,64,3
1012 DATA76,111,3,162,7,189
1013 DATA64,3,105,48,157,31
1014 DATA128,202,208,245,169,58
1015 DATA141,31,128,76,46,230
1016 DATA0,0,0,0,0,0
```

Program 2: 4.0 Version

```
10 FORA=1TO100
20 READB
30 POKE863+A,B
40 NEXT
100 PRINT"{CLEAR}          HHMMSS"
110 INPUT"TIME";A$
120 TI$=A$
130 FORA=1TO6
140 D=VAL(MID$(A$,A,1))
150 POKE870+A,D
160 NEXT
170 POKE144,112:POKE145,3
180 NEW
864 DATA 10, 10, 6, 10, 6, 10
870 DATA 10, 0, 0, 0, 0, 0
876 DATA 0, 0, 87, 90, 165, 143
882 DATA 105, 5, 141, 111, 3, 169
888 DATA 123, 133, 144, 165, 143, 1
    41
894 DATA 110, 3, 205, 111, 3, 48
900 DATA 38, 105, 5, 237, 111, 3
906 DATA 109, 110, 3, 141, 111, 3
912 DATA 238, 109, 3, 162, 7, 189
918 DATA 102, 3, 221, 95, 3, 48
924 DATA 14, 169, 0, 157, 102, 3
930 DATA 202, 240, 6, 254, 102, 3
936 DATA 76, 149, 3, 162, 7, 189
942 DATA 102, 3, 105, 48, 157, 31
948 DATA 128, 202, 208, 245, 169, 5
    8
954 DATA 141, 31, 128, 76, 85, 228
960 DATA 0, 0, 0, 0, 0, 0
```

©

Statement of Ownership, Management And Circulation
As required by 39 U.S.C. 3685

1. COMPUTE! The Journal For Progressive Computing
1a. 537250
2. 12-1-81
3. Monthly 3A. 12 3B. $20.00
4. 625 Fulton Street, P.O. Box 5406, Greensboro, NC 27403
5. Same
6. Robert C. Lock, Publisher and Editor, 625 Fulton Street, Greensboro, NC 27403
Kathleen Martinek, Managing Editor, 625 Fulton Street, Greensboro, NC 27403
7. Small System Services, Inc.; Robert C. Lock, William E. Knox, Joretta F. Klepfer, Marlene R. Pratto; P.O. Box 5406, Greensboro, NC 27403
8. None
9. NA

10. Extent and nature of circulation	Average no. copies each issue during preceding 12 months	Actual no. copies of single issue published nearest to filing date
A. Total No. Copies (Net Press Run)	31,961	40,040
B. Paid Circulation		
1. Sales through dealers and carriers, street vendors and counter sales	15,232	19,491
2. Mail Subscription	10,516	17,773
C. Total Paid Circulation	25,748	37,264
D. Free distribution by mail, carrier or other means: samples, complimentary and other free copies	1,128	350
E. Total Distribution	26,876	37,614
F. Copies Not Distributed	3,597	2,426
1. Office use, left over, unaccounted, spoiled after printing		
2. Return from news agents	1,488	None to date
G. Total	[illegible]	[illegible]

I certify that the statements made by me above are correct and complete. Robert C. Lock, President, Small System Services, Inc.

Tape Load Test And Head Alignment

Louis F. Sander
Pittsburgh, PA

This article shows how to prepare and use a special test tape for the cassette recorder of any PET or CBM. When the tape is LOADed, its contents appear on the screen, allowing the user to see any tape errors *as they occur*. The tape error display is a sensitive indicator of the overall quality of the tape reading process, and one which can be used in curing such mysterious and aggravating problems as defective tapes and dirty or magnetized heads. The test tape can also be used as a working standard for head alignment.

Making The Load Test Tape

The first step in creating your tape is to enter and SAVE the "Test Tape Maker" program that appears later in the article. Then RUN it and follow the instructions on the screen, but be sure you understand the material in this section first.

The instructions ask you to use your Machine Language Monitor. Don't worry if you've never used it before — it's easy. If you have an older PET with Original ROMs, LOAD your monitor from tape and RUN it, being careful not to lose the "Test Tape Maker" instructions from the screen. With any other ROMs, you have a built-in monitor. Activate it by entering SYS 1024.

Once the monitor is running, it will prompt you with a dot. Mount a fully rewound tape, and save the 1st pass program by entering the indicated line exactly as it appears in the "Test Tape Maker" instructions. Then rewind the tape again, and prepare to do something unusual — you are going to record a new header on top of the one already on the tape, but you're going to leave the rest of the tape unchanged! You will do it by initiating another machine language SAVE, this time hitting STOP as soon as the header has been recorded on the tape. Knowing when to hit STOP is the tricky part, but the following paragraphs will teach you the trick.

If you can hear your tapes as they save, your task is easy. Some CB2 amplifiers amplify tape sounds, too, and you're in luck if yours works this way. If it doesn't, just connect your amplifier temporarily to pin eight of the user port connector, which is a convenient pickup point for the Tape Write signal. When you initiate your save, you'll hear about ten seconds of leader tone, followed by three seconds of buzz, followed by two more seconds of leader and a lot more buzz. The three seconds of buzz is the tape header, so you'll want to hit STOP the instant you start hearing the *second* section of leader tone.

Even if you have no way of listening to your SAVEs, you can tell when to hit STOP in making this tape. First, SAVE any program into a fully rewound tape. Then fully rewind it again and LOAD it, using a stopwatch to time the interval between pressing PLAY and seeing the FOUND message on the screen. Then, when recording LOAD TEST, wait exactly this length of time between pressing PLAY & RECORD and hitting STOP. On my PET, this is just over 13 seconds, and it should be the same on yours, but you should use a stopwatch to be sure.

Now that you know when to hit STOP, let's go back to "Test Tape Maker." Use the Monitor to save LOAD TEST onto the rewound 1st pass tape, making the exact entries appearing on your screen. Press PLAY, and as soon as the header has been recorded (the right number of seconds, or the appearance of the second leader tone), hit STOP. The STOP key on the computer is preferable to the one on the recorder, but either one will work. The timing of this move is critical to a fraction of a second, so use your fastest finger.

As soon as you hit STOP, your tape is finished. To be sure you have a good one, rewind it and LOAD it. If all is well, you will see the FOUND LOAD TEST and LOADING messages; then your screen will begin to fill with solid green (or white) squares. Once the screen is full, these will be replaced one-by-one with a full screen of colons, then a screen of shaded squares, then one of minus signs. Finally, an OK will print at the bottom of your screen, and after about 30 seconds, a READY message will appear somewhere on screen. No other characters should appear at any time. The newer machines with dynamic RAMs will not show the last two screens, and 80 column machines will combine the first two on one screen. If you cannot get the perfect "LOAD" described above, either you have made a defective tape, or you have a problem with your recorder. Clean and demagnetize your heads[1], and try a few more loads. If you still don't achieve perfection, try making a new LOAD TEST tape — you may have hit STOP too soon or too late, or you may be working with a defective cassette.

When you have a tape that loads perfectly at least once, load it several more times in succession.

PETTERM

INTELLIGENT TERMINAL

Turn your PET into an intelligent terminal with one of our terminal packages. These are complete assembled hardware and software packages. All include line editing/resend, repeat key, shift lock, output to CBM printer, and more . . . Delivered on PET cassette with manuals.

PETTERM I	All features above $75.00
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

for PET/CBM

FORTH is a new concept in programming, with the speed of compilers and interactive ease of BASIC. Programs become a part of FORTH extending the power of FORTH and your PET.

8050,4040 disk, cassette all PET-CBMs 16k+

Starter	fig-FORTH w editor assembler	$35
Personal	floating point; strings; source	$75
Professional	turnkey development/data base	$259

F S S

software for small computers

1903 Rio Grande
Austin, Texas
78705

1-512-477-2207

P.O. Box 8403
Austin, Texas
78712

master charge

DEALERS INQUIRE

VISA

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

You should get perfect or near-perfect results every time. Anything other than smooth screen filling, with no unusual characters, is an indication of an imperfect load. If you fail to achieve perfection, refer to the material in the next section. Otherwise, consider your test tape ready for use. Mark it with the date it was made, and set it aside in a safe place. If you want a second copy, use "Test Tape Maker" to create one, since there is no way to copy a completed tape. It's a good idea to put a copy of "Test Tape Maker" immediately after LOAD TEST on your tape, so you will have both of them whenever you need them.

Using The Test Tape

Now, whenever you have trouble LOADing a tape, you can evaluate the situation by loading LOAD TEST. If the screen fills properly, you know that your PET worked perfectly during the LOAD. The trouble is probably with your tape — it may be defective, or it may have been made on a recorder whose head is not aligned with yours. Read the Head Alignment section below.

If your screen *doesn't* fill properly, there may be a problem with your machine, and you can use the screen display to evaluate it. Every improper or misplaced character on the screen represents a mishandled byte. By using the second program copy recorded on every tape, PET can automatically correct up to 31 of these. LOAD TEST, by the way, lets you see this as it happens, when "proper" characters appear on the screen in place of the "bad" ones during the 30 seconds just before the READY message. Normally, you should have very few, if any, mishandled bytes. The more you have, the greater your problem. If you have more than a very few, even though PET can correct them, something is awry with your machine's LOAD process, and corrective action is called for.

The first corrective action, of course, is to clean and demagnetize your tape heads[1]. The second is to clean the contacts on the connector and the circuit board where your recorder plugs into your computer. If these steps fail to improve your situation, try a head alignment. If that also fails, see your serviceman.

Head Alignment

For a tape to load properly, your PETs read/write head must be precisely aligned with the magnetic field on the tape. The tape's field is, of course, perfectly aligned with the head of the recorder that made it. A small amount of misalignment between tape and read head often shows up as mishandled bytes, a moderate amount as a ?LOAD ERROR, and a large amount as a complete failure to read the tape.

Misalignment can occur with one of your own tapes if your machine's alignment has changed since you made the tape. It also occurs if a tape you are trying to read was recorded on a machine whose head is out of line with yours. Imperfect alignment between two PETs is quite common, and is often the cause of inability to load other people's tapes.

You can use your LOAD TEST tape to bring *any* recorder's head into alignment with the head that made LOAD TEST. Adjustment procedures have been published elsewhere[2]. Once you know how to make the adjustment, just load your test tape into the appropriate machine and adjust its head for perfect screen patterns. There is no need for any PEEKs to confirm the success of the LOAD, since you can see every mishandled byte right on the screen itself. You can even use LOAD TEST to adjust the head *while the tape is loading*, since it gives you 20-40 seconds of real-time feedback on the quality of your LOAD.

Always remember that you are adjusting the read head to the tape that it is reading. If the recorder which made it was misaligned from "standard," your test tape will be misaligned as well. Nevertheless, you should be able to get *any* recorder to read it. Now that you know how to make and use a "Load Test" tape, *you* need read no further. If you're interested in how and why it works, read on.

Theory Of Operation: Screen Images

Let us consider what is recorded on the Load Test tape. By a series of POKEs, "Test Tape Maker" created a machine language "program" of 1024 "square," 1024 colons, 1024 shaded squared, 997 minus signs, a space, an 'O' and a 'K', all in memory locations 2768 to 6839, (0AD0 – 1AB8 hex). When you saved that material as 1st pass, you made a tape whose header instructed PET to load it into those locations[3]. When you rewound the tape and did the second "computus interruptus" SAVE, you recorded a new header over the old one, but left the remaining material intact. The new header asks PET to load that material into memory locations 32768 – 36839, (8000 – 8FE8 hex), which are very interesting locations.

Experienced PET owners know that "screen memory" occupies the 1000 locations between 32768 and 33767. POKEs to those locations, (such as POKE 33000,42), cause characters to appear instantaneously on the screen. "Load Test" uses a less-well-known fact about screen memory: that POKEs to the screen memory locations *plus* 1024, (and on some machines 2048 or 3072), will *also* put characters on the screen. Clear your screen and POKE (33000 + 1024), 42 to see it for yourself. This multiple POKE ability exists because of a peculiarity in PET's address decoding scheme;

there really *isn't* any memory up there. These second, third, and fourth addresses for each screen position are sometimes called "images" of screen memory.

A little reflection on the above paragraphs will reveal that locations 32768 through 36839 include the screen memory plus its images, and that LOADing a program there will actually put the program material onto the screen up to four times in succession. There we can see the LOAD, and any errors, with our own two eagle eyes.

References

1. "Getting the Most From Your PET Cassette Deck," **COMPUTE!**, #10, March, 1981, page 42.

2. "Detecting Loading Problems and Correcting Alignment on Your PET," **COMPUTE!** #8, January 1981, page 114.

3. "All About LOADing PET Cassettes," **COMPUTE!** #16, September, 1981, page 129.

```
100 PRINT"{02 DOWN}TEST TAPE MAKER ~
    - WORKING - (25 SECONDS)"
110 FORI=2768TO3791:POKEI,160:NEXT
120 FORI=3792TO4815:POKEI,58:NEXT
130 FORI=4816TO5839:POKEI,102:NEXT
140 FORI=5840TO6836:POKEI,45:NEXT
150 POKEI,32:POKEI+1,15:POKEI+2,11
160 PRINT"{CLEAR}ACTIVATE THE ML MO
    NITOR, THEN MOUNT"
170 PRINT"A FULLY REWOUND TAPE AND ~
    ENTER:"
180 IFPEEK(50003)=0THEN310
190 PRINT". S"CHR$(34)"1ST PASS"CHR
    $(34)",01,0AD0,1AB8"
200 PRINT"THEN REWIND AND ENTER:"
210 PRINT". S"CHR$(34)"LOAD TEST"CH
    R$(34)",01,8000,8FE8"
220 PRINT"{REV}HIT STOP AS SOON AS ~
    THE HEADER HAS BEEN"
230 PRINT"{REV}RECORDED. (SEE ARTIC
    LE FOR DETAILS).{UP}":END
300 REM ** INSTR FOR ORIGINAL ROMS
310 PRINT".    S 01,1ST PASS,0AD0,1A
    B8"
320 PRINT"THEN REWIND AND ENTER:"
330 PRINT".S 01,LOAD TEST,8000,8FE8
    "
340 GOTO220
```

MICROMON

An Enhanced Machine Language Monitor

R. Arthur Cochrane
Beech Island, SC

Editor's note: Micromon is for Upgrade and 4.0 BASICs, all memory sizes, all keyboards and is in the public domain. We present it here because many readers live where there are no computer clubs to permit the exchange of public domain programs. If you have enough memory, you can add the additional commands of "Micromon Plus" as well. "Plus" is from $5B00 to $5F48 and you will want to move Micromon from $1000 up to $6000.

There is quite a bit of typing here so we've provided two checksum programs which will find and flag any errors. If you are unfamiliar with machine language programming, see the instructions for typing in "Supermon" in last month's **COMPUTE!**, *page 134. — RTM*

Background

For those who may not know what Micromon is, I will start with a little background. Micromon started as Extramon which is an extended machine language monitor for the TIM monitor in the PET. Extramon was originally written by Bill Seiler. It is for Upgrade BASIC and has the following commands;

A — A simple one line assembler.

B — Set a break point.

C — Compare two ranges of memory and print the addresses of any differences.

D — Disassemble a range of memory.

F — Fill a range of memory with a byte.

H — Hunt a range of memory for a certain HEX or ASCII pattern and print the addresses where they occur.

I — Do a memory dump or a range of memory by printing the HEX and ASCII values.

N — New Locate a machine language program by adding an offset to the three byte instructions.

Q — Start execution of a machine language program and stop execution when the break point is reached.

T — Transfer a range of memory to another part of memory.

W — Single step execution of a machine language program.

Extramon loads into the address range $1000 to $17FF, but the T and N commands can be used to relocate Extramon to another part of memory.

Micromon is an improved version of Extramon and is also by Bill Seiler. Micromon has the same commands as Extramon plus those of the TIM monitor and works on Upgrade BASIC and BASIC 4.0. It works on both BASICs because only 4 ROM routines are used, two of these routines are in the jump table at the top of memory and the other two used by Micromon are found by checking a location to determine the BASIC. The ability to use the up and down cursor control keys to scroll the memory dump and disassembler is added.

Improvements

Now Micromon has been improved by the addition of more instructions to make it a full 4K program. The following instructions have been added:

E — Kill Micromon by restoring the TIM break vector and IRQ vector and return to BASIC.

K — Kill Micromon by restoring the TIM break vector and IRQ vector and do a BRK to the TIM monitor.

O — Calculate a branch instruction offset given a starting and target address.

Z — Change to the opposite character set from the one currently in use.

$ — Print the decimal value, the ASCII values for the two bytes, and the binary value for an input HEX value.

— Print the HEX value, the ASCII values for the two bytes, and the binary value for an input decimal value.

% — Print the HEX value, the decimal value, and the ASCII values for the two bytes for an input binary value.

" — Print the HEX value, the decimal value, and the binary value for an input ASCII value.

+ — Add two HEX numbers.

– — Subtract two HEX numbers.

& — Print the checksum for a range of memory.

An additional module (Micromon Plus) to work with Micromon is also available. This module is about an additional 1K of program and it has the following commands:

I — Set form feeds and a heading for disassemblies and memory dump printouts.

P — Switch output to a printer for hard copy disassemblies and memory dumps.

SM-KIT

for Commodore Computers

A Programming Productivity Tool

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.

ONLY $40

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

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.

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 $129.95 (+ $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

(Dealer Inquiries Invited)

J — Print the address at which a file loads.

Y — Load a file starting at a specific address and not the load address in the file.

› or @ — The DOS commands for reading the disk error channel, sending commands to the disk, or displaying the disk directory.

Micromon is very useful for debugging machine language programs. The disassembler allows the code to be examined and the single step command allows following the execution of code to spot bugs. The Transfer and New Locate commands allow code to be relocated to another part of memory without the need for reassembly. Micromon is a must for any PET machine language programmer.

There are several extended monitors available for the PET. Supermon is one example. Most of the other monitors have some of the same commands as Micromon and maybe a few others. One of the problems with these monitors is that there are different versions for Upgrade BASIC and BASIC 4.0. Micromon will work, as is, on either BASIC. It does not work on Original BASIC though it might be possible to modify it. There is a version of Supermon for each of the three BASICs if a super monitor is needed for Original BASIC.

Because the VIC-20 has Upgrade Basic it will be possible to modify Micromon for VIC use, giving it a powerful machine language monitor. The modification will involve checking the subroutine calls and modifying the scroll for the screen size of the VIC. If anyone is successful in this modification they should be sure to publish the results for others. Because the full Micromon is a 4K program, it would be a good program for programming into a VIC plug-in program cartridge.

Micromon is *free* (so is Supermon), but where do you get it? A PET user group is one source. For those who would like source code, Micromon source code in Carl Moser's MAE assembler format is available. Micromon can be assembled and burned into an EPROM and plugged into an empty socket in the PET so Micromon is available with a SYS and does not have to be loaded each time the PET is reset or powered up.

I hope that you will pass Micromon on to your friends. This program is in the public domain and should be passed around freely. If anyone finds bugs or has comments please contact me about them.

I would like to thank James Strasma for all the information which he provided me for this work on Micromon.

Note To Other 6502 Users

Because Micromon uses only four ROM routines (input a character, output a character, load a program, and save a program) and a few zero page locations (IRQ vector, BRK vector, and screen line pointers) it may be possible for Apple, Atari, or other 6502 users to modify Micromon for their machine. If someone is successful at this be sure to pass the information on to others.

Micromon Instructions

SIMPLE ASSEMBLER

```
.A  2000 A9 12    LDA #$12
.A  2002 9D 00 80 STA $8000,X
.A  2005 DEX:GARBAGE
```

In the above example, the user started assembly at 2000 HEX. The first instruction was load a register with immediate 12 HEX. In the second line the user did not need to type the A and address. The simple assembler retypes the last entered line and prompts with the next address. To exit the assembler, type a return after the address prompt. Syntax is the same as the Disassembler output. A colon (:) can be used to terminate a line.

BREAK SET

```
.B 1000 00FF
```

The example sets a break at 1000 HEX on the FF HEX occurrence of the instruction at 1000. Break set is used with the QUICK TRACE command. A BREAK SET with count blank stops at the first occurrence of the break address.

COMPARE MEMORY

```
.C 1000 2000 C000
```

Compares memory from HEX 1000 to HEX 2000 to memory beginning at HEX C000. Compare will print the locations of the unequal bytes.

DISASSEMBLER

```
.D 2000 3000
., 2000 A9 12    LDA #$12
., 2002 9D 00 80 STA $8000,X
., 2005 AA       TAX
```

Disassembles from 2000 to 3000. The three bytes following the address may be modified. Use the CRSR KEYS to move to and modify the bytes. Hit return and the bytes in memory will be changed. MICROMON will then disassemble that line again.

Disassembly can be done under the control of the cursor. To disassemble one at a time from $1000.

```
.D 1000
```

If the cursor is on the last line, one instruction can be disassembled for each pressing of the cursor down key. If it is held down, the key will repeat and continuous disassembly will occur. Disassembly can even be in reverse! If the screen is full of a disassembly listing, place the cursor at the top line of the screen and press the cursor up key.

EXIT MICROMON

.E

Combine the killing of MICROMON and exit to BASIC.

FILL MEMORY

.F 1000 1100 FF

Fills the memory from 1000 HEX to 1100 HEX with the byte FF HEX.

GO RUN

.G

Go to the address in the PC Register display and begin run code. All the registers will be replaced with the displayed values.

.G 1000

Go to address 1000 HEX and begin running code.

HUNT MEMORY

.H C000 D000 'READ

Hunt thru memory from C000 HEX to D000 HEX for the ASCII string "read" and print the address where it is found. Maximum of 32 characters may be used.

.H C000 D000 20 D2 FF

Hunt memory from C000 HEX to D000 HEX for the sequence of bytes 20 D2 FF and print the address. A maximum of 32 bytes may be used. Hunt can be stopped with the STOP key.

KILL MICROMON

.K

Restore the Break vector and IRQ that was saved before MICROMON was called and break into the TIM monitor. A return to MICROMON can be done with a Go to the value in the PC register.

LOAD

.L "RAM TEST",08

Load the program named RAM TEST from the disk. *Note for cassette users:* To load or save to cassette. Kill MICROMON with the K command to return to the TIM monitor. Then use the TIM monitor L and S commands to load and save to the cassettes. This has to be done because of the repeat keys of MICROMON. BASIC 4.0 users then can return to MICROMON with a Go command to the PC value but BASIC 2.0 users should return to BASIC then SYS to Micromon because the TIM overwrites the IRQ value for loads and saves with a filename.

MEMORY DISPLAY

.M 0000 0008

```
.: 0000 30 31 32 33 34 35 36 37 1234567
.: 0008 38 41 42 43 44 45 46 47 89ABCDE
```

Display memory from 0000 HEX to 0008 in HEX and ASCII. The bytes following the address may be modified by editing and then typing a RETURN.

Memory display can also be done with the cursor control keys.

NEW LOCATER

```
.N 1000 17FF 6000 1000 1FFF
.N 1FB0 1FFF 6000 1000 1FFF W
```

The first line fixes all three byte instructions in the range 1000 HEX to 1FFF HEX by adding 6000 HEX offset to the bytes following the instruction. New Locater will not adjust any instruction outside of the 1000 HEX to 1FFF HEX range. The second line adjusts Word values in the same range as the first line. New Locater stops and disassembles on any bad op code.

CALCULATE BRANCH OFFSET

.O 033A 033A FE

Calculate the offset for branch instructions. The first address is the starting address and the second address is the target address. The offset is then displayed.

QUICK TRACE

```
.Q
.Q 1000
```

The first example begins trace at the address in the PC of the register display. The second begins at 1000 HEX. Each instruction is executed as in the WALK command, but no disassembly is shown. The Break Address is checked for the break on Nth occurrence. The execution may be stopped by pressing the STOP and = (left arrow on business) keys at the same time.

REGISTER DISPLAY

```
.R
   PC IRQ SR AC XR YR SP
.: 0000 E455 01 02 03 04 05
```

Displays the register values saved when MICROMON was entered. The values may be changed with the edit followed by a RETURN.

SAVE

.S "1:PROGRAM NAME",08,0800,0C80

Save to disk drive #1 memory from 0800 HEX up to, *but not including*, 0C80 HEX and name it PROGRAM NAME. See note in LOAD command for cassette users.

TRANSFER MEMORY

.T 1000 1100 5000

Transfer memory in the range 1000 HEX to 1100 HEX and start storing it at address 5000 HEX.

WALK CODE

.W

Single step starting at address in register PC.

.W 1000

Single step starting at address 1000 HEX. Walk will cause a single step to execute and will disassemble the next instruction. Stop key stops walking. The J key finishes a subroutine that is walking then continues with the walk.

EXIT TO BASIC

.X

Return to BASIC READY mode. The stack value saved when entered will be restored. Care should be taken that this value is the same as when the MONITOR was entered. A CLR in BASIC will fix any stack problems. Do not X to BASIC then return to MICROMON via a SYS to the cold start address. Return via a SYS to a BRK (SYS 1024) or SYS to the Warm start of MICROMON (Warm start = Cold start + 3) An X and cold start will write over the TIM break vector that was saved.

CHANGE CHARACTER SETS

.Z

Change from uppercase/graphics to lower/ uppercase mode or vice versa.

HEX CONVERSION

.$4142 16706 A B 0100 0001 0100 0010

A HEX number is input and the decimal value, the ASCII for the two bytes, and the binary values are returned. The ASCII control values are returned in reverse.

HEX conversion can also be scrolled with the cursor control keys.

DECIMAL CONVERSION

.#16706 7142 A B 0100 0001 0100 0010

A decimal number is input and the HEX value, the ASCII for the two bytes, and the binary values are returned.

BINARY CONVERSION

.%0100000101000010 4142 16706 A B

A binary number is input and the HEX value, the decimal number, and the ASCII values are returned.

ASCII CONVERSION

."A 41 65 0100 0001

An ASCII character is input and the HEX value, decimal value, and binary values are returned. Because of the quote, *the control characters can be determined also.*

ADDITION

.+ 1111 2222 3333

The two HEX numbers input are added, and the sum displayed.

SUBTRACTION

.-3333 1111 2222

The second number is subtracted from the first number and the difference displayed.

CHECKSUM

.& A000 AFFF 67E2

The checksum between the two addresses is calculated and displayed.

MICROMON INSTRUCTIONS:

A SIMPLE ASSEMBLE
B BREAK SET
C COMPARE MEMORY
D DISASSEMBLER
E EXIT MICROMON
F FILL MEMORY
G GO RUN
H HUNT MEMORY
K KILL MICROMON
L LOAD
M MEMORY DISPLAY
N NEW LOCATER
O CALCULATE BRANCH
Q QUICK TRACE
R REGISTER DISPLAY
S SAVE
T TRANSFER MEMORY
W WALK CODE
X EXIT TO BASIC
Z CHANGE CHARACTER SETS
$ HEX CONVERSION
 # DECIMAL CONVERSION
% BINARY CONVERSION
" ASCII CONVERSION
+ ADDITION
- SUBTRACTION
& CHECKSUM

MICROMON also has repeat for all keys.

MICROMON is executed by the following: SYS 4096 as listed in Program 2 where it resides in $1000 to $1FFF.

For 8032, make the following changes for MICROMON operation. In location the X stands for the start of MICROMON. Values in HEX.

Location	Old Value	New Value	
X3E7	08	10	To display 16 instead
X3EC	08	10	of 8 bytes.
X3F6	08	10	
X427	08	10	
XDA3	08	10	
XCFC	28	50	To fix scroll.
XD7B	28	50	
XE16	83	87	
XE20	28	50	
XE24	C0	80	

XE26	04	08	
XE37	27	4F	
XE46	28	50	
X681	24	00	To print all characters in Walk command.

Micromon Plus Instructions

PRINTING DISASSEMBLER

.(Shift) D 1000 1FFF

The same as the Disassembler but no **.,** printed before each line. Also the ASCII values for the bytes are output at the end of the line.

FORM FEED SET

.I

Sets a form feed for printout. Gives 57 printed lines per page. Works with the Shift D and Shift M commands.

.I "Heading"

Sets form feed with a message to be printed at the top of each page.

.I X

Cancels form feed.

PRINT LOAD ADDRESS

.J "File name"

Read the load address of the file and print it in hex. Device number 8 is used.

KILL MICROMON ADDITIONS

.(Shift) K

Kill MICROMON and its additions and BRK to the TIM monitor. This is the same as the unshifted K command except now a G command will reinitialize MICROMON and the additions.

LOAD FROM DISK

.(Shift) L "filename"

This is the same as the normal load command except that the disk (device #8) is used as the default, not the cassette.

PRINTING MEMORY DUMP

.(Shift) M F000 F100

The same as the normal Memory dump, but does not print the **.:** and prints out 16 hex bytes and the ASCII for them.

PRINT SWITCHER

.P

If the output is to the CRT then switch the output to the printer (device #4). If the output is not the CRT then clear the output device and restore the output to the CRT.

.P 06

Make device #6 the output device if the current output is the CRT.

SEND TO PROM PROGRAMMER

.U 06 7000 7FFF

This command will send out bytes to a PROM programmer on the IEEE bus. The first byte is the device number and the two addresses are the range of memory to output. A CHR$(2) is sent first to start the programmer. This is followed by the memory bytes as ASCII characters separated by spaces. After all bytes have been sent, a CHR$(3) is sent to stop the programmer. MICROMON then does a checksum on the range to compare against the programmer checksum. Although this is for a particular programmer, it could be modified for others.

SPECIFY LOAD ADDRESS

.Y 7000 "Filename"

This command allows a file to be loaded starting at the address you specify and not the load address it would normally load into. The disk (device #8) is used for loading.

TEXT FLIP FOR 8032 & FAT 40's

.(Shift) Z

This is for 8032 and Fat 40's to go from Text to Graphics mode or vice versa.

DOS SUPPORT

.@ or .>

This reads the error channel from disk device number 8.

.@ disk command or .> disk command

This sends the disk command to disk device number 8.

.@$0 or .>$0

This reads the directory from disk device number 8. The SPACE BAR will hold the display and any other key will start it again and the STOP key will return to command mode.

CONTROL CHARACTERS

.(Up arrow)G

This command will print the control character of the ASCII character input.

Examples of controls:

g	Ring bell
i	Tab set and clear
M	Insert line
n	Text mode
N	Graphics mode
q	Cursor down
Q	Cursor up
s	Home cursor
S	Clear screen
u	Delete line
v	Erase end
V	Erase begin

MICROMON ADDITIONAL INSTRUCTIONS

(Shift) D PRINTING DISASSEMBLER
I HEADING AND FORM FEED CONTROL
J PRINT LOAD ADDRESS
(Shift) K KILL MICROMON ADDITIONS
(Shift) L LOAD FROM DISK
(Shift) M PRINT MEMORY DISPLAY
P PRINTER SWITCHING
U SEND TO PROM PROGRAMMER
Y SPECIFY LOAD ADDRESS
(Shift) Z TEXT/GRAPHICS FLIP
› DOS SUPPORT COMMANDS
@ DOS SUPPORT COMMANDS
(Up arrow) CONTROL CHARACTERS

Program 1.

```
10 DATA 15463,14894,14290,11897,12
    453,13919,14116,11715,1257
    5,14571
20 DATA 13693,11853,12903,14513,12
    137,15006,12654,13291,1243
    6,13899
30 DATA 15366,9999,11834,13512,128
    92,14475,15149,14896,15782
    ,9511
40 DATA 12171,8985
100 Q=4096
110 FOR BLOCK=1TO32
120 FOR BYTE=ØTO127
130 X=PEEK(Q+BYTE):CK=CK+X
140 NEXT BYTE
150 READ SUM
160 IF SUM <> CK THEN PRINT" ERROR ~
    IN BLOCK #"BLOCK:GOTO170
165 PRINT"                    BLOCK"
    BLOCK" IS CORRECT
170 CK=Ø:Q=Q+128
180 NEXT BLOCK
```

Program 2.

```
1ØØØ 4C ØC 1Ø 4C 6F 1Ø 4C CF
1ØØ8 FF 4C D2 FF 78 A5 92 A6
1Ø1Ø 93 8D E5 Ø2 8E E6 Ø2 AD
1Ø18 F6 1F AE F7 1F 8D E3 Ø2
1Ø2Ø 8E E4 Ø2 AD FØ 1F AE F1
1Ø28 1F 85 92 86 93 A5 9Ø A6
1Ø3Ø 91 CD EE 1F DØ Ø5 EC EF
1Ø38 1F FØ 1Ø 8D 9E Ø2 8E 9F
1Ø4Ø Ø2 AD EE 1F AE EF 1F 85
1Ø48 9Ø 86 91 AD EC 1F AE ED
1Ø5Ø 1F EØ 8Ø BØ Ø8 85 34 86
1Ø58 35 85 3Ø 86 31 A9 1Ø 8D
1Ø6Ø 84 Ø2 8D 85 Ø2 A9 ØØ 8D
1Ø68 86 Ø2 8D A2 Ø2 58 ØØ 38
1Ø7Ø AD 7B Ø2 E9 Ø1 8D 7B Ø2
1Ø78 AD 7A Ø2 E9 ØØ 8D 7A Ø2

1Ø8Ø 2Ø 55 19 A2 42 A9 2A 2Ø
1Ø88 29 18 A9 52 DØ 23 A9 3F
1Ø9Ø 2Ø Ø9 1Ø 2Ø 55 19 A9 2E
1Ø98 2Ø Ø9 1Ø A9 ØØ 8D 94 Ø2
1ØAØ 8D A2 Ø2 A2 FF 9A 2Ø A4
1ØA8 18 C9 2E FØ F9 C9 2Ø FØ
1ØBØ F5 A2 1D DD 92 1F DØ 13
1ØB8 8D 87 Ø2 8A ØA AA BD BØ
1ØCØ 1F 85 FB BD B1 1F 85 FC
1ØC8 6C FB ØØ CA 1Ø E5 6C E3
1ØDØ Ø2 A2 Ø2 DØ Ø2 A2 ØØ B4
1ØD8 FB DØ Ø9 B4 FC DØ Ø3 EE
1ØEØ 94 Ø2 D6 FC D6 FB 6Ø A9
1ØE8 ØØ 8D 8C Ø2 2Ø 4F 12 A2
1ØFØ Ø9 2Ø 52 19 CA DØ FA 6Ø
1ØF8 A2 Ø2 B5 FA 48 BD 91 Ø2

11ØØ 95 FA 68 9D 91 Ø2 CA DØ
11Ø8 F1 6Ø AD 92 Ø2 AC 93 Ø2
111Ø 4C 17 11 A5 FD A4 FE 38
1118 E5 FB 8D 91 Ø2 98 E5 FC
112Ø A8 ØD 91 Ø2 6Ø A9 ØØ FØ
1128 Ø2 A9 Ø1 8D 95 Ø2 2Ø E6
113Ø 17 2Ø 55 19 2Ø 13 11 2Ø
1138 3C 18 9Ø 1B 2Ø ØA 11 BØ
114Ø Ø3 4C C5 11 2Ø 7F 11 E6
1148 FD DØ Ø2 E6 FE 2Ø 3B 19
115Ø AC 94 Ø2 DØ 45 FØ E5 2Ø
1158 ØA 11 18 AD 91 Ø2 65 FD
116Ø 85 FD 98 65 FE 85 FE 2Ø
1168 F8 1Ø 2Ø 7F 11 2Ø ØA 11
117Ø BØ 53 2Ø D1 1Ø 2Ø D5 1Ø
1178 AC 94 Ø2 DØ 1D FØ EB A2

118Ø ØØ A1 FB AC 95 Ø2 FØ Ø2
1188 81 FD C1 FD FØ ØB 2Ø 13
119Ø 18 2Ø 52 19 2Ø AE 18 FØ
1198 Ø1 6Ø 4C 93 1Ø 2Ø Ø1 18
11AØ 2Ø ØB 18 2Ø A4 18 2Ø 6F
11A8 18 9Ø 17 8D 89 Ø2 AE 94
11BØ Ø2 DØ 12 2Ø 13 11 9Ø ØD
11B8 AD 89 Ø2 81 FB 2Ø 3B 19
11CØ DØ EC 4C 8E 1Ø 4C 93 1Ø
11C8 2Ø Ø1 18 2Ø ØB 18 2Ø A4
11DØ 18 A2 ØØ 2Ø A4 18 C9 27
11D8 DØ 14 2Ø A4 18 9D A3 Ø2
11EØ E8 2Ø Ø6 1Ø C9 ØD FØ 22
11E8 EØ 2Ø DØ F1 FØ 1C 8E 97
11FØ Ø2 2Ø 77 18 9Ø CC 9D A3
11F8 Ø2 E8 2Ø Ø6 1Ø C9 ØD FØ
```

```
1200 Ø9 2Ø 6F 18 9Ø BC EØ 2Ø
1208 DØ EC 8E 88 Ø2 2Ø 55 19
1210 A2 ØØ AØ ØØ B1 FB DD A3
1218 Ø2 DØ ØA C8 E8 EC 88 Ø2
1220 DØ F2 2Ø 8E 11 2Ø 3B 19
1228 AC 94 Ø2 DØ Ø5 2Ø 13 11
1230 BØ DE 4C 93 1Ø 2Ø 39 14
1238 2Ø 13 11 9Ø ØD AØ 2C 2Ø
1240 E7 1Ø 2Ø AB 12 2Ø AE 18
1248 DØ EE 2Ø B3 15 DØ E3 2Ø
1250 47 19 2Ø 13 18 2Ø 52 19
1258 2Ø ØE 1E 48 2Ø ØB 13 68
1260 2Ø 22 13 A2 Ø6 EØ Ø3 DØ
1268 14 AC 8B Ø2 FØ ØF AD 96
1270 Ø2 C9 E8 B1 FB BØ 1D 2Ø
1278 A1 12 88 DØ F1 ØE 96 Ø2
```

```
1280 9Ø ØE BD E9 1E 2Ø AD 15
1288 BD EF 1E FØ Ø3 2Ø AD 15
1290 CA DØ D2 6Ø 2Ø B7 12 AA
1298 E8 DØ Ø1 C8 98 2Ø A1 12
12AØ 8A 8E 88 Ø2 2Ø 1A 18 AE
12A8 88 Ø2 6Ø AD 8B Ø2 2Ø B6
12BØ 12 85 FB 84 FC 6Ø 38 A4
12B8 FC AA 1Ø Ø1 88 65 FB 9Ø
12CØ Ø1 C8 6Ø A8 4A 9Ø ØB 4A
12C8 BØ 17 C9 22 FØ 13 29 Ø7
12DØ Ø9 8Ø 4A AA BD 98 1E BØ
12D8 Ø4 4A 4A 4A 4A 29 ØF DØ
12EØ Ø4 AØ 8Ø A9 ØØ AA BD DC
12E8 1E 8D 96 Ø2 29 Ø3 8D 8B
12FØ Ø2 98 29 8F AA 98 AØ Ø3
12F8 EØ 8A FØ ØB 4A 9Ø Ø8 4A
```

```
1300 4A Ø9 2Ø 88 DØ FA C8 88
1308 DØ F2 6Ø B1 FB 2Ø A1 12
1310 A2 Ø1 2Ø F1 1Ø CC 8B Ø2
1318 C8 9Ø FØ A2 Ø3 CØ Ø3 9Ø
1320 F1 6Ø A8 B9 F6 1E 8D 92
1328 Ø2 B9 36 1F 8D 93 Ø2 A9
1330 ØØ AØ Ø5 ØE 93 Ø2 2E 92
1338 Ø2 2A 88 DØ F6 69 3F 2Ø
1340 Ø9 1Ø CA DØ EA 4C 52 19
1348 2Ø Ø1 18 A9 Ø3 2Ø AC 13
1350 AØ 2C 4C 5Ø 15 BD Ø5 Ø1
1358 CD F8 1F DØ ØB BD Ø6 Ø1
1360 CD F9 1F DØ Ø3 2Ø D7 18
1368 A5 97 CD 83 Ø2 FØ ØA 8D
1370 83 Ø2 A9 1Ø 8D 84 Ø2 DØ
1378 24 C9 FF FØ 2Ø AD 84 Ø2
```

```
1380 FØ Ø5 CE 84 Ø2 DØ 16 CE
1388 85 Ø2 DØ 11 A9 Ø2 8D 85
1390 Ø2 A5 9E DØ Ø8 A9 ØØ 85
1398 97 A9 Ø2 85 A8 AD F3 1F
13AØ 48 AD F2 1F 48 Ø8 48 48
13A8 48 6C 9E Ø2 8D 89 Ø2 48
13BØ 2Ø A4 18 2Ø 19 19 DØ F8
13B8 68 49 FF 4C AE 12 2Ø 39
13CØ 14 AE 94 Ø2 DØ ØD 2Ø 13
13C8 11 9Ø Ø8 2Ø D6 13 2Ø AE
13DØ 18 DØ EE 4C 4A 12 2Ø 55
13D8 19 A2 2E A9 3A 2Ø 29 18
13EØ 2Ø 52 19 2Ø 13 18 A9 Ø8
13E8 2Ø Ø3 19 A9 Ø8 2Ø B9 13
13FØ A9 12 2Ø Ø9 1Ø AØ Ø8 A2
13F8 ØØ A1 FB 29 7F C9 2Ø BØ
```

```
1400 Ø2 A9 2E 2Ø Ø9 1Ø C9 22
1408 FØ Ø4 C9 62 DØ ØA A9 14
1410 2Ø Ø9 1Ø A9 22 2Ø Ø9 1Ø
1418 2Ø 3B 19 88 DØ DB A9 92
1420 4C Ø9 1Ø 2Ø Ø1 18 A9 Ø8
1428 2Ø AC 13 2Ø B3 15 2Ø D6
1430 13 A9 3A 8D 6F Ø2 4C 5C
1438 15 2Ø Ø1 18 85 FD 86 FE
1440 2Ø Ø6 1Ø C9 ØD FØ Ø3 2Ø
1448 Ø6 18 4C 55 19 2Ø 4C 18
1450 85 FD 86 FE A2 ØØ 8E A4
1458 Ø2 2Ø A4 18 C9 2Ø FØ F4
1460 9D 8D Ø2 E8 EØ Ø3 DØ F1
1468 CA 3Ø 14 BD 8D Ø2 38 E9
1470 3F AØ Ø5 4A 6E A4 Ø2 6E
1478 A3 Ø2 88 DØ F6 FØ E9 A2
```

```
1480 Ø2 2Ø Ø6 1Ø C9 ØD FØ 22
1488 C9 3A FØ 1E C9 2Ø FØ F1
1490 2Ø A4 15 BØ ØF 2Ø 84 18
1498 A4 FB 84 FC 85 FB A9 3Ø
14AØ 9D A3 Ø2 E8 9D A3 Ø2 E8
14A8 DØ D7 8E 92 Ø2 A2 ØØ 8E
14BØ 94 Ø2 A2 ØØ 8E 89 Ø2 AD
14B8 94 Ø2 2Ø C3 12 AE 96 Ø2
14CØ 8E 93 Ø2 AA BD 36 1F 2Ø
14C8 84 15 BD F6 1E 2Ø 84 15
14DØ A2 Ø6 EØ Ø3 DØ 14 AC 8B
14D8 Ø2 FØ ØF AD 96 Ø2 C9 E8
14EØ A9 3Ø BØ 1E 2Ø 81 15 88
14E8 DØ F1 ØE 96 Ø2 9Ø ØE BD
14FØ E9 1E 2Ø 84 15 BD EF 1E
14F8 FØ Ø3 2Ø 84 15 CA DØ D2
```

```
1500 FØ Ø6 2Ø 81 15 2Ø 81 15
1508 AD 92 Ø2 CD 89 Ø2 FØ Ø3
1510 4C 91 15 2Ø 3C 18 AC 8B
1518 Ø2 FØ 2E AD 93 Ø2 C9 9D
1520 DØ 1F 2Ø 13 11 9Ø ØA 98
1528 DØ 6F AE 91 Ø2 3Ø 6A 1Ø
```

```
1530 08 C8 D0 65 AE 91 02 10
1538 60 CA CA 8A AC 8B 02 D0
1540 03 B9 FC 00 91 FB 88 D0
1548 F8 AD 94 02 91 FB A0 41
1550 8C 6F 02 20 B3 15 20 E7
1558 10 20 AB 12 A9 20 8D 70
1560 02 8D 75 02 A5 FC 20 B8
1568 15 8E 71 02 8D 72 02 A5
1570 FB 20 B8 15 8E 73 02 8D
1578 74 02 A9 07 85 9E 4C 93

1580 10 20 84 15 8E 88 02 AE
1588 89 02 DD A3 02 F0 0D 68
1590 68 EE 94 02 F0 03 4C B2
1598 14 4C 8E 10 E8 8E 89 02
15A0 AE 88 02 60 C9 30 90 03
15A8 C9 47 60 38 60 CD 8C 02
15B0 D0 03 60 A9 91 4C 09 10
15B8 48 4A 4A 4A 4A 20 32 18
15C0 AA 68 29 0F 4C 32 18 8D
15C8 7D 02 08 68 29 EF 8D 7C
15D0 02 8E 7E 02 8C 7F 02 68
15D8 18 69 01 8D 7B 02 68 69
15E0 00 8D 7A 02 A9 80 8D 86
15E8 02 D0 21 AD 13 E8 10 03
15F0 4C 55 13 D8 68 8D 7F 02
15F8 68 8D 7E 02 68 8D 7D 02

1600 68 8D 7C 02 68 8D 7B 02
1608 68 8D 7A 02 A5 90 8D 82
1610 02 A5 91 8D 81 02 BA 8E
1618 80 02 20 D7 18 AD 12 E8
1620 58 AD 7C 02 29 10 F0 03
1628 4C 6F 10 2C 86 02 50 1F
1630 AD 7A 02 CD 99 02 D0 6D
1638 AD 7B 02 CD 98 02 D0 65
1640 AD 9C 02 D0 5D AD 9D 02
1648 D0 55 A9 80 8D 86 02 30
1650 14 4E 86 02 90 D2 AE 80
1658 02 9A AD F5 1F 48 AD F4
1660 1F 48 4C 1F 17 20 55 19
1668 20 30 19 8D 89 02 A0 00
1670 20 0B 19 AD 7B 02 AE 7A
1678 02 85 FB 86 FC 20 52 19

1680 A9 24 8D 8C 02 20 52 12
1688 20 E4 FF F0 FB C9 03 D0
1690 03 4C 93 10 C9 4A D0 56
1698 A9 01 8D 86 02 D0 4F CE
16A0 9D 02 CE 9C 02 AD 12 E8
16A8 C9 EE F0 04 C9 6F D0 3E
16B0 A2 53 4C 85 10 A9 00 F0
16B8 12 AD 9A 02 AE 9B 02 8D
16C0 9C 02 8E 9D 02 A9 40 D0
16C8 02 A9 80 8D 86 02 20 06
16D0 10 C9 0D F0 11 C9 20 D0
16D8 5C 20 60 18 20 FC 18 20
16E0 06 10 C9 0D D0 4F 20 55
16E8 19 AD 86 02 F0 22 78 A9
16F0 A0 8D 4E E8 CE 13 E8 2C
16F8 12 E8 AD F0 1F AE F1 1F

1700 8D 82 02 8E 81 02 A9 3B
1708 A2 00 8D 48 E8 8E 49 E8
1710 AE 80 02 9A 78 AD 81 02
1718 85 91 AD 82 02 85 90 AD
1720 7A 02 48 AD 7B 02 48 AD
1728 7C 02 48 AD 7D 02 AE 7E
1730 02 AC 7F 02 40 4C 8E 10
1738 20 4C 18 8D 98 02 8E 99
1740 02 A9 00 8D 9A 02 8D 9B
1748 02 20 5D 18 8D 9A 02 8E
1750 9B 02 4C 93 10 20 E6 17
1758 8D A0 02 8E A1 02 20 5D
1760 18 8D 8D 02 8E 8E 02 20
1768 5D 18 8D 8F 02 8E 90 02
1770 20 06 10 C9 0D F0 0A 20
1778 06 10 C9 57 D0 03 EE 8C

1780 02 20 3C 18 AE 94 02 D0
1788 18 20 0A 11 90 13 AC 8C
1790 02 D0 1A B1 FB 20 C3 12
1798 AA BD F6 1E D0 06 20 E7
17A0 10 4C 93 10 AC 8B 02 C0
17A8 02 D0 33 F0 03 8C 8B 02
17B0 88 38 B1 FB AA ED 8D 02
17B8 C8 B1 FB ED 8E 02 90 1E
17C0 88 AD 8F 02 F1 FB C8 AD
17C8 90 02 F1 FB 90 10 88 18
17D0 8A 6D A0 02 91 FB C8 B1
17D8 FB 6D A1 02 91 FB 20 3B
17E0 19 88 10 FA 30 9E 20 4C
17E8 18 85 FD 86 FE 20 5D 18
17F0 8D 92 02 8E 93 02 20 A4
17F8 18 20 60 18 85 FB 86 FC

1800 60 20 4C 18 B0 F6 20 60
1808 18 B0 03 20 5D 18 85 FD
1810 86 FE 60 A5 FC 20 1A 18
1818 A5 FB 48 4A 4A 4A 4A 20
1820 32 18 AA 68 29 0F 20 32
1828 18 48 8A 20 09 10 68 4C
1830 09 10 18 69 F6 90 02 69
1838 06 69 3A 60 A2 02 B5 FA
1840 48 B5 FC 95 FA 68 95 FC
1848 CA D0 F3 60 A9 00 8D 97
1850 02 20 A4 18 C9 20 F0 F9
1858 20 84 18 B0 08 20 A4 18
```

```
1860 20 6F 18 90 07 AA 20 6F
1868 18 90 01 60 4C 8E 10 A9
1870 00 8D 97 02 20 A4 18 C9
1878 20 D0 09 20 A4 18 C9 20

1880 D0 0F 18 60 20 99 18 0A
1888 0A 0A 0A 8D 97 02 20 A4
1890 18 20 99 18 0D 97 02 38
1898 60 C9 3A 08 29 0F 28 90
18A0 02 69 08 60 20 06 10 C9
18A8 0D D0 F8 4C 93 10 A5 9B
18B0 C9 EF D0 07 08 20 CC FF
18B8 85 9E 28 60 20 C6 18 AD
18C0 13 E8 6A 90 F7 60 20 AE
18C8 18 D0 0B 20 D7 18 A9 03
18D0 85 B0 A9 00 85 AF 60 08
18D8 78 AD 40 E8 09 10 8D 40
18E0 E8 A9 7F 8D 4E E8 A9 3C
18E8 8D 11 E8 A9 3D 8D 13 E8
18F0 AD EE 1F 85 90 AD EF 1F
18F8 85 91 28 60 8D 7B 02 8E

1900 7A 02 60 8D 89 02 A0 00
1908 20 52 19 B1 FB 20 1A 18
1910 20 3B 19 CE 89 02 D0 F0
1918 60 20 6F 18 90 0B A2 00
1920 81 FB C1 FB F0 03 4C 8E
1928 10 20 3B 19 CE 89 02 60
1930 A9 7C 85 FB A9 02 85 FC
1938 A9 05 60 E6 FB D0 07 E6
1940 FC D0 03 EE 94 02 60 98
1948 48 20 55 19 68 A2 2E 20
1950 29 18 A9 20 2C A9 0D 4C
1958 09 10 A2 00 BD 76 1F 20
1960 09 10 E8 E0 1C D0 F5 A0
1968 3B 20 47 19 AD 7A 02 20
1970 1A 18 AD 7B 02 20 1A 18
1978 20 52 19 AD 81 02 20 1A

1980 18 AD 82 02 20 1A 18 20
1988 30 19 20 03 19 4C 93 10
1990 4C 8E 10 20 4C 18 20 FC
1998 18 20 5D 18 8D 82 02 8E
19A0 81 02 20 30 19 8D 89 02
19A8 20 A4 18 20 19 19 D0 F8
19B0 F0 DB 20 60 1C AE 80 02
19B8 9A 6C 94 00 4C 8E 10 A0
19C0 01 84 D4 88 84 D1 84 96
19C8 84 9D A9 02 85 DB A9 A3
19D0 85 DA 20 06 10 C9 20 F0
19D8 F9 C9 0D F0 1A C9 22 D0
19E0 DB 20 06 10 C9 22 F0 36
19E8 C9 0D F0 0B 91 DA E6 D1
19F0 C8 C0 10 F0 C7 D0 EA AD
19F8 87 02 C9 4C D0 E1 AD 00

1A00 C0 C9 40 D0 06 20 22 F3
1A08 4C 12 1A C9 4C D0 AD 20
1A10 56 F3 20 BC 18 A5 96 29
1A18 10 D0 E1 4C 93 10 20 06
1A20 10 C9 0D F0 D2 C9 2C D0
1A28 F0 20 6F 18 29 0F F0 C3
1A30 C9 03 F0 FA 85 D4 20 06
1A38 10 C9 0D F0 BA C9 2C D0
1A40 E6 20 F9 17 20 06 10 C9
1A48 2C D0 F4 20 60 18 85 C9
1A50 86 CA 20 06 10 C9 20 F0
1A58 F9 C9 0D D0 EC AD 87 02
1A60 C9 53 D0 F7 AD 00 C0 C9
1A68 40 D0 06 20 A4 F6 4C 93
1A70 10 C9 4C D0 D4 20 E3 F6
1A78 4C 93 10 20 01 18 20 3B

1A80 19 20 3B 19 20 0B 18 20
1A88 52 19 20 13 11 90 0A 98
1A90 D0 15 AD 91 02 30 10 10
1A98 08 C8 D0 0B AD 91 02 10
1AA0 06 20 1A 18 4C 93 10 4C
1AA8 8E 10 20 01 18 20 C0 1A
1AB0 4C 93 10 20 55 19 A2 2E
1AB8 A9 24 20 29 18 20 13 18
1AC0 20 2F 1B 20 E6 1A 20 52
1AC8 19 20 CC 1A 20 CF 1A 20
1AD0 52 19 A2 04 A9 30 18 0E
1AD8 92 02 2E 93 02 69 00 20
1AE0 09 10 CA D0 EF 60 A5 FC
1AE8 A6 FB 8D 93 02 8E 92 02
1AF0 20 52 19 A5 FC 20 FA 1A
1AF8 A5 FB AA 20 52 19 8A 29

1B00 7F C9 20 08 B0 0A A9 12
1B08 20 09 10 8A 18 69 40 AA
1B10 8A 20 09 10 C9 22 F0 04
1B18 C9 62 D0 0A A9 14 20 09
1B20 10 A9 22 20 09 10 28 B0
1B28 05 A9 92 20 09 10 60 20
1B30 52 19 A6 FB A5 FC AC 00
1B38 C0 C0 40 D0 03 4C D9 DC
1B40 C0 4C D0 03 4C 83 CF 4C
1B48 8E 10 20 5B 1B B0 F8 20
1B50 52 19 20 13 18 20 C3 1A
1B58 4C 93 10 A2 04 A9 00 85
1B60 FC 20 17 1C 20 83 1B 85
1B68 FB 20 78 1B 20 92 1B CA
1B70 D0 F7 08 20 52 19 28 60
1B78 20 06 10 C9 0D F0 0F C9

1B80 20 F0 0B C9 30 90 C0 C9
```

```
1B88 3A B0 BC 29 0F 60 68 68
1B90 18 60 85 FE A5 FC 48 A5
1B98 FB 48 06 FB 26 FC 06 FB
1BA0 26 FC 68 65 FB 85 FB 68
1BA8 65 FC 85 FC 06 FB 26 FC
1BB0 A5 FE 65 FB 85 FB A9 00
1BB8 65 FC 85 FC 60 20 17 1C
1BC0 8D 93 02 48 48 20 52 19
1BC8 20 52 19 68 20 1A 18 20
1BD0 52 19 68 AA A9 00 20 36
1BD8 1B 20 52 19 20 CC 1A 4C
1BE0 93 10 20 F4 1B 20 52 19
1BE8 20 13 18 20 2F 1B 20 E6
1BF0 1A 4C 93 10 A2 0F A9 00
1BF8 85 FB 85 FC 20 17 1C 20
```

```
1C00 83 1B 20 11 1C 20 78 1B
1C08 20 11 1C CA D0 F7 4C 52
1C10 19 4A 26 FB 26 FC 60 20
1C18 A4 18 C9 20 F0 F9 60 A9
1C20 02 4D 4C E8 8D 4C E8 4C
1C28 93 10 20 0B 18 4C F6 17
1C30 20 2A 1C 18 A5 FB 65 FD
1C38 85 FB A5 FC 65 FE 85 FC
1C40 4C 50 1C 20 2A 1C 20 13
1C48 11 84 FC AD 91 02 85 FB
1C50 20 52 19 20 13 18 4C 93
1C58 10 20 60 1C 00 6C EC 1F
1C60 78 AD E5 02 AE E6 02 85
1C68 92 86 93 AD 9E 02 AE 9F
1C70 02 85 90 86 91 58 60 20
1C78 2A 1C 20 3C 18 20 52 19
```

```
1C80 A0 00 8C 92 02 8C 93 02
1C88 20 13 11 90 1D AD 94 02
1C90 D0 18 A0 00 18 B1 FB 6D
1C98 92 02 8D 92 02 98 6D 93
1CA0 02 8D 93 02 20 3B 19 4C
1CA8 88 1C AD 93 02 20 1A 18
1CB0 AD 92 02 20 1A 18 4C 93
1CB8 10 AD A2 02 D0 04 A5 9E
1CC0 D0 06 68 A8 68 AA 68 40
1CC8 AD 6F 02 C9 11 D0 7D A5
1CD0 D8 C9 18 D0 ED A5 C4 85
1CD8 FD A5 C5 85 FE A9 19 8D
1CE0 9C 02 A0 01 20 8C 1E C9
1CE8 3A F0 1A C9 2C F0 16 C9
1CF0 24 F0 12 CE 9C 02 F0 CA
1CF8 38 A5 FD E9 28 85 FD B0
```

```
1D00 E1 C6 FE D0 DD 8D 87 02
1D08 20 45 1E B0 B5 AD 87 02
1D10 C9 3A D0 11 18 A5 FB 69
1D18 08 85 FB 90 02 E6 FC 20
1D20 D6 13 4C 39 1D C9 24 F0
1D28 1A 20 0E 1E 20 AB 12 A9
1D30 00 8D 8C 02 A0 2C 20 4F
1D38 12 A9 00 85 9E 4C 4A 12
1D40 4C C2 1C 20 3B 19 20 B3
1D48 1A 4C 39 1D C9 91 D0 F0
1D50 A5 D8 D0 EC A5 C4 85 FD
1D58 A5 C5 85 FE A9 19 8D 9C
1D60 02 A0 01 20 8C 1E C9 3A
1D68 F0 1A C9 2C F0 16 C9 24
1D70 F0 12 CE 9C 02 F0 15 18
1D78 A5 FD 69 28 85 FD 90 E1
```


```
1D00 E1 C6 FE D0 DD 8D 87 02
1D08 20 45 1E B0 B5 AD 87 02
1D10 C9 3A D0 11 18 A5 FB 69
1D18 08 85 FB 90 02 E6 FC 20
1D20 D6 13 4C 39 1D C9 24 F0
1D28 1A 20 0E 1E 20 AB 12 A9
1D30 00 8D 8C 02 A0 2C 20 4F
1D38 12 A9 00 85 9E 4C 4A 12
1D40 4C C2 1C 20 3B 19 20 B3
1D48 1A 4C 39 1D C9 91 D0 F0
1D50 A5 D8 D0 EC A5 C4 85 FD
1D58 A5 C5 85 FE A9 19 8D 9C
1D60 02 A0 01 20 8C 1E C9 3A
1D68 F0 1A C9 2C F0 16 C9 24
1D70 F0 12 CE 9C 02 F0 15 18
1D78 A5 FD 69 28 85 FD 90 E1
```


```
1D80 E6 FE D0 DD 8D 87 02 20
1D88 45 1E 90 03 4C C2 1C AD
1D90 87 02 C9 3A F0 06 C9 24
1D98 F0 1D D0 27 20 15 1E 38
1DA0 A5 FB E9 08 85 FB B0 02
1DA8 C6 FC 20 D9 13 A9 00 85
1DB0 9E 20 40 1E 4C 96 10 20
1DB8 15 1E 20 D5 10 20 B6 1A
1DC0 4C AD 1D 20 15 1E A5 FB
1DC8 A6 FC 85 FD 86 FE A9 10
1DD0 8D 9C 02 38 A5 FD ED 9C
1DD8 02 85 FB A5 FE E9 00 85
1DE0 FC 20 0E 1E 20 AB 12 20
1DE8 13 11 F0 07 B0 F3 CE 9C
1DF0 02 D0 E0 EE 8B 02 AD 8B
1DF8 02 20 B9 13 A2 00 A1 FB
```

```
1E00 8E 8C 02 A9 2C 20 4D 19
1E08 20 52 12 4C AD 1D A2 00
1E10 A1 FB 4C C3 12 A9 83 85
1E18 C8 85 FE A9 00 85 C7 A9
1E20 28 85 FD A0 C0 A2 04 88
1E28 B1 C7 91 FD 98 D0 F8 C6
1E30 C8 C6 FE CA D0 F1 A2 27
```

```
1E38 A9 20 9D 00 80 CA 10 FA
1E40 A9 13 4C 09 10 C0 28 D0
1E48 02 38 60 20 8C 1E C9 20
1E50 F0 F3 88 20 75 1E AA 20
1E58 75 1E 85 FB 86 FC A9 FF
1E60 8D A2 02 85 A7 A5 AA F0
1E68 0A A5 A9 A4 C6 91 C4 A9
1E70 00 85 AA 18 60 20 8C 1E
1E78 20 99 18 0A 0A 0A 0A 8D

1E80 97 02 20 8C 1E 20 99 18
1E88 0D 97 02 60 B1 FD C8 29
1E90 7F C9 20 B0 02 09 40 60
1E98 40 02 45 03 D0 08 40 09
1EA0 30 22 45 33 D0 08 40 09
1EA8 40 02 45 33 D0 08 40 09
1EB0 40 02 45 B3 D0 08 40 09
1EB8 00 22 44 33 D0 8C 44 00
1EC0 11 22 44 33 D0 8C 44 9A
1EC8 10 22 44 33 D0 08 40 09
1ED0 10 22 44 33 D0 08 40 09
1ED8 62 13 78 A9 00 21 81 82
1EE0 00 00 59 4D 91 92 86 4A
1EE8 85 9D 2C 29 2C 23 28 24
1EF0 59 00 58 24 24 00 1C 8A
1EF8 1C 23 5D 8B 1B A1 9D 8A

1F00 1D 23 9D 8B 1D A1 00 29
1F08 19 AE 69 A8 19 23 24 53
1F10 1B 23 24 53 19 A1 00 1A
1F18 5B 5B A5 69 24 24 AE AE
1F20 A8 AD 29 00 7C 00 15 9C
1F28 6D 9C A5 69 29 53 84 13
1F30 34 11 A5 69 23 A0 D8 62
1F38 5A 48 26 62 94 88 54 44
1F40 C8 54 68 44 E8 94 00 B4
1F48 08 84 74 B4 28 6E 74 F4
1F50 CC 4A 72 F2 A4 8A 00 AA
1F58 A2 A2 74 74 74 72 44 68
1F60 B2 32 B2 00 22 00 1A 1A
1F68 26 26 72 72 88 C8 C4 CA
1F70 26 48 44 44 A2 C8 0D 20
1F78 20 20 20 50 43 20 20 49

1F80 52 51 20 20 53 52 20 41
1F88 43 20 58 52 20 59 52 20
1F90 53 50 41 42 43 44 46 47
1F98 48 4C 4D 4E 51 52 53 54
1FA0 57 58 2C 3A 3B 24 23 22
1FA8 2B 2D 4F 5A 4B 25 26 45
1FB0 4D 14 38 17 25 11 35 12
1FB8 9D 11 B5 16 C8 11 BF 19
1FC0 BE 13 55 17 B9 16 5A 19
1FC8 BF 19 29 11 C9 16 B5 19
1FD0 48 13 23 14 93 19 AA 1A
1FD8 4A 1B BD 1B 30 1C 43 1C
1FE0 7B 1A 1F 1C 59 1C E2 1B
1FE8 77 1C B2 19 00 10 55 13
1FF0 EB 15 B9 1C C6 15 8E 10
1FF8 BC 18 30 35 32 37 38 31
```

Program 3.

```
10 DATA 15965,14778,13059,14282,14
    416,17693,12979,12903,1767
    6,21760
20 DATA 14416,17693,12979,12903
100 Q=23296
110 FOR BLOCK=1TO8
120 FOR BYTE=0TO127
130 X=PEEK(Q+BYTE):CK=CK+X
140 NEXT BYTE
150 READ SUM
160 IF SUM <> CK THEN PRINT" ERROR
    IN BLOCK #"BLOCK:GOTO170
165 PRINT"                    BLOCK"
    BLOCK" IS CORRECT
170 CK=0:Q=Q+128
180 NEXT BLOCK
190 PRINT"ANY REMAINING PROBLEMS AR
    E EITHER WITHIN THE FINAL"
200 PRINT"SHORT BLOCK OR WITHIN DAT
    A STATEMENTS IN THIS PROGR
    AM."
```

Program 4.

```
5B00 78 A5 90 A6 91 CD EE 6F
5B08 D0 05 EC EF 6F F0 30 8D
5B10 9E 02 8E 9F 02 AD EE 6F
5B18 AE EF 6F 85 90 86 91 A5
5B20 92 A6 93 8D E5 02 8E E6
5B28 02 AD 3C 5F AE 3D 5F 8D
5B30 E3 02 8E E4 02 AD F0 6F
5B38 AE F1 6F 85 92 86 93 AD
5B40 3E 5F AE 3F 5F E0 80 B0
5B48 08 85 34 86 35 85 30 86
5B50 31 A9 10 8D 84 02 8D 85
5B58 02 A9 00 8D 86 02 8D A2
5B60 02 8D E7 02 8D E8 02 58
5B68 00 A2 0C DD 15 5F D0 13
5B70 8D 87 02 8A 0A AA BD 22
5B78 5F 85 FB BD 23 5F 85 FC

5B80 6C FB 00 CA 10 E5 4C 8E
5B88 60 20 39 64 20 13 61 90
5B90 17 20 EF 60 8E 8C 02 20
```

```
5B98 52 62 20 AB 5B 20 AB 62
5BA0 20 93 5C 20 AE 68 D0 E4
5BA8 4C 9B 60 A2 1E 20 F1 60
5BB0 A0 00 B1 FB 20 60 5C CC
5BB8 8B 02 C8 90 F5 60 A5 B0
5BC0 C9 03 D0 19 20 06 60 AA
5BC8 A9 04 E0 0D F0 09 20 6F
5BD0 68 29 1F C9 04 90 AF 20
5BD8 E3 5B 4C 9B 60 20 CC FF
5BE0 4C 93 60 85 B0 85 D4 20
5BE8 09 5C AE 00 C0 E0 40 D0
5BF0 0B 20 BA F0 20 2D F1 A5
5BF8 96 D0 E2 60 E0 4C D0 5D

5C00 20 D5 F0 20 48 F1 4C F7
5C08 5B A9 00 85 96 8D FC 03
5C10 85 0D 8D E8 02 60 20 39
5C18 64 AE 94 02 D0 10 20 13
5C20 61 90 0B 20 31 5C 20 93
5C28 5C 20 AE 68 D0 EB 4C A8
5C30 5B A2 05 20 F1 60 20 13
5C38 68 A2 02 20 F1 60 A9 10
5C40 20 03 69 A9 10 20 B9 63
5C48 A2 04 20 F1 60 A0 10 A2
5C50 00 A1 FB 20 60 5C 20 3B
5C58 69 88 D0 F5 60 4C 8E 60
5C60 29 7F C9 20 B0 02 A9 20
5C68 4C 09 60 20 06 60 C9 0D
5C70 F0 19 C9 20 D0 03 20 17
5C78 6C C9 58 F0 50 20 71 5D

5C80 8E E8 02 A2 02 20 A7 5C
5C88 4C 9B 60 A2 04 20 C1 5C
5C90 4C 9B 60 20 55 69 AE E7
5C98 02 F0 31 CE E7 02 D0 2C
5CA0 AE E8 02 F0 1A A2 06 20
5CA8 C1 5C A2 14 20 F1 60 BD
5CB0 A3 02 20 09 60 E8 EC E8
5CB8 02 D0 F4 A2 03 D0 02 A2
5CC0 09 20 55 69 CA D0 FA A9
5CC8 39 8D E7 02 60 A9 00 8D
5CD0 E7 02 8D E8 02 4C 9B 60
5CD8 20 09 5C 20 CC FF 20 06
5CE0 60 C9 0D F0 16 C9 24 F0
5CE8 24 48 20 9E 5D 68 20 09
5CF0 60 20 06 60 C9 0D D0 F6
5CF8 4C DD 5B 20 52 69 20 C5

5D00 5D 20 06 60 C9 0D F0 F0
5D08 20 09 60 D0 F4 A2 00 20
5D10 82 5D 20 8B 5D 20 55 69
5D18 20 55 69 A0 03 D0 02 A0
5D20 02 84 D1 A9 08 85 AF 20
5D28 06 60 AA A4 96 D0 36 20
5D30 06 60 A4 96 D0 2F C6 D1
5D38 D0 ED 20 36 6B 20 52 69
5D40 20 06 60 F0 05 20 09 60
5D48 D0 F6 20 55 69 A9 00 85
5D50 AF 20 E4 FF F0 C9 D0 05
5D58 20 E4 FF F0 FB C9 20 F0
5D60 F7 C9 03 D0 BA 20 12 5E
5D68 20 55 69 4C 93 60 20 17
5D70 6C C9 22 D0 7B A2 00 20
5D78 06 60 C9 0D F0 0C C9 22

5D80 F0 08 9D A3 02 E8 E0 40
5D88 90 ED 60 86 D1 A9 A3 85
5D90 DA A9 02 85 DB 20 CC FF
5D98 20 F3 5D 4C C9 5D A9 08
5DA0 85 D4 85 B0 AC 00 C0 C0
5DA8 40 D0 0B 20 BA F0 A9 6F
5DB0 20 28 F1 4C F7 5B C0 4C
5DB8 D0 36 20 D5 F0 A9 6F 20
5DC0 43 F1 4C F7 5B A9 6F 85
5DC8 D3 A9 08 85 D4 85 AF AC
5DD0 00 C0 C0 40 D0 0B 20 B6
5DD8 F0 A5 D3 20 64 F1 4C F7
5DE0 5B C0 4C D0 0B 20 D2 F0
5DE8 A5 D3 20 93 F1 4C F7 5B
5DF0 4C 8E 60 A9 08 85 D4 A9
5DF8 60 85 D3 AD 00 C0 C9 40

5E00 D0 06 20 66 F4 4C F7 5B
5E08 C9 4C D0 E4 20 A5 F4 4C
5E10 F7 5B A9 00 85 AF AD 00
5E18 C0 C9 40 D0 03 4C 8F F3
5E20 C9 4C D0 CC 4C CE F3 A9
5E28 02 2C 4C E8 08 A9 0E 28
5E30 F0 02 09 80 20 09 60 4C
5E38 93 60 20 09 5C 20 6E 5D
5E40 20 8B 5D 20 06 60 8D FB
5E48 00 20 06 60 8D FC 00 20
5E50 12 5E 20 52 69 A9 24 A2
5E58 20 20 29 68 20 13 68 4C
5E60 93 60 20 60 6C 00 6C 3E
5E68 5F A0 08 84 D4 A0 4C 8C
5E70 87 02 A0 00 4C C4 69 20
5E78 17 6C 29 9F 4C 34 5E 4C

5E80 8E 60 20 A4 68 20 6F 68
5E88 29 1F C9 04 90 F1 85 D4
5E90 20 2A 6C A5 FD A6 FE 8D
5E98 92 02 8E 93 02 20 3C 68
5EA0 A5 D4 20 E3 5B A9 02 20
5EA8 09 60 20 52 69 20 13 61
5EB0 90 0F AE 94 02 D0 0A A1
5EB8 FB 20 1A 68 20 3B 69 D0
5EC0 E9 A9 03 20 09 60 20 EF
5EC8 60 20 CC FF 20 F8 60 4C
```

```
5ED0 7D 6C 20 09 5C 20 01 68
5ED8 20 6E 5D 86 D1 20 04 5F
5EE0 20 8D 5D 20 06 60 20 06
5EE8 60 A9 00 85 AF AD 00 C0
5EF0 C9 40 D0 06 20 52 F3 4C
5EF8 01 5F C9 4C D0 81 20 8C

5F00 F3 4C 12 6A AD 00 C0 C9
5F08 40 D0 03 4C 0A F4 C9 4C
5F10 D0 EA 4C 49 F4 50 C4 49
5F18 CD 40 3E DA 4A CB CC 5E
5F20 55 59 BE 5B 89 5B 6B 5C
5F28 16 5C D8 5C D8 5C 27 5E
5F30 3A 5E 62 5E 69 5E 77 5E
5F38 82 5E D2 5E 69 5B 00 5B
5F40 31 30 32 31 38 31 AA AA
```

©

Self-Modifying Programs In BASIC

David Williams
Toronto, Canada

The notion of a program which alters itself as it runs raises feelings of doubt and mistrust in many novice computer users. It seems that such a program would be doomed to failure through some kind of logical paradox. In fact this is not the case. Providing that the part of the program which guides the modification process is separate from that which is being changed, and that no attempt is made to execute program lines which are in the process of being modified, no problems need arise.

As a demonstration, try keying in the following program. As you do so, be careful *not* to include any spaces in lines 10 or 20, or between the quote marks in line 120. Line 20 should consist of a string of exactly twenty π's.

```
10 GOTO1ØØ
2Ø ^^^^^^^^^^^^^^^^^^^^
3Ø RETURN
1ØØ FORI=826TO838:POKEI,32:NEXT
11Ø INPUTS$
12Ø S$="GOTO2ØØ:"+S$+CHR$(13)
13Ø FORI=1TOLEN(S$):POKE838+I,ASC(MID$(S
     ¬$,I)):NEXT
14Ø POKE175,2:POKE212,2:POKE594Ø8,
     ¬PEEK(594Ø8)ANDNOT32:POKE188,Ø:
     ¬POKE176,2
15Ø END
2ØØ POKE175,Ø:POKE176,3
21Ø I=Ø
22Ø PK=PEEK(517+I)
23Ø IFPK=ØTHEN3ØØ
24Ø POKE1Ø38+I,PK
25Ø I=I+1
26Ø GOTO22Ø
3ØØ FORI=ITO19:POKE1Ø38+I,32:NEXT
4ØØ GOSUB2Ø
READY.
```

When you have finished entering the program, SAVE it before you first run it. If you have made any typing mistakes it is possible that the program may destroy itself or crash the PET when it is run. Having a copy on tape could save you a lot of re-typing!

When the program is run, a question mark and flashing cursor should appear on the screen. This is the input line 110. Respond to this by typing in some simple instruction in BASIC, such as PRINT 2+3*5, and hit the return key. Within the next couple of seconds the number 17 (the correct response to our input instruction) should be printed, followed by the word READY and the flashing cursor.

The output from this program is less interesting than another result, which can be seen by LISTing the program after it has run. Line 20 will be found to have changed from a meaningless string of π's to:

20 PRINT 2+3*5

the very same instruction that was entered while the program was running. In fact the π's were there only to reserve a set of twenty addresses into which the new line was POKEd. There are still twenty characters in line 20, but most of them are now blanks, which are not visible in the listing and do not cause any problems when the line is executed. Since the number of characters in the line is unchanged, the program can be run repeatedly, altering the contents of this line each time.

Maybe you now think that the program is far more complicated than it needs to be to achieve the result of poking the desired instruction into line 20. Surely all that needs to be done is to poke the ASCII numbers corresponding to P,R,I,N,T,etc. into the 20 addresses of the line. Write your own program to do this, if you want, but you're in for a disappointment. When your program is working properly, the new line will LIST exactly as it should, but when you try to execute it you will get a SYNTAX ERROR. The problem is that BASIC instruction words are stored in PET's memory as single token characters (the LISTing routine translates them back into English words) and the machine cannot understand them except in token form.

The demonstration program not only enters the new line in correct token form, it also does so without invoking the line editor, which would cause the erasure of any pre-existing variables, strings, etc. in memory. To provide this, enter "X=5" in direct mode, then start the program without erasing memory by entering "GOTO 10". Put in any simple BASIC instruction, such as PRINT "DONE", when line 110 asks for it. When the program has finished, enter PRINT X in direct mode. The value 5 will be returned, showing that it is still in memory.

Let's now look at the program to see how it works. The first few lines are arranged so that the changeable line is as near the start of the program as possible. This makes its addresses easy to find (e.g. by using the machine-language monitor), and also protects them from being messed around with by any editing of the rest of the program. Lines 100 to 130 take the input instruction, in string form, prefix it with "GOTO 200", and then POKE it, letter by letter, into the second cassette buffer in the PET starting several characters from the start of the buffer. This buffer is used by the program for one of its originally intended purposes, as an input/output device. Line 140 contains a set of POKEs which "persuade" the PET that a second cassette unit is present, that its "Play" key is pressed, and that this is the device from which it should take its next input and to which it should make its next output, starting at the beginning of the buffer.

At line 150, an END instruction is encountered. This makes the PET print READY into the start of the second cassette buffer and then to take the instructions which are waiting for it in the later locations in the buffer. These are first translated into token form (just what we wanted!) and entered into another buffer, from which they are later read by the routines which execute BASIC instructions. However, the first instruction to be executed is GOTO 200, which re-starts the program and leaves the instructions which we want to put into line 20, in token form, in the basic input buffer.

Line 200 restores the keyboard as the PET's input device and the screen as its output device. Lines 210 to 260 copy the desired text from the basic input buffer into the addresses occupied by line 20, then line 300 fills the remainder of these addresses with blanks. Finally, line 400 demonstrates that the new line actually works, and the machine prints the word READY on the screen as the program ends.

There is an obvious criticism which can be made of this program as it stands. Why go to the trouble of copying the instructions into line 20 when they could have been executed directly from the basic input buffer? This is a valid criticism, provided the instructions are to be executed only once, and that they can legally be performed in direct mode. In practical applications of this technique, however, one or the other of these conditions is often not true.

So much for the mechanics of simple self-modifying programs. Their potential usefulness is great. They represent a class of interactive programs which allow the user not only to supply the values of variables and to make simple choices, but also to give precise logical instructions to the program as it operates.

Probably the simplest applications are in general mathematical programs. These can easily be written to draw the graph of any function, to use an iterative method to solve any equation, or any similar task. The program asks the user to enter the equation he is interested in, and then writes this into one of its own lines. This line can later be executed as many times as necessary for the program to complete its job.

I have recently written a self-modifying program with a very different purpose: to teach students how to set up computer programs in the form of flow-charts. The program allows a student to draw a flow-chart on the PET screen, with BASIC instructions placed on the diagram in the appropriate places. When the diagram is complete, its instructions can be executed without the student having to write a conventional program. The PET simply follows the logic lines on the diagram. When an instruction is encountered, it is written into one of several modifiable lines in the main program and executed appropriately.

I am sure there are thousands of other applications, but I'll leave them for you to discover... ©

VIC-20 Update

TINYMON1: A Simple Monitor For The VIC

Jim Butterfield
Toronto, Canada

One of the things you may miss on the VIC is a Machine Language Monitor: it's not there.

Commodore will be releasing a very powerful MLM on a plug-in cartridge, and serious programmers will certainly want to use it. But for occasional use, a tape-loadable MLM might be very handy.

Here's an early version that may be of use. It should fit on any VIC, with or without extra memory added; and it honors all the commands from the built-in Monitors we know from PET/CBM usage. One minor syntax change: the two addresses of the Memory display command (.M) should be separated by a space rather than a comma.

It's not really practical to type TINYMON directly into a VIC. DATA statements in decimal would take up more room than is available in small VICs; and hex entry would need an MLM to be in place already. So I've prepared the program so that it can be entered on a PET and saved on tape. After it's been created once, the VIC can make its own copies. You'll need a PET with Upgrade ROM or 4.0 ROM to do the job, since the Original ROM PETs don't have a Machine Language Monitor and things would get too complicated.

TINYMON loads like a BASIC Program, and copies can be made with a simple LOAD and SAVE sequence as you would do with BASIC. When you load TINYMON and say RUN, however, some interesting things happen ... the monitor system is repacked into the top of memory, and it will stay there until you turn the power off. You can say .X to return to BASIC and load and run BASIC programs, providing they are not too big. TINYMON grabs about 760 bytes of memory, so you lose a little space.

Find A Zero

Once you're back in BASIC, the question arises: how can you invoke TINYMON when desired? Not an easy trick, since memory is more mobile in the VIC than in the PET/CBM. The thing to do is to find a zero value in memory and SYS to that location. If you have a basic (5K) VIC, SYS 4096 is safe. The sure way is to PEEK first and ensure that there's a zero there (location 10 is often zero).

TINYMON1 must be considered preliminary. It was designed with two major considerations: to use minimum space, and to automatically load into any VIC regardless of the memory fitted. The space consideration is fairly obvious: with 3500-odd bytes available on a small VIC, you want to use up as little as posible. The automatic load feature was tricky to implement; VIC may relocate programs as it loads. What's more, the screen area tends to move around as you add memory.

I scratched my head over the .S (Save) command. If VIC automatically relocates programs during loading, will a SAVEd Machine Language program be safe? As it turns out, VIC has a new tape format available – when a tape is written, it may be defined as "absolute" and will not relocate when it loads. This seems the best compromise, but it has one drawback – the PET/CBM won't load this type of tape. Perhaps that's a design decision that will need to be revised...

Finding Space In Zero Page

VIC is desperately short of zero page space; machine language programmers will have to cope with the shortage as best they can. I have used the same locations that the big Commodore MLM is expected to use. There's a difference, however, the Commodore job will swap out selected parts of zero page and put them back later; I didn't want to give up the space for that kind of luxury. As a result, you may be annoyed by some locations that are disturbed by TINYMON1.

For those unfamiliar with the PET/CBM Machine Language Monitor, the commands are:

.R – display 6502 registers;
Users can use screen editing to type over a display and change the registers;

.M FFFF TTTT – display memory (from .. to);

Users can use screen editing to type over a display and change memory;

.X – exit to BASIC;
It may be wise to type CLR in BASIC after exiting;

.G AAAA – GOTO (execute) address;

.S "PPPP",01,FFFF,TTTT – Save (program-name, device, from, to);

.L "PPPP" – Load (program-name)

There's a delicate tradeoff between features and memory space. There will undoubtedly be other small monitors with a different balance. In any case, I wrote one because I had nothing ... and others in the same position will undoubtedly greet TINYMON1 with glad cries.

Program 1: TINYMON1

Enter on a PET/CBM, using the Machine Language Monitor. Do not try to RUN, but follow your entry with the checksum program, Program 2.

First, make the following change:

```
.: 0028 01 04 18 08 18 08 18 08
```

Now, enter TINYMON1:

```
.: 0400 00 18 04 64 00 99 22 93
.: 0408 11 11 12 1d 1d 1d 20 54
.: 0410 49 4e 59 4d 4f 4e 20 00
.: 0418 31 04 6e 00 99 22 11 20
.: 0420 4a 49 4d 20 42 55 54 54
.: 0428 45 52 46 49 45 4c 44 22
.: 0430 00 4c 04 78 00 9e 28 c2
.: 0438 28 34 33 29 aa 32 35 36
.: 0440 ac c2 28 34 34 29 aa 30
.: 0448 37 38 29 00 00 00 ea ea
.: 0450 a5 2d 85 22 a5 2e 85 23
.: 0458 a5 37 85 24 a5 38 85 25
.: 0460 a0 00 a5 22 d0 02 c6 23
.: 0468 c6 22 b1 22 d0 3c a5 22
.: 0470 d0 02 c6 23 c6 22 b1 22
.: 0478 f0 21 85 26 a5 22 d0 02
.: 0480 c6 23 c6 22 b1 22 18 65
.: 0488 24 aa a5 26 65 25 48 a5
.: 0490 37 d0 02 c6 38 c6 37 68
.: 0498 91 37 8a 48 a5 37 d0 02
.: 04a0 c6 38 c6 37 68 91 37 18
.: 04a8 90 b6 c9 bf d0 ed a5 37
.: 04b0 85 33 a5 38 85 34 6c 37
.: 04b8 00 00 00 00 00 00 00 00
.: 04c0 bf 78 ad fe ff 00 ae ff
.: 04c8 ff 00 8d 16 03 8e 17 03
.: 04d0 a9 80 20 90 ff 58 00 00
.: 04d8 68 85 05 68 85 04 68 85
.: 04e0 03 68 85 02 68 85 01 68
.: 04e8 85 00 00 ba 86 06 38 a5
.: 04f0 01 e9 02 85 01 a5 00 00
.: 04f8 e9 00 00 85 00 00 20 b2
.: 0500 fe 00 a2 42 a9 2a 20 db
.: 0508 fd 00 a9 52 d0 1c a9 3f
.: 0510 20 d2 ff 20 b2 fe 00 a9
.: 0518 2e 20 d2 ff a9 00 00 85
.: 0520 27 20 40 fe 00 c9 2e f0
.: 0528 f9 c9 20 f0 f5 a2 07 dd
.: 0530 e6 ff 00 d0 12 85 1c 8a
.: 0538 0a aa bd ee ff 00 85 c1
.: 0540 bd ef ff 00 85 c2 6c c1
.: 0548 00 00 ca 10 e6 4c 4b fd
.: 0550 00 20 bd fd 00 90 f8 20
.: 0558 ee fd 00 20 bd fd 00 90
.: 0560 f0 20 ee fd 00 20 4c fe
.: 0568 00 f0 1f 20 b2 fe 00 a2
.: 0570 2e a9 3a 20 db fd 00 20
.: 0578 c5 fd 00 a9 05 20 6f fe
.: 0580 00 a5 c3 c5 c1 a5 c4 e5
.: 0588 c2 b0 df 4c 50 fd 00 4c
.: 0590 50 fd 00 20 fe fd 00 85
.: 0598 c1 86 c2 60 a5 c2 20 cc
.: 05a0 fd 00 a5 c1 48 4a 4a 4a
.: 05a8 4a 20 e4 fd 00 aa 68 29
.: 05b0 0f 20 e4 fd 00 48 8a 20
.: 05b8 d2 ff 68 4c d2 ff 18 69
.: 05c0 f6 90 02 69 06 69 3a 60
.: 05c8 a2 02 b5 c0 48 b5 c2 95
.: 05d0 c0 68 95 c2 ca d0 f3 60
.: 05d8 20 0d fe 00 90 07 aa 20
.: 05e0 0d fe 00 90 01 60 4c 4b
.: 05e8 fd 00 a9 00 00 85 2a 20
.: 05f0 40 fe 00 c9 20 f0 f9 20
.: 05f8 20 fe 00 90 17 20 40 fe
.: 0600 00 c9 30 90 10 20 35 fe
.: 0608 00 06 2a 06 2a 06 2a 06
.: 0610 2a 05 2a 85 2a 38 60 c9
.: 0618 3a 08 29 0f 28 90 02 69
.: 0620 08 60 20 cf ff c9 0d d0
.: 0628 f8 68 68 4c 50 fd 00 a5
.: 0630 91 c9 fe d0 05 08 20 cc
.: 0638 ff 28 60 20 61 fe 00 2c
.: 0640 2d 91 30 f8 60 20 4c fe
.: 0648 00 d0 08 a9 03 85 9a a9
.: 0650 00 00 85 99 60 85 1e a0
.: 0658 00 00 20 af fe 00 b1 c1
.: 0660 20 cc fd 00 20 a4 fe 00
.: 0668 c6 1e d0 f1 60 20 0d fe
.: 0670 00 90 0b a2 00 00 81 c1
.: 0678 c1 c1 f0 03 4c 4b fd 00
.: 0680 20 a4 fe 00 c6 1e 60 a9
.: 0688 02 85 c1 a9 00 00 95 c2
.: 0690 a9 05 60 e6 c1 d0 06 e6
.: 0698 c2 d0 02 e6 27 60 a9 20
```

```
.:  06a0 2c a9 0d 4c d2 ff a2 00
.:  06a8 00 bd d0 ff 00 20 d2 ff
.:  06b0 e8 e0 16 d0 f5 20 b2 fe
.:  06b8 00 a2 2e a9 3b 20 db fd
.:  06c0 00 a5 00 00 20 cc fd 00
.:  06c8 a5 01 20 cc fd 00 20 99
.:  06d0 fe 00 20 6f fe 00 4c 50
.:  06d8 fd 00 20 fe fd 00 85 01
.:  06e0 86 00 00 20 99 fe 00 85
.:  06e8 1e 20 83 fe 00 d0 fb f0
.:  06f0 ea 20 bd fd 00 a9 05 85
.:  06f8 1e 20 83 fe 00 d0 fb f0
.:  0700 dc 20 cf ff c9 0d f0 07
.:  0708 20 bd fd 00 85 01 86 00
.:  0710 00 a6 06 9a a5 00 00 48
.:  0718 a5 01 48 a5 02 48 a5 03
.:  0720 a6 04 a4 05 40 78 a6 06
.:  0728 9a 6c 02 c0 4c 4b fd 00
.:  0730 a0 01 84 ba 84 b9 88 84
.:  0738 b7 84 90 84 93 a9 02 85
.:  0740 bc a9 40 85 bb 20 cf ff
.:  0748 c9 20 f0 f9 c9 0d f0 1a
.:  0750 c9 22 d0 d9 20 cf ff c9
.:  0758 22 f0 26 c9 0d f0 0b 91
.:  0760 bb e6 b7 c8 c0 10 f0 c5
.:  0768 d0 ea a5 1c c9 4c d0 e2
.:  0770 a9 00 00 20 d5 ff 20 58
.:  0778 fe 00 a5 90 29 10 d0 f0
.:  0780 4c 50 fd 00 20 cf ff c9
.:  0788 0d f0 e2 c9 2c d0 f0 20
.:  0790 0d fe 00 29 0f f0 d3 c9
.:  0798 03 f0 fa 85 ba 20 cf ff
.:  07a0 c9 0d f0 ca c9 2c d0 e6
.:  07a8 20 bd fd 00 20 cf ff c9
.:  07b0 2c d0 f4 20 fe fd 00 85
.:  07b8 ae 86 af 20 cf ff c9 20
.:  07c0 f0 f9 c9 0d d0 ec a5 1c
.:  07c8 c9 53 d0 f8 20 b2 fe 00
.:  07d0 a9 01 85 b9 20 82 f6 4c
.:  07d8 50 fd 00 0d 20 20 20 50
.:  07e0 43 20 20 53 52 20 41 43
.:  07e8 20 58 52 20 59 52 20 53
.:  07f0 50 4d 52 58 47 3a 3b 4c
.:  07f8 53 86 fd 00 b7 fe 00 23
.:  0800 ff 00 02 ff 00 f4 fe 00
.:  0808 e1 fe 00 2d ff 00 2d ff
.:  0810 00 1b fd 00 00 00 00 00
```

Whew! TINYMON1 for the VIC is now entered. Check it with the following program:

Program 2: A Checking Program

Type the following direct line on the screen of your PET/CBM:

```
forj=1024to2071step8:t=0:fork=jtoj+7:t=t+peek(k):next:?t;:next
```

You should see the following numbers appear on the screen of your PET. Check them carefully. Each one represents one line of entry, starting at 0400 hexadecimal. If any of these totals is wrong, you've entered the line incorrectly.

The numbers in brackets appearing to the right won't appear on your screen; they are there to help you locate an incorrect line.

When you are satisfied that the program is entered correctly, SAVE it to cassette tape. It may now be loaded into your VIC.

462	255	506	399	575	541	592	511	(0400)
769	620	756	780	802	910	886	853	
801	784	876	840	835	1383	753	0	
1422	589	816	720	584	680	535	576	
944	972	1130	845	876	1357	1010	1188	(0500)
1311	852	898	1109	1125	897	809	1021	
1340	1078	1005	1212	905	902	770	1239	
762	1133	1388	652	659	629	1072	803	
748	150	617	413	1020	1030	1057	818	(0600)
944	844	705	831	939	1072	639	1033	
943	824	1137	970	929	1149	1395	940	
654	840	807	926	706	1146	1015	1146	
1175	742	563	645	695	860	1064	1042	(0700)
1235	1202	1355	922	1445	1346	789	1068	
1104	1204	975	1306	1339	1169	1168	1210	
1340	1204	972	522	460	520	591	942	
1010	1079	280						(0800)

©

VIC Color Tips

Charles Brannon
Editorial Assistant

Users of other computers, such as the ATARI or Apple, will find the VIC harder to use for color graphics because there are no dedicated statements for controlling these features. First time users will not know the difference, but this article should make things easier. Before we begin, it should be noted that there will soon be available a VIC Super Expander Cartridge that will add special sound and graphics commands to BASIC, as well as adding 3K of memory.

"Poking" Graphics

The only command that can be used for graphics besides PRINT is POKE. POKE places a number into a memory location. Its format is POKE A,B. A is the memory location, and B is the value to be placed there, zero to 255. Some spots in memory can control Input/Output chips, such as the Video Interface Chip inside of the VIC. Location 36879 is the control register for background and border colors. To get each combination, you place a number from zero to 255 into 36879, as previously mentioned. For any particular combination, you can look up the colors in the table at the end of this article (Table 2). There is an easier way, however, at least from a programming standpoint.

An Easier Way

The DEF FN command allows the programmer to design his own function. The VIC has, for example, the standard INT function. INT(X) will give you the whole-number value of the argument X by dropping the fractional portion. It does not round X. To provide a rounding-up function, we can use the DEF FN command. To round dollar and cents amounts, the statement DEF FNR(V) = INT (V*100 + .5)/100 is executed at the start of the program. After that, FNR(X) will give you the rounded version of X, or any value in parentheses. PRINT FNR(3.1415927) will return 3.14, while PRINT FNR(500.076) will give 500.08 The R after the FN is a label to remind you what the function does. Here R stands for Round. These labels have the same format as numerical variable names.

What we want to do is to devise a formula which will give us the right number from the table for each color, one to sixteen. We will give the background color from one to sixteen through the FN routine, and it will give us the number ready for POKEing. To get any background color from any of the sixteen possible colors, just multiply the color number by 16 and then subtract eight. We can code this as DEF FNC(V) = V*16-8. Remember, V is just a dummy variable used to define the relationship of the argument (what we give the routine) in the formula. Next we use a little shorthand. The number 36879 (the color control) is a little hard to remember, and it does not look much different than any other memory location. We will make it easier to remember (make it *mnemonic*) by making it a variable, SCREEN = 36879. Now we can call forth any of our sixteen colors with the statement: POKE SCREEN, FNC (*color*), where *color* is the number from one to sixteen. This almost looks like a real graphics command.

Adding Border Colors

What about the border colors? In addition to the background, you can have eight border colors, numbered from zero to seven. This is one less than the corresponding number on the color keys (CTRL-6 would be 5). Now just take this number and add it to the number that you POKE into SCREEN. Now we just use: POKE SCREEN, FNC(*color*) + *border*, where *border* is the border color, zero to seven. If you don't use border colors, or don't add anything to FNC(color), then the border will be black.

Remember that if the background is the same color as the text, the cursor will become invisible. If you need to, set things straight with POKE 36879,27 or hold down RUN/STOP and press RESTORE to reset.

The little program at the end of this article demonstrates what I've been talking about by displaying all the combinations of screen and border colors. It's simple to figure out so look it over, and get to work on your VICtorious applications!

Table 1. Screen/Border Colors

Screen		Border	
1	Black	0	Black
2	White	1	White
3	Red	2	Red
4	Cyan	3	Cyan
5	Purple	4	Purple
6	Green	5	Green
7	Blue	6	Blue
8	Yellow	7	Yellow
9	Orange		
10	Light Orange		
11	Pink		
12	Light Cyan		
13	Light Purple		
14	Light Green		
15	Light Blue		
16	Light Yellow		

```
100 REM * ANOTHER RAINBOW *
110 DEF FNC(V)=V*16-8
120 SCREEN=36879
130 FOR BK=1 TO 16
140 PRINT "{CLEAR}{WHT}";
150 IF BK>1 THEN PRINT "{BLK}";
160   PRINT "SCREEN";BK
170   FOR BD=0 TO 7
180   POKE SCREEN,FNC(BK)+BD
190   PRINT,"BORDER";BD
200   FOR W=1 TO 500:NEXT W
210  NEXT BD
220 NEXT BK
230 POKE SCREEN,27
240 END
```

Table 2. POKE Values

BACKGROUND #	BORDER							
	0	1	2	3	4	5	6	7
1:	8	9	10	11	12	13	14	15
2:	24	25	26	27	28	29	30	31
3:	40	41	42	43	44	45	46	47
4:	56	57	58	59	60	61	62	63
5:	72	73	74	75	76	77	78	79
6:	88	89	90	91	92	93	94	95
7:	104	105	106	107	108	109	110	111
8:	120	121	122	123	124	125	126	127
9:	136	137	138	139	140	141	142	143
10:	152	153	154	155	156	157	158	159
11:	168	169	170	171	172	173	174	175
12:	184	185	186	187	188	189	190	191
13:	200	201	202	203	204	205	206	207
14:	216	217	218	219	220	221	222	223
15:	232	233	234	235	236	237	238	239
16:	248	249	250	251	252	253	254	255

©

VIC Memory Map Above Page Zero

Jim Butterfield
Toronto, Canada

Editor's Note: Next month we'll have a VIC zero page map and Jim's comments on the VIC's memory. — RTM

0100-103E	256-318	Tape error log
0100-01FF	256-511	Processor stack area
0200-0258	512-600	Basic input buffer
0259-0262	601-610	Logical file table
0263-026C	611-620	Device # table
026D-0276	621-630	Sec Adds table
0277-0280	631-640	Keybd buffer
0285	645	Serial bus timeout flag
0286	646	Current color code
0287	647	Color under cursor
0288	648	Screen memory page
0289	649	Max size of keybd buffer
028A	650	Repeat all keys
028B	651	Repeat speed counter
028C	652	Repeat delay counter
028D	653	Keyboard Shift/Control flag
028E	654	Last shift pattern
028F-0290	655-656	Keyboard table settup pointer
0291	657	Keymode (Kattacanna)
0292	658	0=scroll enable
0293	659	VIC chip control
0294	660	VIC chip command

0295-0296	661-662	Bit timing	
0297	663	RS-232 status	
0298	664	# bits to send	
0299-029A	665	RS-232 speed/code	
029B	667	RS232 receive pointer	
029C	668	RS232 input pointer	
029D	669	RS232 transmit pointer	
029E	670	RS232 output pointer	
029F-02A0	671-672	IRQ save during tape I/O	
0300-0301	768-769	Error message link	
0302-0303	770-771	Basic warm start link	
0304-0305	772-773	Crunch Basic tokens link	
0306-0307	774-775	Print tokens link	
0308-0309	776-777	Start new Basic code link	
030A-030B	778-779	Get arithmetic element link	
0314-0315	788-789	Hardware interrupt vector	(EABF)
0316-0317	790-791	Break interrupt vector	(FED2)
0318-0319	792-793	NMI interrupt vector	(FEAD)
031A-031B	794-795	OPEN vector	(F40A)
031C-031D	796-797	CLOSE vector	(F34A)
031E-031F	798-799	Set-input vector	(F2C7)
0320-0321	800-801	Set-output vector	(F309)
0322-0323	802-803	Restore I/O vector	(F3F3)
0324-0325	804-805	INPUT vector	(F20E)
0326-0327	806-807	Output vector	(F27A)
0328-0329	808-809	Test-STOP vector	(F770)
032A-032B	810-811	GET vector	(F1F5)
032C-032D	812-813	Abort I/O vector	(F3EF)
032E-032F	814-815	USR vector	(FED2)
0330-0331	816-817	LOAD link	
0332-0333	818-819	SAVE link	
033C-03FB	828-1019	Cassette buffer	
0400-0FFF	1024-4095	3K RAM expansion area	
1000-1FFF	4096-8191	Normal Basic memory	
2000-7FFF	8192-32767	Memory expansion area	
8000-8FFF	32768-36863	Character bit maps	
9000-900F	36864-36879	Video Interface Chip	
9110-912F	37136-37167	6522 Interface Chips	
9400-95FF	37888-38399	Alternate Colour Nybble area	
9600-97FF	38400-38911	Main Colour Nybble area	
A000-BFFF	40960-49151	Plug-in ROM area	
C000-FFFF	49152-65535	ROM: Basic and Operating System	

VIC usage: The 6560 V. I. Chip

$9000	Interlace (0)	Left Margin (=5)		36864
$9001	Top Margin (=25)			36865
$9002	Sc. Adds b9	# Columns (=22)		36866
$9003	b0	# Rows (=23)	Double Char	36867
$9004	Raster Value In: 68-b1			36868
$9005	Screen Address b13-b10	Character Address b13-b10		36869
$9006	Light Pen	Horizontal		36870
$9007	(option)	Vertical		36872
$9008	Potentiometer	X		36872
$9009	Sense (option)	Y		36874
$900A	Λ Voice 1			36874
$900B	Λ Voice 2			36875
$900C	Λ Voice 3			36876
$900D	Λ Noise			36877
$900E	Multi-Colour Mode (=0)	Sound Amplitude		36878
$900F	Screen Background	Fore/Back-Ground	Frame Color	36879

Values, where shown, are the normal default VIC values.

Light Pen and Potentiometer are implemented in hardware but not used in ROM programs.

VIC Usage: The 6522-A

$9110	RS-232 or Parallel User Port						37136
$9111	unused — see $911F						37137
$9112	DDRB (for $9110)						37138
$9113	DDRA (for $911F)						37139
$9114	T1-L	RS-232 Send speed;					37140
$9115	T1-H	Tape write timing					37141
$9116	T1 Latch						37142
$9117	T1 Latch						37143
$9118	T2-L	RS-232					37144
$9119	T2-H	Input timing					37145
$911A	Shift Register (not used)						37146
$911B	T1 Control	T2C	SR Control		PBLE	PALE	37147
$911C	RS-232 Send		Cb1 cont	Tape motor		CA1 cont	37148
$911D / $911E	NMI:	T1	T2	CB1: RS-232 IN	CA1: RSTR butn		37149 / 37150
$911F	ATN out	Tape sens	Joystick ----- Butn 0 1 2		In; Serl	In: Clok	37151

VIC Usage; The 6522-B

$9120	RS232 in / Joy 3	Tape out	---- Keyboard Row Select ----			37152
$9121	Keyboard Row Input					37153
$9122	DDRB (for $9120)					37154
$9123	DDRA (for $9121)					37155
$9124	T1-L	Cassette tape read;				37156
$9125	T1-H	Keyboard (interrupt)				37157
$9126	T1 Latch	timing				37158
$9127	T1 Latch					37159
$9128	T2-L	Serial Bus timeout;				37160
$9129	T2-H	Cassette R/W timing				37161
$912A	Shift Register (unused)					37162
$912B	T1 Control T2C	SR Control		PBLE	PALE	37163
$912C	Serial bus out		Clock line out		CA1 Contl	37164
$912D / $912E	IRQ:	T1	T2	CA1: Tape		37165 / 37166
$912F	unused — see $9121					37167

©

ZAP!!

Dub Scroggin
Ft. Walton Beach, FL

"Zap!!" is an exciting and challenging VIC-20 game program that is designed for up to six players and up to five rounds per player. Each player may select from any of five skill levels and may change levels each round, if desired. Using keyboard controls, players maneuver a block around the screen and through a field of randomly placed and color coded graphic figures. The object is to run over and erase as many of these figures as possible in two minutes, but also to avoid running into asterisks and being zapped. After the player block is moved it cannot be stopped, but the direction of movement may be changed. The higher the skill level, the faster the block moves and the more asterisks there are. The number of scoring figures is increased also so that a higher score is possible too.

The figures on the screen count differently toward the score. If a player is "zapped," he retains his score, but his time is over. Players may run off the screen, but may strike a hidden asterisk if they do so. A vertical wraparound feature prevents players from wandering too far off the screen. A variety of colors, graphics, and sound effects add excitement to the program.

As with most computer games, proficiency at Zap! will take some practice and a lot of concentration.

The player block is moved around the screen by the computer PEEKing at the keyboard to determine the value of the last control key pressed. A direction factor is then assigned to the variable DR (steps 590-620). When moving left, DR is -1, right is 1, up is -22 and down is 22. This factor is then added to the position of the block (B) (step 650). The old block is then erased by POKEing it to 32 (blank) and a new one is placed in position (step 570). This all happens so quickly that the illusion of motion is created.

Scoring and zaps are determined by PEEKs at the block's position to see if any other figure is there (steps 670-720). Depending on the figure found at "PEEK (B)", a score is assigned and the loop continues, or if the figure is an asterisk, a "Zap!!" routine is initiated and the round ends.

In each pass through the loop (steps 550-780), several things happen or are checked for. The elapsed time is printed and there is a check to see if the time is up. If so, the loop is terminated and the round ends. A block is POKEd into position B. Steps 580 and 585 provide the vertical wraparound effect. A check is made for direction change input from the keyboard. A tone is sounded based on the current direction of movement. The old position of the block is erased and a new position is calculated. A check is made to see if any figures have been struck. If so, they are either scored or, in the case of an asterisk, the loop is terminated. After a new total score is calculated and displayed, the loop begins again.

Steps 640 and 760 are time delay steps to slow the block's motion and to increase speed as the skill level increases. If a faster or slower movement is desired, these steps may be altered.

A number of REMarks have been included in the program listing as an aid to understanding it, but I recommend that they not be typed in on your computer. This program uses all but about 250 bytes of standard VIC-20 memory and including all the remarks may result in an "out of memory" error.

Good luck and I hope you enjoy the game.

```
10 PRINT"{CLEAR}"
20 DIM PL(6),R(5)
30 FORY=1TO5:FORX=1TO6:Z(X,Y)=0:NEXTX:NE
   XTY
40 C=30720:TB=0:TS=0
50 POKE36879,239
60 CP=0:GOTO810
70 PRINTTAB(3)"{06 DOWN}BY DUB SCROGGIN"

80 REM-404 WOODROW ST.,FT. WALTON BEACH,
   FL 32548
90 CP=1
100 FORT=1TO2000:NEXTT
110 PRINT"{CLEAR}"
120 PRINTTAB(5)"{DOWN}DIRECTIONS"
130 PRINTTAB(5)"7777777777"
140 PRINT"{DOWN}YOU WILL HAVE 2 MIN.":PRI
    NT"TO GET YOUR BEST SCORE"
150 PRINT"{YEL}MOVEMENT:":PRINT"{DOWN}CRS
    R DN=LEFT":PRINT"CRSR RT=RIGHT":
    PRINT"F5=UP"
160 PRINT"F7=DOWN":PRINT"{HOME}{04 DOWN}"

170 PRINTTAB(14)"{03 DOWN}{WHT}SCORING:"
180 PRINTTAB(14)"{BLK}W=1"
190 PRINTTAB(14)"{CYN}Q=2"
200 PRINTTAB(14)"{YEL}Z=3"
210 PRINTTAB(14)"{RED}S=5"
220 PRINTTAB(14)"{WHT}A=10"
230 PRINT"{DOWN}YOU ARE: {BLU}{REV} {OFF}
    "
240 PRINT"{DOWN}DON'T HIT A {PUR}*{BLU} O
    R":PRINT"YOU WILL GET {PUR}ZAPPE
    D."
250 PRINT"{WHT}{DOWN}PRESS ANY KEY TO STA
    RT"
260 GETA$:IFA$=""THEN260
270 PRINT"{CLEAR}{WHT}HOW MANY ROUNDS (1-
    5)"
280 INPUTRN:IFRN<1ORRN>5THENPRINT"HUH?":G
    OTO270
290 PRINT"{DOWN}HOW MANY PLAYERS":PRINT"(
    1-6)";
300 INPUTPN:IFPN<1ORPN>6THENPRINT"HUH?":G
    OTO290
310 FORR=1TORN
320 FORP=1TOPN:PRINT"{BLU}{DOWN}PLAYER #"
    ;P
330 PRINT"{DOWN}WHAT SKILL LEVEL?"
340 PRINT"PRESS 0,1,2,3 OR 4";
350 INPUT S
360 IFS>4 ORS<0THENPRINT"HUH?":GOTO340
370 PRINT"{CLEAR}{BLU}{REV}SCORE TO BEAT:
    ";TB:PRINT"{REV}SKILL LEVEL:";SL

380 PRINT"{REV}PLAYER #";PB
390 FORT=1TO2000:NEXTT:PRINT"{CLEAR}"
400 DEF FN A(L)=INT(RND(1)* L )+7702
410 FORF=1TO40-2*S:D=FNA(483)
420 POKED,87:POKED+C,0:NEXTF
430 FORF=1TO25:D=FNA(483)
440 POKED,81:POKED+C,3:NEXTF
450 FORF=1TO10+4*S:D=FNA(505)
460 POKED,42:POKED+C,4:NEXTF
470 FORF=1TO19:D=FNA(483)
480 POKED,90:POKED+C,7:NEXTF
490 FORF=1TO14:D=FNA(483)
500 POKED,83:POKED+C,2:NEXTF
510 FORF=1TO9+S:D=FNA(505)
```

```
520 POKED,65:POKED+C,1:NEXTF
530 B=7932
540 TI$="000000"
550 PRINT"{HOME}TIME:";120-INT(TI/60);"{L
    EFT}  "
560 IFTI/60>=120THENGOTO930
570 POKEB,160:POKEB+C,6
580 IFB<7636THENB=8229+B-7635
585 IFB>8229THENB=7636+B-8230
590 IFPEEK(197)=31THENH=190:DR=-1:GOTO630

600 IFPEEK(197)=23THENH=200:DR=1:GOTO630
610 IFPEEK(197)=55THENH=210:DR=-22:GOTO63
    0
620 IFPEEK(197)=63THENH=220:DR=22
630 POKE36878,15:POKE36876,H
640 FORT=1TO30-5*S:NEXTT
650 POKEB,32:B=B+DR
660 SC=0
670 IFPEEK(B)=42THENGOTO790
680 IFPEEK(B)=87THENSC=1:GOTO740
690 IFPEEK(B)=81THENSC=2:GOTO740
700 IFPEEK(B)=90THENSC=3:GOTO740
710 IFPEEK(B)=83THENSC=5:GOTO740
720 IFPEEK(B)=65THENSC=10:GOTO740
730 GOTO760
740 TS=TS+SC
750 POKE36878,15:POKE36876,160+PEEK(B)
760 FORT=1TO30-5*S:NEXTT
770 PRINT"{HOME}{DOWN}SCORE=";TS
780 GOTO550
790 POKE36878,15
800 FORPI=1TO40:POKE36876,180-PI:NEXTPI
810 PRINT"{PUR}{RIGHT}{06 DOWN}&&&& &&&& ~
    &&&& && &&"
820 PRINT"{RIGHT}&&&& &&&& &&&& && &&"
830 PRINT"{RIGHT}   & &  & &  & && &&"
840 PRINT"{RIGHT}  &  &&&& &&&& && &&"
850 PRINT"{RIGHT} &   &&&& &&&& && &&"
860 PRINT"{RIGHT}&&&& &  & &"
870 PRINT"{RIGHT}&&&& &  & &    && &&{BLU
    }"
880 IFCP=0THEN70
890 POKE36878,0:POKE36876,0
900 FORT=1TO2000:NEXTT:PRINT"{CLEAR}"
910 PRINT"{WHT}{DOWN}{REV}YOU LASTED";INT
    (TI/60)-3;"{LEFT} SECONDS{OFF}"
920 GOTO970
930 POKE36878,15:FORAC=1TO80:POKE36876,21
    0-AC:NEXTAC
940 POKE36876,0:POKE36878,0
950 PRINT"{CLEAR}"
960 PRINT"{HOME}{04 DOWN} {BLU}{REV}....T
    IME IS UP...."
970 PRINTTAB(6)"{DOWN}{REV}{WHT}SCORE=";T
    S
980 Z(P,R)=Z(P,R-1)+TS
990 IFTS>TBTHENTB=TS:SL=S:PB=P
1000 PRINT"{DOWN}{BLK}ROUND #:";R;"{DOWN}"

1010 FORX=1TOPN
1020 PRINT"{YEL}PLAYER #";X;":";Z(X,R)
1030 NEXTX
1040 TS=0:DR=0:H=0:PRINT"{DOWN}"
1050 NEXTP:NEXTR
1060 END
```

©

CAPUTE!

1. **COMPUTE!** #12, pg. 94. The authors suggest that the following lines should be changed to:

```
255 UG=GU:PRINT
257 IF GU=0 THEN 270
290 WD=WI: WF=WI: WT=(12-WI)/2
370 IF WI>4 THEN 300
530 IF WT<0 OR WT=0 THEN WT=1: RN=0
610 IF PEEK(KY)=251 THEN ME=ME-1: KK= -1
1060 IF WD<3 THEN PRINT "LITTLE";: GOTO 1200
1120 PRINT "CHEATER";
1200 PRINT "FOO";
```

2. **COMPUTE!** #17, pg. 112. The following changes to the "Atari Program Library" will lock all cataloged programs. Then, after adding new programs to a disk, only the unlocked (new) programs need to be cataloged:

```
420 IF B<3 THEN ? "DISK IS #";VOL$:
    XIO35,#3,0,0,"D:*.*":?:?"INSERT LIBRARY
    DISK"
370 IF IN$(1,1)="*" OR DSN$="DOS.SYS" OR
    DSN$="DUP.SYS" OR DSN$="MEM.SAV" OR
    DSN$="DISK.CAT" THEN 310
```

3. **COMPUTE!** #17, pg. 143. Mr. Swaim has suggested the following lines as an alternative way to load the X$ array for business keyboard users and to correct an error in transcription:

```
110 X$(1)=CHR$(164):X$(2)=CHR$(175): X$(3)=
    CHR$(185): X$(4)=CHR$(162)
112 X$(5)=CHR$(18)+CHR$(184)+CHR$(146):
    X$(6)=CHR$(18)+CHR$(183)+CHR$(146)
115 X$(7)=CHR$(18)+CHR$(163)+CHR$(146):
    X$(8)=CHR$(18)+CHR$(32)+CHR$(146)
```

4. **COMPUTE!** #17, pg. 152. The correct SYS is 7168 in line 120.

5. **COMPUTE!** #17, pg. 162. Table 1 is missing number 9, Subtraction:

Load FPAC1 with subtrahend

```
LDA AL       source address
LDY AH       for minuend
JSR $C58F
```

(Addressed value is loaded into FPAC2, FPAC1 is subtracted from FPAC2 and result in FPAC1; FPAC2 unchanged.)

Number 6 should include JSR $C0D1.

COMPUTE!'s Listing Conventions

Many programs which are listed in **COMPUTE!** use cursor control keys, color keys, and so forth. We have established a listing convention which we believe eases the task of typing programs in accurately.

Atari Conventions

For the Atari, all the editing and cursor-control characters are spelled out and surrounded by brackets: [CLEAR] for "clear screen." Other characters, such as CTRL-T (the "ball" character) will be listed as the "normal" character, but within brackets: [T]. A series of identical control characters will be indicated by a number within the brackets: [3 DOWN] means type the cursor-down key three times; [12 R] means type CTRL-R twelve times.

Two control characters, [=] and [–] should be shifted. Any reverse field text will be enclosed within vertical lines. (Press the Atari logo key [⅄] for each vertical line you see.)

PET/CBM/VIC Conventions

Generally, PET/CBM/VIC programs will contain bracketed words for any special characters: [DOWN] means the cursor-down key; [3 DOWN] means type the cursor-down key three times.

If a program line runs over onto the next line down, the ~ symbol indicates where the line broke (in case the number of spaces is unclear between quotes). An underline means that that key is shifted.

8032/Fat 40 Conventions

SET WINDOW TOP	[SET TOP]
SET WINDOW BOTTOM	[SET BOT]
SCROLL UP	[SCR UP]
SCROLL DOWN	[SCR DOWN]
INSERT LINE	[INST LINE]
DELETE LINE	[DEL LINE]
ERASE TO BEGINNING	[ERASE BEG]
ERASE TO END	[ERASE END]
TOGGLE TAB	[TGL TAB]
TAB	[TAB]
ESCAPE KEY	[ESC]

All Commodore Machines

CLEAR SCREEN	[CLEAR]
HOME CURSOR	[HOME]
CURSOR UP	[UP]
CURSOR DOWN	[DOWN]
CURSOR RIGHT	[RIGHT]
CURSOR LEFT	[LEFT]
INSERT CHARACTER	[INST]
DELETE CHARACTER	[DEL]
REVERSE FIELD ON	[RVS]
REVERSE FIELD OFF	[OFF]

VIC Conventions

SET COLOR TO BLACK	[BLK]
SET COLOR TO WHITE	[WHT]
SET COLOR TO RED	[RED]
SET COLOR TO CYAN	[CYN]
SET COLOR TO PURPLE	[PUR]
SET COLOR TO GREEN	[GRN]
SET COLOR TO BLUE	[BLU]
SET COLOR TO YELLOW	[YEL]
FUNCTION ONE	[F1]
FUNCTION TWO	[F2]
FUNCTION THREE	[F3]
FUNCTION FOUR	[F4]
FUNCTION FIVE	[F5]
FUNCTION SIX	[F6]
FUNCTION SEVEN	[F7]
FUNCTION EIGHT	[F8]
ANY NON-IMPLEMENTED FUNCTION	[NIM]

©

New Products

New Journal For Math And Science Teachers

A new periodical for educators interested in using computers to teach mathematics and science is *The Journal Of Computers In Mathematics And Science Teaching.* The *Journal* is published by the Association for computers in Mathematics and Science Teaching, a professional non-profit organization. It is published quarterly in September, December, March and June. ACMST membership is $7.00 per year and includes a subscription to the Journal. Single copy price is $2.50. Contact them at P.O. Box 4455, Austin, TX 78765.

Dental Computer Newsletter

The DCN represents an international group of dentists, physicians and office management people who have interests in office computers. Though the emphasis is on micro computers, many members use minis. We cater to all makes and brand names. DCN offers members:

1. A monthly newsletter
2. Software exchange
3. Advice and experience
4. Access to members world wide
5. Computer bulletin board

1982 annual membership dues are $15.00. A little over a dollar per month. Overseas subscriptions are $23.00 (US). Membership runs from January to January. If you join mid year, we will supply you with the year's back issues. Checks can be sent to:

Dental Computer Newsletter
E. J. Neiburger DDS–Editor
1000 North Ave.
Waukegan, IL 60085 USA

Orders: You may order past issues starting with V1:1 (Nov. '78) for $15 per year ... $45 for all past issues. Membership/equipment listings are $5.00. Commercial software and DCN software exchange lists (24 pages+) are $4.00.

Sue Neiburger RN
Managing Editor

E. J. Neiburger DDS
Editor/Publisher

Please feel free to call any Tuesday, Thursday, Saturday or Sunday 9-12 noon (CST) (312)223-5077.

Capital Children's Museum And Reston Publishing Company To Develop Software

The Capital Children's Museum of Washington, DC and Reston Publishing Company (A Prentice-Hall Company) are pleased to announce an agreement to produce a series of educational book/software packages for microcomputers. The series is designed to promote a creative, interactive use of computers by children, parents, and teachers. The product will be implemented initially on the Atari 800 Personal Computer System; design and testing are being managed by Superboots, the software development arm of the Capital Children's Museum.

The first package, entitled *Paint*, will be available in early 1982. *Paint* will be a versatile educational tool that will be suitable for use either in the home or in a classroom setting. The book accompanying the software will be a guide to a wide range of activities which parents or teachers can use to extend a child's interest in computer learning.

Early in 1981, Atari made a significant donation to the Capital Children's Museum in the form of 30 Atari microcomputer systems. This contribution has allowed the Museum to establish a computer learning environment called *Future Center*, to put computer programs in exhibits, and to utilize other computers in Superboots. Superboots is the software development lab where computer programs are created. Software is used both in *Future Center* and in exhibits and is marketed outside the Museum through Reston Publishing Company.

For more information, contact: Bob Evans, Administrator, Superboots, Capital Children's Museum, 800 Third Street, N.E., Washington, DC 20002 (202)543-8600. Nikki Hardin, Editor, Reston Publishing Company, 11480 Sunset Hills Road, Reston, VA 22090 (703)437-8900.

Artworx Announces New Atari Software

Arthur M. Walsh, Manager of Software Products for Artworx Software Company, 150 N. Main St., Fairport, New York 14450, announced the addition of nine

ATARI 48K — w/disk

CRYSTALWARE

TRS-80 48 — w/disk

APPLE 48K — w/disk

THE FINEST IN FANTASY GAME SOFTWARE

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

★★★ ADVENTURE GAMES ★★★

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

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

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

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

★★★ SPACE GAMES ★★★

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

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

★★★ WAR GAMES ★★★

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

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

★★★ ARCADIA ★★★

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

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

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

VISA

12-ADVENTURE PACK- (#1-4) $112

14-THE WARRIOR- (#7-8) $64

13-SPACEOUT- (#5-6) $58

15-ARCADE- (#9-11) $60

master charge

Crystal has many other fine fantasy and space games. For a copy of CRYSTAL VISION which includes a complete catalog please send $3.00 to the address below.

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

(408) 778-2966 **CRYSTAL COMPUTER** **17120 Monterey Rd., Morgan Hill, CA 95037**

new programs to its present line of software for home computers. All nine programs are available for the Atari computer. They include space games (ENCOUNTER AT QUESTAR IV, $23.95; ROCKET RAIDERS, $19.95; SPACE TRAP, $14.95), a flight landing simulator (PILOT, $16.95), an excellent blockade game (BLOCKADE, $14.95), two suspenseful adventures (CRANSTON MANOR, $21.95 diskette; THE VAULTS OF ZURICH, $21.95), a text editor (TEXT EDITOR, $39.95 diskette) and a "player missile" editor (PM EDITOR, $29.95).

These and other Artworx programs are available at computer stores or can be ordered directly from Artworx toll free at 800-828-6573 or 716-425-2833.

Cimarron Releases File Handling System For The Commodore Business Computer

Costa Mesa, CA — CIMARRON CORPORATION announced today that it is making its proprietary file handling system called CMAR available to systems houses and retailers who are developing business applications software for the Commodore line of small business computers.

CMAR is a keyed file access method that provides the foundation for CIMARRON'S Legal Time Accountinng and Medical Accounting packages marketed by Commodore nationally. CMAR is compatible with all present Commodore disc subsystems utilizing the existing disc format. It is written in 6502 machine language and interacts directly with Commodore Basic 4.0.

With its post "Binary Tree" technology referred to as *Inverted Key File Method*, CMAR offers a

MICRO BUSINESS WORLD INC.

Information Line (213) 996-2252
TOLL FREE MAIL ORDER LINES
(800) 423-5886 Outside Calif

HEWLETT PACKARD — Special Of The Month — LIMITED QUANTITIES

HP 83 — OUR PRICE **$1399.00** — MSL 2250.00 — Save 951.00

HP DISC DRIVE SINGLE MASTER MODEL 82902M — OUR PRICE **$1099.00** — MSL 1500.00 — Save 401.00

COMPUTERS

apple computer

*Available in store only.

	List	Our Price
Apple II-16K	$1330.00	$ 999.00
Apple II-32K	1430.00	
Apple II-48K	1530.00	
Apple II-64K	1725.00	
Apple III 128-K	3915.00	2997.00
Apple II w/interface Dos. 3.3	645.00	499.00
Apple II Disk II Drive Only	525.00	429.00

HEWLETT PACKARD

	List	Our Price
HP-125 Microcomputer	3750.00	2990.00
HP-85 Microcomputer	3250.00	2475.00
HP-83 Microcomputer	2250.00	1777.00
16K Exp-Memory Module	295.00	255.00
Graphics Plotter 7225	2450.00	2075.00
Personality Mod. For 7225	750.00	665.00
2631B Impact/Printer/ Hvy Dty	3950.00	3250.00
Option 020 For 2631B	150.00	125.00
8 Drives to Choose from 82902S	1300.00	1125.00
9895A 8" Dual Drive	6850.00	5500.00
Graphics Tablet 9111A	2050.00	1678.00
HP-11C Slim-Line Advanced	135.00	
HP-12C Slim-Line Financial	150.00	
HP-41 CV New 2.2 Bytes Mem	325.00	250.00
HP-41-C Calculator	250.00	185.00
Card Reader For 41CV/C	215.00	162.00
Printer For 41CV/C	385.00	289.00
Optical Wand For 41 CV/C	125.00	97.00
Quad Ram Equals 4 Mem. Mods	95.00	81.00
Memory Modules For 41C		25.00
HP-97 Programble Printer	750.00	595.00
HP-67 Programble Calculator	375.00	295.00
HP-34C Programble Scientific	150.00	117.00
HP-38C Programble Bus. R/E	150.00	117.00
HP-32E Adv. Scientific	55.00	48.00
HP-37E Business Mgmt.	75.00	57.00

commodore

	List	Our Price
8032-32K 80 Col Crt	1495.00	1097.00
4032-32K 40 Col Crt	1295.00	989.00
4016-16K 40 Col Crt	995.00	788.00
8050-Dual Disk 950K	1795.00	1344.00
4040-Dual Disk 343K	1295.00	989.00
C2N-Cassette Drive	95.00	77.00
4022-80 Col Printer	795.00	639.00
8024-Mannsman Talley	1995.00	1545.00
25CPS-Starwriter	1895.00	1399.00
CBM-IEEE Modem	279.00	225.00
Voice Synthesizer	395.00	325.00
VIC 20	299.00	255.00
PET to IEEE Cable	39.95	34.00
IEEE to IEEE Cable	49.95	39.00

XEROX

	List	Our Pr
Xerox 820-1 5-1/4" Disk	2995.00	2399.00
Xerox 820-2 8" Disk	3795.00	2995.00
CP/M Operating System	200.00	169.00
Word Processing Software	500.00	429.00
Super Calc	295.00	249.00

ATARI PERSONAL COMPUTERS

	List	Our Price
400 16K Bytes of Ram	595.00	337.45
800 16K Bytes of Ram	1080.00	739.00
410 Program Recorder	90.00	77.00
810 Disk Drive	600.00	457.00
825 Printer (80 col Centronic 737)	999.95	769.00
820 Printer (40 col impact)	450.00	353.00
830 Acoustic Modem	200.00	155.00
850 Interface Module	220.00	192.00
Atari Visicalc	200.00	164.00
Atari 400-16K	595.00	327.00

TEXAS INSTRUMENTS

	List	Our Price
TI 99/4A Console New Improved	950.00	385.95
10" Color Monitor High Resolution	374.95	317.95
32K Memory Module	399.95	312.95
Speech Synthesizer	149.95	127.95
Disk Memory Drive	499.95	390.95
RF Modulator	49.95	42.50
Telephone Coupler (Modem)	224.95	185.95
Printer (Solid State)	399.95	315.95

ALTOS 8CS 8000-15	5990.00	4450.00
SHARP-64K Z80 FULL KBD YX-3200		CALL
NEC PC 8001	1295.00	1050.00

APPLE II STUDENT SYSTEM

- Apple II Plus System-48K RAM
- Disk II Floppy Disk & Interface (DOS 3.3)
- 12" Grn. Phs. Video Monitor

*Available in store only

SAVE $655 — Our Price $1795 — List $2450.00

PRINTERS

	List	Our Price
EPSON		
MX 80 FT	$ 745.00	$ 549.00
MX 80 IMPACT	645.00	450.00
MX 70 IMPACT	500.00	390.00
MX 100	995.00	765.00
APPLE SILENTYPE	645.00	299.00
ANADEX 9501	1650.00	1299.00
NEC		
5510	3195.00	2445.00
5515	3295.00	2545.00
3510	2495.00	1795.00
3515	2545.00	1849.00
OKIDATA		
MICROLINE 80	545.00	395.00
MICROLINE 82	649.00	549.00
MICROLINE 83	1050.00	769.00
PAPER TIGER		
445G with Graphics	795.00	695.00
460G with Graphics	1,394.00	899.00
560G New full size	1,695.00	1,139.00
DIABLO (LETTER QUALITY)		
630 R102 bi-directional tractors	2,965.00	2,350.00
1640K109 keyboard tractors	4,000.00	2,899.00
630 RO Receive Only	2,710.00	2,250.00
1650K 136 keyboard tractors	4,000.00	3,100.00

MONITORS

	List	Our Price
BMC Grn. Phs 12"	275.00	165.00
NEC Grn. Phs. 12"	225.00	179.00
Zenith 12"	159.00	119.00

LE MONITOR

	List	Our Price
* 9" B & W	179.00	159.00
* 9" Green Phs	189.00	165.00
* 12" B & W	209.00	185.00
* 12" Green Phs.	229.00	199.00
** 12" Green Phs.	199.00	159.00

* Anti Reflective Screen
** T.V. Grade Screen

SANYO MONITORS

	List	Our Price
13" Color (new) high quality	550.00	388.00
12" Green Phs.	360.00	240.00
12" B & W	340.00	217.00
15" B & W	370.00	235.00
9" B & W The Best Seller!	235.00	145.00

1 Year Extended WARRANTY — $99.00 — INQUIRE

MEMBER BBB

SAM CAN! XEROX 820

WORD PROCESSING SYSTEM (INCLUDES SOFTWARE)

MSL 3495.00 — OUR PRICE $2649.00 — Save 846.00

TERMINALS

TELEVIDEO

	List	Our Price
910	$ 699.00	$ 599.00
912C	950.00	699.00
920C	995.00	795.00
950C	1,195.00	949.00

DISKETTES SOLD IN BOXES OF 10

(Min. Purchase) $100

PRICE PER DISKETTE	List	Our Price
DYSAN		
104/1 5" SOFT SECTOR	6.00	3.99
104/1D = " DBL. DEN.-SOFT SEC.	6.40	4.60
3740/1 8" SOFT SECTOR	7.25	4.75
3740/1D 8" DBL. DEN.-SOFT SECTOR	10.75	6.90
MAXELL		
MD-1 5" SOFT SEC-TOR/DBL. DEN.	5.00	3.50
MD-2 " SOFT SECTOR/DBL. SIDE./DBL. DEN.	7.00	4.90
FD-1 8" SOFT SEC./DBL. DEN.	6.50	4.50
FD-1 8" SOFT SEC./DBL. SIDE/DBL. DEN.	8.50	5.95

DRIVES

CORVUS

	List	Our Price
5 MBYTES	3,750.00	$3,050.00
10 MBYTES	5,350.00	4,449.00
20 MBYTES	6,450.00	5,325.00

SOFTWARE FOR APPLE II

*Available in store only.

	List	Our Price
*Apple Fortran	200.00 $	147.00
DOS 3.3 (16 Sector)	60.00	45.00
*Apple PILOT (16 Sector)	150.00	119.00
Apple FORTRAN (requires A2B0006) (16 Sector)	200.00	159.00
Language System with Apple Pascal	495.00	399.00
BPI General Ledger System	395.00	319.00
BPI Inventory Package	395.00	319.00
Visidex	200.00	159.00
Visicalc	200.00	159.00
Desktop Plan II	200.00	159.00
Microlab Database System	150.00	119.00
Stoneware DB Master	229.00	189.00
Muse Supertext II	150.00	119.00
Softape Magic Window	99.00	72.00

commodore VIC-20

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

MSL 299.00 — OUR PRICE $255.00 — Save 44.00

SOFTWARE FOR COMMODORE

	List	Our Price
Ozz-The Information Wizard	395.00	$289.00
Wordcraft 80	395.00	289.00
Irma-Info Retrieval & Mgmt. Aid	495.00	389.00
Dow Jones Portfolio Mgmt.	149.00	119.00
Pascal Development Pkg.	295.00	219.00
Ebs-Receivables, Inventory	750.00	569.00
Bpi-General Ledger	395.00	389.00
Word Pro 3-40 Column	250.00	175.00
Word Pro 4-80 Column	375.00	259.00
Word Pro 4 Plus	450.00	319.00

MODEMS

	List	Our Price
NOVATION CAT MODEM	189.95	140.00
NOVATION D-CAT	199.00	150.00
NOVATION APPLE-CAT	349.00	310.00
HAYES MICROMODEM	379.00	295.00
HAYES SMARTMODEM	279.00	239.00

CALCULATORS

CASIO

	List	Our Price
HR-10 Paper Feed	49.95	39.00
HR-12 Paper Feed	54.95	42.00
FR-100 Paper Feed	79.95	59.00
FR-1210 Paper Feed	129.95	99.00
PQ-20	29.95	23.00
LC-785	12.95	10.00
LC-3165	12.95	10.00
FX-68 Scientific	29.95	23.00
FX-81 Scientific	19.95	17.00
FX-3600P Scientific	39.95	29.95
FX-602P "Computer Talk" 88 Memories Programming Upper & Lower Case Dot Matrix 512 Step	129.95	99.00
FX-702P Solves Problems with Alpha-Numeric Clarity, uses Basic Language	199.95	159.00

TELE. ANSW. DEVICES

PHONE MATE

	List	Our Pr
900	119.95	86.00
905 Remote	199.95	144.00
910	159.95	115.00
920	199.95	144.00
925 Remote	239.95	173.00
930 Remote	299.95	216.00
950 Remote	339.95	245.00
960 Remote	399.95	288.00

MICRO MAIL ORDER — BUSINESS WORLD WAREHOUSE

18720 Oxnard, #108 Tarzana, CA 91356

Telex: 182852
Answer: MICKO TZNA

OUTSIDE CA CALL TOLL FREE 1 (800) 423-5886 IN CA (213) 996-2252

Name (Please print) ______
Address ______
City ______ State ______ Zip ______

Qty	Make	Model	Description	Price	Total

TOTAL ORDER $ ______
TAX IF APPLICABLE* ______
SHIPPING & HANDLING** ______
TOTAL ENCLOSED $ ______

*California residents add 6% sales tax
**Add 3% Shipping & Handling — Add 3% surcharge for credit cards. Orders cannot be shipped unless accompanied by payment, including shipping, handling and tax where applicable.

☐ Certified Check or M.O. ☐ Allow 2 weeks clearance for personal check
☐ Bank Wire Transfer
☐ VISA ☐ MasterCard

CREDIT CARD # ______
Exp. Date ______ Signature ______

new level of efficient disc management for programs requiring sophisticated multi-key accessing. CMAR performs all read, write, change and delete functions and supports up to five opened keys concurrently. Future hard disc emulations with an unlimited number of secondary keys will provide a "big system" working environment at low cost.

With a random access speed gain of nearly 60% and a sequential speed gain of over 45%, even programs with heavy posting routines, sorts and fetches can become practical and manageable.

CMAR integrates its dynamic file substructure with the application program and Commodore Basic. Since the file system is dynamic, no file reorganization is ever required as in ISAM and Binary Tree configurations.

CMAR requires less than 10K bytes of memory and can be expanded to any operational file size, limited only by disc storage capacity. Files can consist of 2,000 to 20,000 keys depending on key size. Additionally, two kinds of key files are permitted; a primary key and its associated data record both based on relative file format.

CMAR is available through CIMARRON and is priced at $99.00 dealer net. A multiple quantity discount structure for application developers is also offered. Documentation only can be purchased separately for $10.00 on diskette in WordCraft 80 format.

OSI Screen Editor

Edit All is a full screen editor for OSI computers from DMP Systems, 319 Hampton Blvd., Rochester NY 14612. It replaces the standard I/O routines to allow the user to edit any program line that is on the screen. As editing takes place, the line is dynamically expanded or contracted.

Edit All supports a scroll window screen handler that allows you to define where on the screen you want your output to go. All output to the screen is via a window whose length, height and width are all user changeable. Full cursor control is supported along with an instant screen clear. Edit All works with OS65D Basic and Assembler. Price is $19.95 for 5¼ disk.

World Book Encyclopedia Now On CompuServe

Columbus, Ohio — The CompuServe Information Service today announced the signing of an agreement with World Book

consumer computers Mail Order DISCOUNTS

ORDER TOLL-FREE

NEC Microcomputer

PC-8001A 32K Computer	CALL
PC-8012A I/O Unit w/32K RAM	CALL
PC-8013A Dual Mini-Disk Drive Unit	CALL
PC-8001 Multi Cardware (FDI/O & 32K)	CALL
CP/M 2.2 Operating System for NEC	129
WordStar configured for NEC	299
SuperCalc configured for NEC	279
NEC Wordprocessor & Accounting Software	CALL

Many more software packages and languages; (Pascal, Fortran, Cobol, etc) are available configured for the NEC 8001A Computer.
Please call or write for a product price list.

ATARI 800 16K $749

Atari 400 w/16K	349
410 Program Recorder	65
810 Disk Drive	449
825 80 col. 7x8 Dot matrix impact printer	699
822 40 col. Quiet Thermal Printer	349
850 Interface Module	159
Atari 16K Ram Module	69
Axlon Ramcram 32K Module	189

Video Monitors

Amdek/Leedex Video 100 12" B&W	155
Amdek/Leedex Video 100G 12" Green Phospher	179
Amdek (Hitachi) 13" Color w/audio output	389
NEC 12" Green Phospher Display JB-1201M	CALL
NEC 12" Lo-Res Color Display	CALL
NEC 12" Hi-Res RGB Color Display	CALL
Sanyo 9" B&W Display	185
Sanyo 9" Green Phospher Display	CALL
Sanyo 12" B&W Display	269
Sanyo 12" Green Phospher Display	285
Sanyo 13" Color Display	449
Zenith 12" Green Phospher Display ZVM-121	149

ZENITH 12" GREEN $149

VIC20™ $259

Personal Computer
Color · Sound · Graphics
Call or write for more info.
Disk drives available soon!

apple computer

Authorized Dealer

APPLE II PLUS

16K *NOW* **$1025**
48K *NOW* **$1089**
64K* *NOW* **$1199**

*48K Apple with 16K RamBoard

APPLE DISK DRIVES $439

w/controller and DOS 3.3 **$499**

Apple Cards and Hardware

Language System w/Pascal & BASICS	379
Silentype Printer w/Interface card	349
Hayes Micromodem II	299
Novation Apple-Cat	339
Videx Videoterm 80 column card	269
Videx Keyboard Enhancer	115
Z-80 Softcard by Microsoft	299
16 K RamCard by Microsoft	169
CPS Multi-function card	189

Software for the Apple

VisiCalc version 3.3	159
VisiFile (NEW data base manager)	199
VisiTrend/VisiPlot	219
DB Master	169
WordStar (Apple 80 col. version)	249
Dow Jones Portfolio Evaluator	45
Apple Post	45
Apple Writer	65
Dow Jones News & Quotes Reporter	85
Apple Plot	60
Tax Preparer	99
Real Estate Analyzer	129

16K RAMBOARD *by ConComp* for Apple II Computers

FOR ONLY **$129.95**

AVAILABLE NOW

S-100

California Computer Systems

Floppy Disk Controller	**$369**
64K Dynamic Ram Board, 200ns	**$499**
Z-80 CPU board w/monitor ROM	**$269**
16K Static memory board, 200ns	369
32K Static memory board, 200ns	599
S-100 12 Slot Mainframe	475
4-Port Serial Interface	299
2-Port Serial/2-Port Parallel Interface	299
4-Port Parallel Interface	229

Printers

Silentype w/Apple II interface **$349**

Epson MX-80 or MX-80 FT CALL

Anadex 9501 w/2K Buffer	1349
C. Itoh Starwriter 25 CPS daisywheel	1449
C. Itoh Starwriter 45 CPS daisywheel	1649
Epson MX-70	CALL
Epson MX-80 & MX-80 F/T	CALL
Epson MX-100	CALL
NEC 8023 Impact Dot Matrix	695
NEC Spinwriters (Latest models)	CALL
Paper Tiger IDS-445G w/graphics	699
Paper Tiger IDS-460G w/graphics	949
Paper Tiger IDS-560G w/graphics	1249
Silentype Printer w/Apple interface	349
Qume Sprint Daisywheels (Latest models)	CALL

ORDER TOLL FREE

800-854-6654

In California and outside continental U.S.

(714) 698-8088

Telex 695-000 Beta CCMO

Send Orders To:

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.

consumer computers Mail Order

8314 Parkway Drive
La Mesa, Calif. 92041

Encyclopedia to cooperate in developing the on-line, electronic version of the encyclopedia.

Robert King, chairman of the board of directors at World Book, a subsidiary of Scott & Fetzer Company, said, "We expect to develop significant new services utilizing our existing database, both for the home and business markets. This service will add a new dimension to the printed edition of World Book."

CompuServe subscribers can access the World Book service for the standard fee of $5 per hour weekday evenings, all day weekends and holidays. Weekday daytime access is also available. To use the service, a subscriber needs a personal computer or terminal, a telephone and a modem.

CompuServe also offers the latest news from major newspapers and a national wire service, corporate stock and commodities information, home banking, electronic mail and real-time communications, computer games, family information and computing power for programming activities.

CompuServe is available through a local telephone call in more than 260 US cities. CompuServe is an H&R Block Company.

K-BYTE™ Introduces K-RAZY SHOOT-OUT™ Between Space Commanders And Alien Droids

It's a battle of wits and fast action, this game in ROM form for Atari® 400™ or Atari® 800™ Personal Computer Systems. K-BYTE's K-RAZY SHOOT-OUT is available in a solid-state cartridge (ROM pack), individually tested to assure reliability and customer satisfaction.

K-BYTE's K-RAZY SHOOT-OUT is destined to be a favorite of Atari enthusiasts across the country.

For details on K-RAZY SHOOT-OUT™, contact Chuck Miller at K-BYTE™, Division of Kay Enterprises Co., 1705 Austin, Troy, Michigan 48099. Phone: (313)524-9878.

The Stock Market Goes Home

Marketscan, a new personal online stock market service was formally announced today by Barbara Hyland, Manager of Info Globe.

"We're introducing Marketscan to meet the demands for online information by personal computer users," says Hyland. With Marketscan a user can get daily quotations for all stocks listed on the Toronto, Montreal, Alberta, Vancouver, New York and American Stock Exchanges. The high, low, close, and trading volume are displayed in familiar formats. Marketscan provides a variety of services. A user can analyze a specific stock on one or several exchanges or several stocks at one time, monitor an