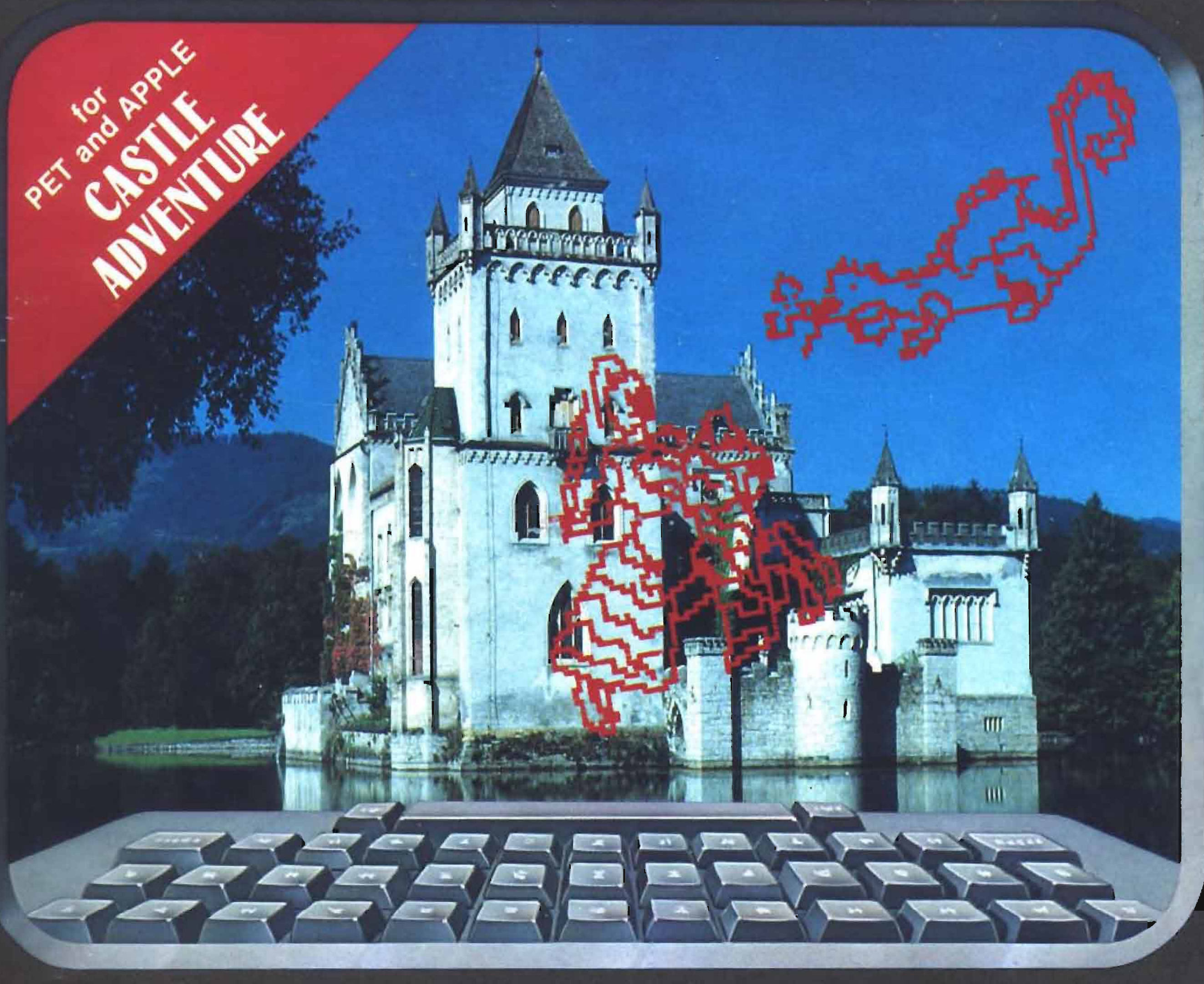
NO. 54 NOVEMBER 1982

MICRO

THE 6502/6809 JOURNAL



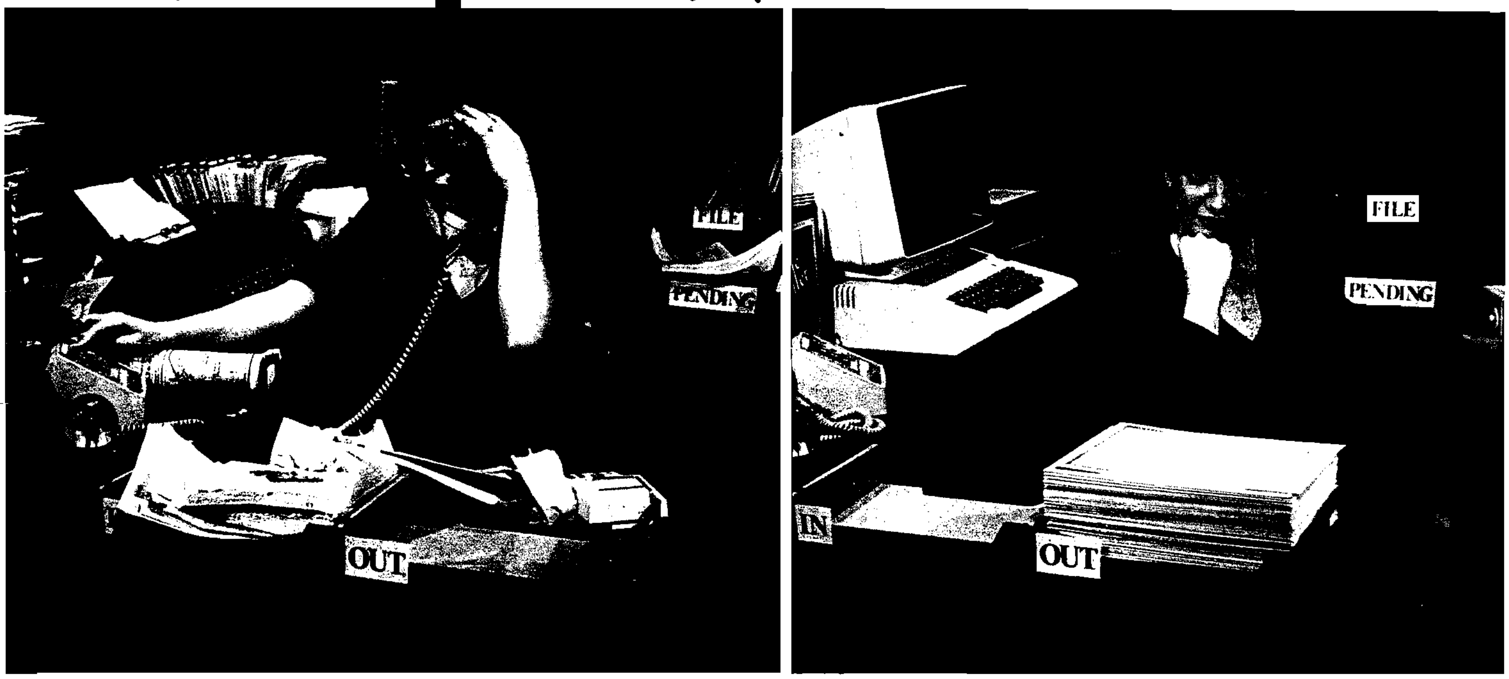
Games Feature

Atari Character Graphics

Hi-Res Graphics and Memory Use on the APPLE

PET Graphic-80 Conversion

Magic Window II



Makes Paperwork Pandemonium Vanish

MAGIC WINDOW II turns your APPLE into a sophisticated word processor. But because MAGIC WINDOW II operates so much like a standard typewriter, it's extremely simple to use. In fact, because of its unique menu structure, it's the easiest to learn, and function selection is virtually error free.

MAGIC WINDOW II's powerful word processing features include automatic formatting, editing, centering, and justification — and these are all done easily "on the video screen" before you ever print. Just type your first draft quickly, then go back and make any needed corrections. You can insert or delete letters or words, even move whole paragraphs with just a few simple keystrokes.

And MAGIC WINDOW II can support 40-, 70-, and 80-column displays, as well as automatically providing 80-column visibility with scrolling. The rule is: What you see on the screen is what you get in print. No word processor on the market has both the features of MAGIC WINDOW II and its simplicity of use.

And as an extra assurance that your document is perfect before printing and mailing, you can use MAGIC WORDS. With incredible speed, MAGIC WORDS proofreads your document for spelling errors and typos, shows you each one in context on the screen, and allows you to correct or ignore each in sequence. Unlike any other spelling checker, it will then automatically create a corrected file as you go so you never need to return to MAGIC WINDOW II to update it yourself manually. Or, if you're busy, MAGIC WORDS will go through your file without waiting for you to act on each error, and provide you with a printout of the errors and their locations so you can correct them at your convenience.

And with a 14,000-word basic dictionary and plenty of memory for you to add technical terms or customer names that you use frequently, MAGIC WORDS becomes a totally personalized program that can catch all your spelling errors and typos.

But now comes the mailing, the really time-consuming part — unless you have MAGIC MAILER. A mailing list merge system, MAGIC MAILER lets you insert each name and address (or whatever is in your records) into your document quickly and efficiently.

With MAGIC MAILER, you never have to retype a document or an internal address. With just a few keystrokes, each letter becomes an original, and the final phase of the paperwork process is complete — efficiently and to perfection.

Let's face it: The letters, invoices, and other documents you send out represent you to the public. They should be as professional as possible — and they can be. All it takes is a little MAGIC.

Each program is available separately or ask your local software store for Artsci's MAGIC PAK. All three products are included in an attractive library box.

213 · 985-2922

artsci

MICRO™

November Highlights

Games Feature

This month we expand a bit from our usual content and offer an array of games for a wide variety of computers. Although we don't usually publish games, we feel that they may be valuable, particularly where they demonstrate techniques or cultivate a skill in the user. Instead of making games a regular part of MICRO, we prefer to do it all at once!

There are specific games for nearly every computer, including the PET, VIC, Apple, C1P/Superboard, Atari 400/800, and the SYM. In addition, you will find that most can be converted easily to run on even more machines. These represent a variety of types of games too — from action games like "Space Invasion" and "Shootdown," to strategy games like "Number Shuffle," "GOMOKU," and "23 Matches," to "Castle Adventure."

Written in the style of Scott Adams' famous games, "Castle Adventure" (p. 41) tests your memory and analytical skills. You will find yourself in the evil baron's castle in a quest for treasure and the kidnapped princess. "Castle Adventure" was originally written for the PET. We have provided specific line changes to make it run on the Apple. However, except for disk commands, it is written in straight Microsoft BASIC, so owners of OSI, Atari (with Microsoft BASIC), and Color Computer (with extended BASIC) machines should be able to easily adapt "Castle Adventure."

"Solve the Pagoda Puzzle Using Recursive Assembly" (p. 53) is particularly interesting because it demonstrates a technique — having a subroutine call itself — that you may want to apply to your own programs. The author's application is in solving the "Pagoda" or "Tower of Hanoi" puzzle. The program solves the puzzle for a stack of disks of any practical height and outputs a list of specific moves. With a minimum of changes, it will run on any 6502 computer. You may want to try incorporating this routine into a BASIC program that actually shows the disks being moved from peg to peg.

"GOMOKU" (p. 59) is a fast, machine-language version of the oriental game of strategy. It is presented here for VIC, with modifications for PET. "Number Shuffle" is an Atari computer version of "Magic Square," the game where you slide the little numbered squares around until you get them in order. "Space Invasion," for the C1P/Superboard and "Shootdown" are arcade-style action games. Finally, "23 Matches" is a short machine-language game for the SYM. It makes ingenious use of the SYM's LCD display.

This month's editorial (p. 7) offers some thoughts on games and their place in MICRO and in our society.

68000 Coverage Continues

Preliminary results from our survey indicate that a lot of you are interested in the 68000. As part of our continuing effort to keep you informed on this powerful new processor, we present two articles this month on the 68000. Dr. Hootman's detailed discussion of the 68000 instruction set continues (p. 27) with the binary arithmetic operations. Handy reference tables are included. Jelemensky and Whiteside (p. 13) conclude their demonstration of 68000 programming techniques.

We haven't seen any games yet for the 68000. Is it because its users haven't gotten over the speed and power? Or is it because these machines aren't finding their way into very many homes?

Atari Coverage Takes Off

With the addition of Contributing Editor Paul Swanson to our staff, MICRO's Atari coverage has improved considerably. Paul's column, a new "From Here to Atari" (p. 103), starts this month. In addition, he continues his character graphics article series with a discussion of fine scrolling (p. 82). This month's data sheet (p. 109 — compiled by Paul Swanson) is a handy reference for serious Atari programmers.

Atari users will be interested in programming extra colors, even in the limited high-resolution modes. Richard and Donna Marmon (p. 96) illustrate two techniques — one that uses adjacent color dots, and one that quickly alternates displays.

Hardware

In keeping with our lighter theme in this issue, the hardware articles we present are simple, single-evening projects. All involve modifications of existing equipment. Ralph Tenny (p. 19) shows how to get a high-quality picture from your color computer using a monitor instead of a TV. A monitor requires a composite video signal, not available on the CC, and the author shows you how to add it. Jim Strasma (p. 35) shows how to take Commodore's cheapest model PET and convert it into a machine with 80-column business capabilities, yet with all the graphic characters still available from the keyboard. OSI owners can now use Atari's inexpensive joysticks with their machines. Joseph Ennis (p. 9) shows how to make the simple changes in your computer board.

Graphics for the Apple

For those interested in improving their game and graphic programming skills, our graphics articles for Atari and Apple will help. Apple programmers will learn about 3-D rotation from Chris Williams (p. 99). If you have done much graphics programming on the Apple, you have probably been annoyed by the unfortunate location of the graphics pages. Authors Berns (p. 93) and Weston (p. 79) present a number of techniques to circumvent this problem.

New Color Computer Column Expands 6809 Coverage

John Steiner's new monthly column "CoCo Bits" covers the 6809-based TRS-80 Color Computer (p. 38). This month he discusses some problems associated with transferring cassette programs to disk and presents a short program to move the game "BEDLAM." Also for the Color Computer owner, Ron Anderson discusses FLEX09 (p. 23) as it is implemented by Frank Hogg Laboratories. FLEX is a universal operating system that opens up a wide range of software for the 6800 and 6809 to the Color Computer owner.

MICRO

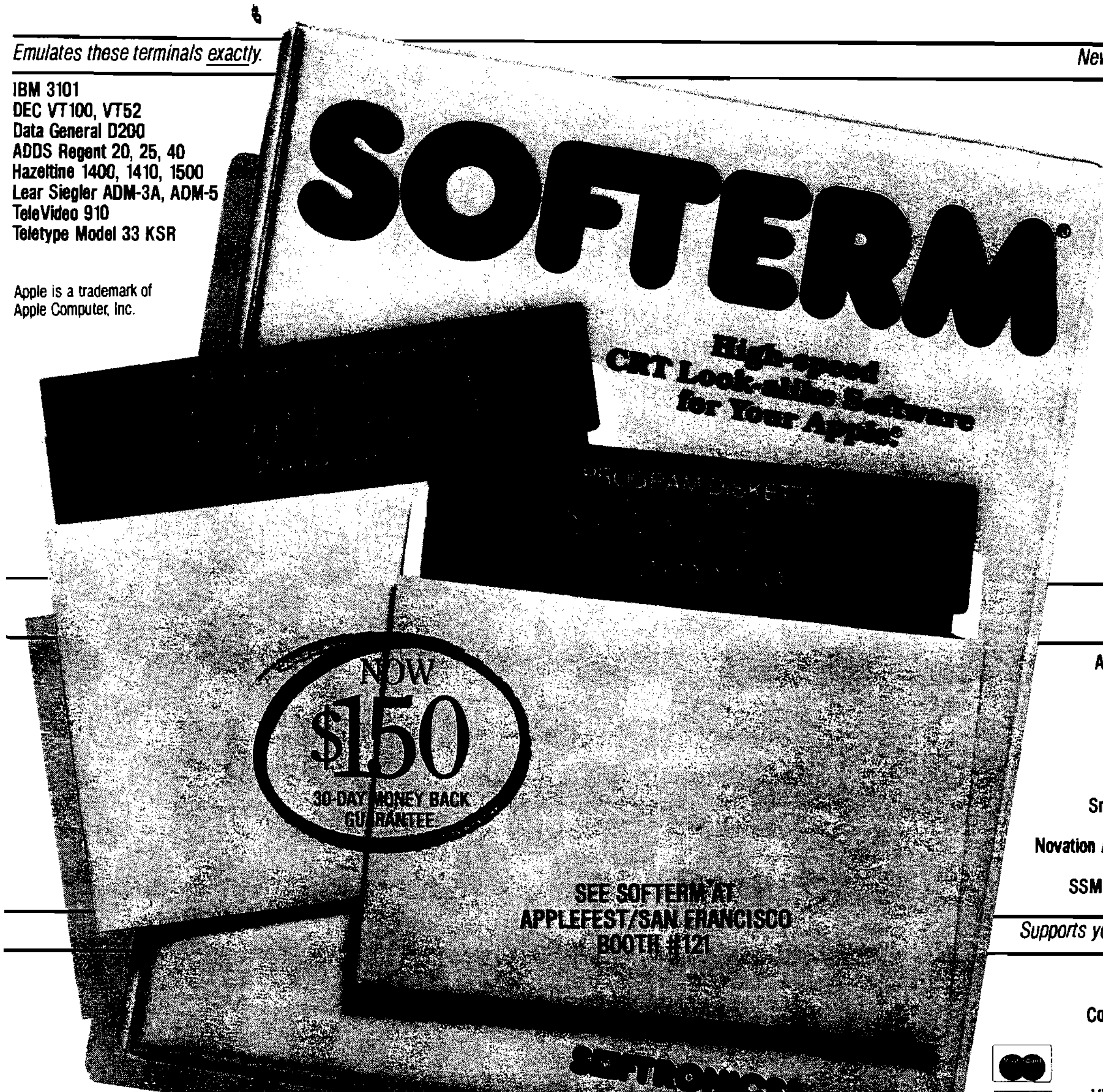
Emulates these terminals exactly.

IBM 3101
DEC VT100, VT52
Data General D200
ADDS Regent 20, 25, 40
Hazeltine 1400, 1410, 1500
Lear Siegler ADM-3A, ADM-5
TeleVideo 910
Teletype Model 33 KSR

Apple is a trademark of Apple Computer, Inc.

New File Transfer Language

BREAK
CATALOG
CHAIN
CONFIGURE
CONNECT
CONVERSE
DIAL
END
HANGUP
LOG
MONITOR
NOLOG
ONERR
PAUSE
PROMPT
RECEIVE
REMARK
RETRIES
SEND
SPECIAL
SPEED
TIMEOUT
XMIT:WAIT



Supports these interface boards.

Apple Communications Card
Apple Parallel Printer
Apple Serial Interface
Apple Super Serial Card
Bit 3 Dual-Comm Plus™
CCS 7710, 7720, 7728
Hayes Micromodem II™, Smartmodem™ 300, & 1200
Intra Computer PSIO
Novation Apple-Cat II™ 300 & 1200
Orange Micro Grappler™
SSM ASIO, APIO, AIO, AIO II™

Supports your 80-column hardware.

ALS Smarterm™
Bit 3 Full-view 80™
Computer Stop Omnivision™
M&R Sup'R'Terminal™
STB Systems STB-80™
Videx Videoterm™
Vista Computer Vision 80™
Wesper Micro Wizard 80™

Your host computer won't know the difference!

Softerm provides an *exact* terminal emulation for a wide range of CRT terminals which interface to a variety of host computer systems. Special function keys, sophisticated editing features, even local printer capabilities of the terminals emulated by Softerm are fully supported. Softerm operates with even the most discriminating host computer applications including video editors. And at speeds up to 9600 baud using either a direct connection or any standard communications modem.

Unmatched file transfer capability

Softerm offers file transfer methods flexible enough to match any host computer requirement. These include *character protocol* with user-definable terminator and acknowledge strings, block size, and character echo wait, and the intelligent *Softrans*™ protocol which provides reliable error-free transmission and reception of data. The character protocol provides maximum flexibility for text file transfers. *Any* type file may be transferred using the Softrans protocol which provides automatic binary encoding and decoding, block checking with error recovery, and data compression to enhance line utilization. A FORTRAN 77 source program is supplied with Softerm which is easily adaptable to any host computer to allow communication with Softerm using the Softrans protocol.

Softerm file transfer utilizes an easy to use *command language* which allows simple definition of even complex multiple-file transfers with handshaking. Twenty-three high-level commands include *DIAL, CATALOG, SEND, RECEIVE, ONERR, HANGUP, MONITOR* and others which may be executed in immediate command mode interactively or from a file transfer macro command file which has been previously entered and saved on disk.

Built-in utilities

Softerm disk utilities allow DOS commands such as CATALOG, INIT, RENAME, and DELETE to be executed allowing convenient file maintenance. Local file transfers allow files to be displayed, printed, or even copied to another file without exiting the Softerm program. Numerous editing options such as tab expansion and space compression are provided to allow easy reformatting of data to accommodate the variations in data formats used by host computers. Softerm supports automatic dialing in both terminal and file transfer modes. Dial utilities allow a *phone book* of frequently used numbers to be defined which are accessed by a user-assignable name and specify the serial interface parameters to be used.

Online Update Service

The Softronics Online Update Service is provided as an additional support service at no additional cost to Softerm users. Its purpose is to allow fast turnaround of Softerm program fixes for user-reported problems using the *automatic patch facility* included in Softerm as well as a convenient distribution method for additional terminal emulations and I/O drivers which become available. *User correspondence* can be electronically mailed to Softronics, and *user-contributed* keyboard macros, file transfer macros, and host adaptations of the Softrans FORTRAN 77 program are available on-line.

Most advanced communications software available

Just check Softerm's 300 page user manual. You simply can't buy a more sophisticated package or one that's easier to use. Available now for only $150 from your local dealer or Softronics, Inc.

SOFTRONICS

6626 Prince Edward, Memphis, TN 38119, 901-755-5006

NO. 54 NOVEMBER 1982

MICRO™

THE **6502/6809** JOURNAL

Advertising Sales Representatives
See Page 127

Subscription/Dealer inquiries
(617) 256-5515

DEPARTMENTS

GAMES FEATURE

HARDWARE

GRAPHICS

About the Cover

Our brave knight, Godfrey de Goodheart, boldly chases dragons through Baron Von Evil's castle in search of the fair Princess Fatima. MICRO features "Castle Adventure" by David Malmberg [page 41]. It is written for PET, Apple, and other Microsoft BASIC computers.

The photo, by Kenneth Witham, is of Schloss Anif in Salzburg, Austria. The knight and dragon graphics were drawn on the Apple Graphics Tablet.

MICRO is published monthly by:
MICRO INK, Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to
MICRO INK
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

MICRO™

Editorial

Responsible Gamesmanship

MICRO does not publish games. We've run editorials explaining why — outlining the weaknesses, drawbacks, and worthlessness of many computer games. The computer was not developed to fill arcades or to force squeals of delight or anguish from mesmerized users who've spent hours killing the same aliens over and over again.

So why have we not only added games in this issue, but FEATURED them? We aren't giving in; we still believe many games are a waste of time. But we also believe that games — when written and presented properly — can educate. In fact, they can act as an effective tool at all educational levels.

For example, there are games that simulate business environments, games that demand logical thought, games that teach us how to program, how to spell, or to calculate mathematical equations. The variations of these games that are most successful actively involve the student/participant in problem solving and decision making. They are not just drills to enable us to push the right button at the right time or to give the right answer; they are lessons in learning — they can expand our understanding of both artificial and human intelligence.

You see, it's the games that just pit one person against the computer in a mindless battle of eye-hand coordination that irritate us the most. [Does an image of your neighborhood's favorite arcade leap to mind?] Maybe these florescent, noisy battlegrounds provide entertainment for those who need to let off a little steam; but to have energetic, lively, questioning children and adolescents glued to machines in meaningless combat for hours on end is scary.

Whose responsibility is it [yours, ours, the schools, the manufacturers] to offer at least enough of the really worthwhile stuff to balance off what's already so, unfortunately, popular?

Judah Schwartz, Professor of Engineering Science and Education at MIT, summed up the software situation in a recent issue of *Classroom Computer News*. Although his comments were directed specifically toward educational material, they can be as easily applied to games in general: "My hope is that the publishers of this country — who control the curriculum far more than they even begin to realize — will stop doing what they are now doing and start to provide materials for computers which are more open-ended, which are more tool-like in nature, which will help children to assume a more active role, which will not trivialize the nature of education, and which will work to make schools more nearly the collaborative community of learners that they should be."

Computers are efficient, friendly, and generally expensive. As with everything else, we want them to be used to their full potential. One way is through well-written, mind-boggling, educational games. So, MICRO would like to promote the use of these types of games. We encourage manufacturers to continue to produce quality products that get the most out of the computer and the participant. We encourage publishers of books and magazines to support the use of stimulating games that require both the use of skill and the growth of skills.

We hope you enjoy the games we present in this issue, but also hope you will learn some new techniques and some good methods for writing your own games. We hope you will give some thought to the social impact of computer games, as well.

Micro Staff

&-Amper-Magic™

MACHINE LANGUAGE SPEED WHERE IT COUNTS...
IN YOUR PROGRAM!

For the first time, Amper-Magic makes it easy for people who don't know machine language to use its power! Now you can attach slick, finished machine language routines to your Applesoft programs in seconds! And interface them by name, not by address!

You simply give each routine a name of your choice, perform the append procedure once at about 15 seconds per routine, and the machine language becomes a permanent part of your BASIC program. (Of course, you can remove it if you want to.)

Up to 255 relocatable machine language routines can be attached to a BASIC program and then called by name. We supply some 20 routines on this disk. More can be entered from magazines. And more library disks are in the works.

These routines and more can be attached and accessed easily. For example, to allow the typing of commas and colons in a response (not normally allowed in Applesoft), you just attach the Input Anything routine and put this line in your program:

```
xxx   PRINT "PLEASE ENTER THE DATE."; : & INPUT,DATE$
```

Some routines on this disk are:

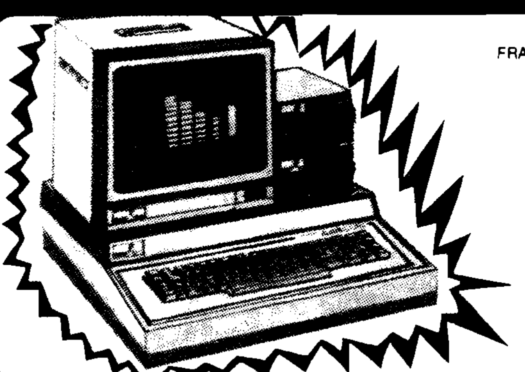
- Binary file info
- Delete array
- Disassemble memory
- Dump variables
- Find substring
- Get 2-byte values
- Gosub to variable
- Goto to variable
- Hex memory dump
- Input anything
- Move memory
- Multiple poke decimal
- Multiple poke hex
- Print w/o word break
- Restore special data
- Speed up Applesoft
- Speed restore
- Store 2-byte values
- Swap variables

&-MAGIC makes it Easy to be Fast & Flexible!

PRICE: $75

Anthro - Digital Software
P.O. Box 1385
Pittsfield, MA 01202
The People - Computers Connection

&-Magic and Amper-Magic are trademarks of Anthro-Digital, Inc.
Applesoft is a trademark of Apple Computer, Inc.



FRANKLIN

ACE 100

ENTIRE SYSTEM **1995.00**

ACE 100 64K   2 RANA DRIVES
12" NEC GREEN SCREEN MONITOR

VOICE SYNTHESIZER FOR AIM/SYM/KIM WITH SPEAKER $139.00
AIM 65, 4K, COMPLETE WITH MANUALS $469.00

NEW FULLY GUARANTEED PARTS

6502P, 1 MHZ	$ 5.95			555 TIMER	$ .39
6512P, 1 MHZ	5.95	2114, 1Kx4, 45ONS	$ 2.25	XR2206	4.95
6520P, PIA	4.45	6116, 2Kx8, CMOS	7.50	XR2211	4.95
6522P, VIA	6.45	4116, 16Kx1, DYNAMIC	1.95	DM8131	3.45
6532P, RIOT	7.95	4164, 64Kx1, DYNAMIC	10.95	RED L.E.D., TI ¾, HI EFF	.15
6545-1P, CRT	19.95	2716,2Kx8	3.95	IN4004	.09
6551P, UART	8.95	2532, 4Kx8	8.50	IN914	.04
6592P, PRINT CTRL	26.80	4N33, OPTO-COUPLER	2.95	IN4148	.04
6847P, VDG	15.60	VOLTRAX SC-01A	69.00	8 POSITION DIP SWITCH	4.95

PHONE ORDERS WELCOME – FREE CATALOG – HOURS 9 A.M. TO 5 P.M. CST.

CASHIERS CHECK or MONEY ORDER
PERSONNEL CHECKS, 2 WEEKS WITH D.L.#
MC or VISA, MUST HAVE CARD NO.
SHIPPED VIA U.P.S. or 1st CLASS MAIL
C.O.D. ACCEPTED, ADD $5.00
MAIL ORDER ONLY

ADD $1.75 FOR U.P.S.
ADD $2.75 for 1st CLASS MAIL
ADD ADDITIONAL $5.00 FOR C.O.D.
ADD ADDITIONAL $3.75 FOR OUT OF U.S.
TEXAS RESIDENTS ADD 5% SALES TAX
PRICES GOOD THROUGH DECEMBER 31, 1982

BEDFORD MICRO SYSTEMS   817 - 283-0013
P.O. BOX 1182   BEDFORD, TEXAS 76021

Installing Atari Joysticks on the OSI

by Joseph Ennis

A simple, non-destructive modification to the OSI C1P or Superboard is described. This allows the use of inexpensive Atari joysticks. Demonstration software and programming information are also included.

Joystick
requires:
C1P
Hardware components
One or two Atari joysticks

Installing Atari joysticks on the OSI 600 board-based computers (Superboard or C1P) is easy. It takes $9.95 per joystick and five minutes.

Sears is a good source for Atari joysticks; they stock them under two catalog numbers — 6K99835 for a single joystick or T3K7687 for a pair. You'll pay $19.95 for two. In addition you will need one 12-pin male Molex connector. You may purchase one from your OSI dealer for $1.00, or from Technical Products Co., Box 12983, University Station, Gainsville, FL 32604 (Molex connectors are $4.95 for four pair of male and female).

My 600 board came without J4 mounted, but it took me about one minute to push a connector into the holes in the printed wiring board and solder the twelve pins to the board.

When you get the joysticks, cut the Atari cables as close to the connector as possible, and strip enough of each wire to connect to a male Molex connector (about 1/8 inch). I used a drill press vise to hold the Molex connector and an X2 magnifying glass to aid in soldering. I also used the trick of wrapping several turns of number 18 buss wire around the tip of my 15-watt pencil iron to give me a fine soldering point.

Solder the wires of both joysticks (if you plan to use two) to the single male Molex connector according to the following table:

Molex Pin	Joystick 1	Joystick 2
1	—	Black
2	Black	—
3	—	—
4	—	—
5	Green	Green
6	Blue	Blue
7	Brown	Brown
8	White	White
9	Orange	Orange
10	—	—
11	—	—
12	—	—

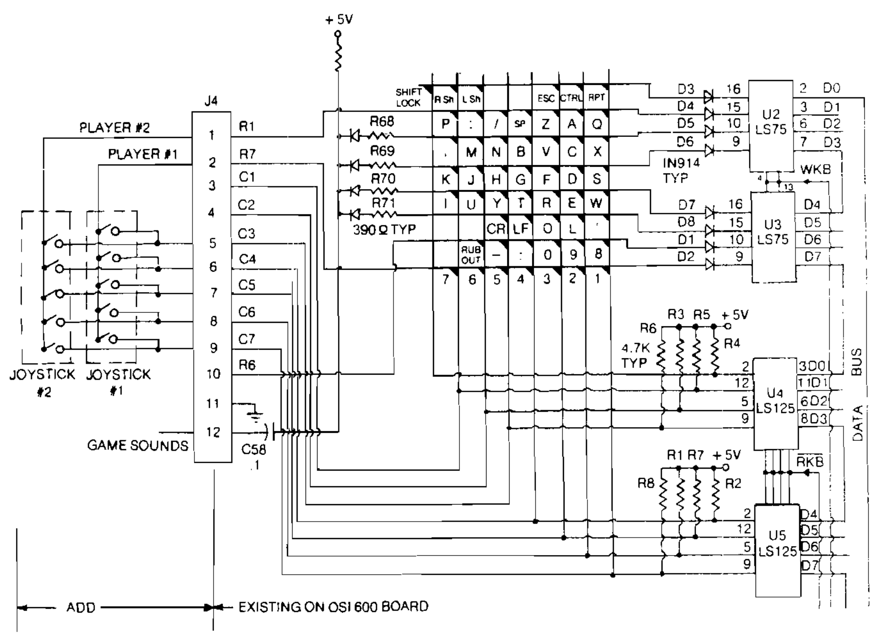
For those interested in how this modification works, figure 1 shows the schematic of the 600 board keyboard area with the joysticks connected to connector J4.

Testing is also easy. First do the usual inspection for poor workmanship, solder bridges, etc. Then plug the joystick connector into J4, the connector in the lower left corner or closest to the keyboard (see photograph). Plug in the joystick connector so that pin 1, the one with a black wire on it, is toward the back of the 600 board (away from the keyboard). Bring up power and, without touching the joystick, check to see that the keyboard works as before. Now pick up joystick 1, the one with its black wire connected to pin 2. Move the controls on the joystick and note that characters are printed to the monitor screen according to the following table. Pick up joystick 2 and perform the same test. Any problem is most likely a soldering problem.

	Joystick 1	Joystick 2
Fire	Q	1
Up	A	2
Right	Z	3
Down	Space	4
Left	/	5

Don't worry about Up-Right or Up-

***Figure 1:* Keyboard/Joystick Schematic**



Right-While-Firing. These combinations are there, but right now they decode to non-printable symbols and don't show on the monitor screen.

This completes installation and testing. The rest of the joystick operation is software. At the time of this writing OSI had not yet come out with any joystick programs for the 600 board computers, so there had not been any standardization of the joystick move/decode tables. Much joystick software for the OSI is sold by Aardvark Technical Services, 2352 South Commerce, Walled Lake, MI 48088. Aardvark uses a slightly different method of connecting joysticks and a different decode. There are advantages and disadvantages. Aardvark's Mod is longer and requires cutting some traces on the 600 board. However, they have a lot of good software already developed according to their convention. Fortunately, the difference is not great. Joystick 2 in this mod has the same decode as joystick 1 in the Aardvark mod. Therefore, all one-player games need no changes. Aardvark's joystick 2 is connected so that it decodes to: Fire = 8 Up = 9 Right = 0 Down = : and Left = -. For two-player Aardvark games the code must be changed.

When writing software for joysticks, the following line must appear early in the program:

POKE 530,1 :REM TURNS OFF KEYBOARD SCAN, POKE 530,0 WILL TURN KEYBOARD SCAN ON AGAIN

Later in the program, when joystick 1 is to be polled, program:

POKE 57088,128: P = PEEK(57088)

and when joystick 2 is to be polled, program:

POKE 57088,2: P = PEEK(57088)

Table 1 gives all the possible values for variable P.

A joystick demo program has been included at the end of this article. This demo illustrates most of the techniques for animated graphics and their use with joysticks.

This program, when running, will display two tank symbols. Each tank symbol will be controlled by one joystick. A study of listing 1 will illustrate the programming techniques required by programs using joystick inputs. It is not necessary to have a joystick decode table with all seventeen of the entries given in table 1, since FIRING is merely the position values, less 127. Therefore, note that in line 70 FIRE is set equal to 128 and in line 1000 P is checked to see that it is less than FIRE. If it is, then the program jumps to the FIRE subroutine at 2000. The last line in the FIRE subroutine adds 127 back to P, taking out the effect of FIRE before turning control over to the MOVE loop. This halves the time of the loop and the size of the joystick decode table, as only eight values are needed for two joysticks. This still allows the players to move while firing.

There is one disadvantage with the technique used in this joystick mod. When two players are playing and both players are moving their joysticks at the same time, there are combinations where there can be feedback through the joystick switches. With the keyboard polling routine that OSI uses, one data line at a time is set on the LS75 latches, U2 and U3 (see figure 1). When one latch (these are inverting latches) is set it pulls down the line to one of the rows of keys on the keyboard. If any key in that row is pushed, its position will be read by the LS125 bus drivers. This is why a PEEK to the keyboard address will return the value of 255 (all ones) when no keys are pushed. When only one of the LS75 latch stages is energized, then only one row is set for decode.

Pushing a key in any other row will not produce any output on the data bus. This is why selecting one joystick to row 1 and the other to row 7 will allow one joystick to be decoded independently of the other, even though they are connected to the same columns.

Listing 1

```
Ø REM*****************
1 REM  JOYSTICK     **
2 REM  DOODLER      **
3 REM               **
4 REM     by        **
5 REM Joseph Ennis  **
6 REM               **
7 REM A DEMO of JOY-**
8 REMSTICK TECHNIQUES*
9 REM*****************
1Ø REM JOYSTICK ONE IS SET AT 1=FIRE, 2=UP, 3=DOWN, 4=LEFT
11 REM JOYSTICK TWO IS SET AT Q=FIRE, A=UP, Z=RIGHT, SPACE=DOWN, /=LEF
12 REM YOU WILL NOTE SOME INTERACTION PROBLEMS WITH BOTH FIRE BOTTONS
13 REM SET TO COLUMN 7  THIS CAN BE FIXED BY SETTING FIRE ON
14 REM JOYSTICK TWO TO FIRE=; AND MAKING A FEW CHANGES IN FIRE SUBROUT
15 DIM K(8), M(8), S(8)
2Ø X=Ø: U=Ø
3Ø FOR X=1 TO 8: READ K(X): NEXT: REM LOADS KEY DECODE TABLE
4Ø FOR X=1 TO 8: READ M(X): NEXT: REM LOADS MOVE TABLE
5Ø FOR X=1 TO 8: READ S(X): NEXT: REM LOADS SYMBOL TABLE
6Ø A=5348Ø: B=54Ø61: REM TANK A&B START LOCATIONS
7Ø AA=2: BB=128: NOOP=254: FIRE=128: C=57Ø88: SHELL=46: BLANK=32
71 REM FIRE REALLY EQUALS 127 BUT SETTING TO 128 SAVES AT 1ØØØ & 211Ø
74 FOR X=1 TO 32: PRINT: NEXT
75 INPUT"SELECT SPEED (1=FAST 2ØØ=SLOW)";DELAY
9Ø FOR X=1 TO 32: PRINT: NEXT: REM SLOW SCREEN CLEAR
1ØØ POKE 53Ø,1: REM TURN OFF AUTOMATIC KEYBOARD SCAN
11Ø POKE A,S(4): POKE B,S(8) : REM INITIALIZE TANK LOCATIONS
12Ø POKE C,AA: P=PEEK(C): IF P<NOOP THEN F=2: GOTO 1ØØØ
13Ø POKE C,BB: P=PEEK(C): IF P<NOOP THEN F=3: GOTO 1ØØØ
14Ø GOTO 12Ø: REM LOOP WAITS FOR JOYSTICK MOVEMENT
2ØØ DATA 19Ø, 158, 222, 2Ø6, 238, 23Ø, 246, 182
21Ø DATA -32, -31, +Ø1, +33, +32, +31, -Ø1, -33
22Ø DATA 248, 249, 25Ø, 251, 252, 253, 254, 255
99Ø REM MOVEMENT SUBROUTINE STARTS ON 1ØØØ
1ØØØ IF P<FIRE THEN GOSUB 2ØØØ
1ØØ5 IF P=NOOP THEN GOTO1Ø5Ø
1Ø1Ø FOR X=1 TO 8
1Ø2Ø IF K(X)=P THEN MOVE=M(X): SYMBOL=S(X): X=8: REM SETS UP FOR MOVE
1Ø25 NEXT X
1Ø3Ø IF F=2 THEN POKE A,BLANK: A=A+MOVE: POKEA,SYMBOL: REM CLASSIC MOV
1Ø4Ø IF F=3 THEN POKE B,BLANK: B=B+MOVE: POKE B,SYMBOL: REM MOVE B
1Ø45 FOR X=1 TO DELAY: Z=2: NEXT X
1Ø5Ø ON F GOTO 1ØØØ,13Ø,12Ø: REM LOOPS BACK TO JOYSTICK DET LINES
199Ø REM FIRE DECODE SUBROUTINE STARTS ON 2ØØØ
2ØØØ REM FIRE DECODE SUBROUTINE
2ØØ5 IF F=2 THEN SYMBOL=PEEK(A): L=A: REM GUN NEEDS TO KNOW WAY TANK F
2Ø1Ø IF F=3 THEN SYMBOL=PEEK(B): L=B: REM WAY TANK B FACES
2Ø2Ø FOR X=1 TO 8
2Ø3Ø IF S(X)=SYMBOL THEN W=X: L=L+M(W): X=8
2Ø35 NEXT X
2Ø4Ø FOR U=1 TO 1Ø*RND(X): REM MOVE SHELL, JUST EFFECTS NO ATTEMPT TO
2Ø5Ø POKE L,SHELL: V=1: POKE L,BLANK: L=L+M(W)
2Ø6Ø NEXT U
2Ø7Ø FOR U=1 TO 1Ø
2Ø8Ø POKE L,INT(1ØØ*RND(X)): REM A LITTLE EXPLOSION AT END OF SHELL FL
2Ø9Ø NEXT U
21ØØ POKE L,BLANK: REM CLEANS UP LAST OF EXPLOSION
211Ø P=P+FIRE
212Ø RETURN
```

Table 1

Joystick Position	Value of P	Movement Value
Not FIRING and:		Current Position plus
Up	190	−32
Up/Right	158	−31
Right	222	+01
Down/Right	206	+33
Down	238	+32
Down/Left	230	+31
Left	246	−01
Left/Up	182	−33
FIRING and:		
No movement	127	+00
Up	063	−32
Up/Right	031	−31
Right	095	+01
Down/Right	079	+33
Down	111	+32
Down/Left	103	+31
Left	119	−01
Left/Up	055	−33

When the computer wants to check joystick 1 for movement, a 128 is POKEd into the keyboard address (polling must be suppressed at this time by the POKE 530,1 having already been executed) and only a key closure in row 1 will produce an output on the data bus. It is connected to the only low latch; all the rest are high. As long as 128 has been POKEd to the keyboard address, only a movement of joystick 1 can be read. If joystick 1 is not being moved, then no movement of joystick 2 will produce any output on the data bus. The same thing happens when the computer wants to read joystick 2; a 2 is POKEd to the keyboard address, which allows only the keys in row 7 to be active. No other keys will produce any output on the data bus. The only problem occurs when both joysticks are being moved at the same time. The worse case is when one is moving and firing and the other is only firing. The low set by the latch will feed through the two fire switches and be read as a movement by the bus driver when only a fire was intended. This can really destroy a game. Putting another isolating diode in any of the joystick lines doesn't help, as the LS125's are too sensitive. They see the forward voltage drop of the two diodes in series (one of the diodes D1 through D8 with any additional isolating diode), which causes the LS125 to always stay high. No key closures are detected. A pull-down resistor to a negative supply in conjunction with the isolating diode would be a solution.

Another solution is a hearing aid battery in series with the joystick's isolating diode. Or you could just give up the option of moving while firing. I use a software solution which, while not perfect, has yet to produce any objectional performance in any of the games I am running. Move the orange wire on joystick 1 from pin 9 to pin 4 and make the following software changes: define a new variable in demo program line 70 like FO = 251, and rewrite line 1000 as 1000 IF FIRE < P OR (P AND FO) THEN GOSUB 2000. This way the FIRE push buttons are connected to separate columns and can't feed back through each other's switches. Movements are now only slightly affected. If one joystick is doing Up/Right and the other is doing Up, then both will do Up/Right. But if one is doing Up/Right and the other is doing Up/Left or anything besides pure Up, then both will move their separate ways. In an actual game, this fix is sufficient.

You may contact Mr. Ennis at 212 20 St., Niceville, FL 32578.

MICRO™

An MC68000 Overview, Part 2

by Joe Jelemensky and Tom Whiteside

This second part of the 68000 overview provides simple programming examples to illustrate programming techniques and special features of the MC68000. Part 1 appeared in MICRO (52:32).

A Simple MC68000 Subroutine to Compare Two Strings

As a first example of programming the MC68000, consider the string comparison subroutine in figure 1. This simple subroutine will return with the Zero flag set if the first string matches the second. We will use the convention that all strings must be terminated with a zero. Address registers "A0" and "A1" will be used as pointers to the beginning of the two strings to be compared.

The subroutine documentation shows the calling sequence for STRCMP. In the subroutine usage documentation, the string pointers are initialized using the "MOVEA" (MOVE Address) instructions. The ".L" suffix on the MOVEA instructions tells the assembler that a long address is to be moved. A ".W" suffix specifies a sign extended 16-bit word address. In the MC68000, whenever an instruction has two operands, the first operand is the source and the second is the destination. For the first "MOVEA" instruction, "FIRST" is the absolute memory address to be used as a source for the long word to be moved to the destination "A0". Naturally, we can use any of the other addressing modes to specify the string location instead of absolute if we choose to.

After the string pointers are initialized, the user does a "JSR" (Jump to SubRoutine) to the string compare subroutine followed by a branch based on the Z flag. The "BEQ.S" is a branch if equal to zero (string match). The ".S" suffix tells the assembler that the destination is within the range of an 8-bit signed displacement. An ".L" suffix is used for 16-bit displacements.

The STRCMP subroutine begins and ends with "MOVEM.L" instructions to preserve all the registers that are used. The first "MOVEM" (MOVE Multiple) instruction moves the 32-bit contents of "A0" through "A1" and "D0" to the stack, which is pointed at by "A7". The assembler syntax for the register list on a "MOVEM" instruction can be in the form "A0/A1/A2/A3/A4/A5/A6/A7/D0/D1/D2/D3/D4/D5/D6/D7" or the shortened form "A0-A7/D0-D7". The "-(A7)" destination means to use the pre-decrement indirect addressing mode with the stack pointer "A7". This is equivalent to pushing the registers onto the system stack. The MC68000 assembler automatically adjusts the number to decrement or increment based on the total size of the operation. The final "MOVEM" instruction does just the opposite and loads "A0" through "A1" and "D0" from the stack using the post-increment indirect addressing mode from "A7". This is

***Figure 1:* MC68000 String Compare**

```
ROUTINE:        STRCMP --- STRing CoMPare

PURPOSE:        Compare two strings. If the first string matches the second
                string then return with the "Z" bit set. The user points A0 at
                the start of the first string, and A1 at the start of the second.

ASSUMPTIONS:    Strings terminate in a zero.
                sample string:      FCC 'this is a string'
                                    FCB 00

EXAMPLES:       First string        Second string     Z bit
                'cattle'            'cattle'          1          match
                'cat'               'cattle'          1          match
                ''                  'any string'      1          match
                'cattle'            'cat'             0       no match
                'cat'               'fatcat'          0       no match

USAGE:
                MOVEA.L  FIRST,A0     POINT AT FIRST STRING
                MOVEA.L  SECOND,A1    POINT AT SECOND STRING
                JSR      STRCMP       COMPARE THE STRINGS
                BEQ.S    xxxx         BRANCH IF MATCH IS FOUND


STRCMP  EQU *
*
        MOVEM.L  A0-A1/D0, -(A7)   PRESERVE ALL REGISTERS ON THE STACK
*
LOOP    EQU *
        MOVE.B   (A0)+,D0          GET THE NEXT CHARACTER IN THE
        BEQ.S    QUIT              FIRST STRING. IF AT THE END OF THE
                                   FIRST THEN IT MATCHES!
*
        CMP.B    D0,(A1)+          DOES NEXT CHARACTER IN THE
        BEQ.S    LOOP              SECOND MATCH? IF IT MATCHES KEEP
                                   TRYING. OTHERWISE, FALL THROUGH
                                   WITH Z BIT = 0.
*
QUIT    EQU *
        MOVEM.L  (A7)+,A0-A1/D0    RESTORE THE REGISTERS
        RTS                        AND LEAVE.
```

equivalent to pulling the registers off the system stack.

The actual string compare code is only four instructions long. The "MOVE.B" instruction moves the next character from the first string to "D0" and bumps the first string pointer to the next character. If we have reached the end of the first string then, by our convention that all strings end in zero, the Zero flag will be set and the short branch will be taken. (We found a match!) Otherwise, the "CMP.B" instruction checks to see if the lower byte in the data register matches the next character in the second string and bumps the pointer to the next character. If the characters match, the routine loops back to "LOOP" and tries the next character. If they do not match (including the case where we reach the end of the second string) the code falls through with the Zero flag cleared.

MC68000 Code for a Pascal Loop

The next example illustrates use of the MC68000's powerful DBcc looping instruction. DBcc is designed to speed up the "FOR", "WHILE", and "REPEAT UNTIL" loops used so frequently in high-level languages (HLL). The DBcc instruction has three parameters: a terminating condition, a data register, and a branch displacement. The instruction first sees if the terminating condition has been met, and if so, the branch specified by the branch offset is not taken. If the terminating condition is not met, the specified data register is decremented. If the result of this decrement is not −1 then the branch is taken. Otherwise, the branch falls through. The "cc" part of the instruction can be any of the conditions shown in table 6 (see Part 1, MICRO 52:38).

The following Pascal procedure and accompanying MC68000 code fragment illustrate how the DBcc instruction works (figure 2). The "REPEAT UNTIL" loop will continue until "i" has counted down to −1 or "CAT" equals "RAT". The MC68000 code uses "D0" for "i" and uses "DBEQ" to loop until "D0" = −1 or the previous comparison sets the Z flag. The DBcc instruction takes no more time than a simple branch instruction when the branch is taken. The equivalent code without the DBcc instruction is obviously much longer.

High-Level Language Procedure Calls

It is becoming increasingly important for processors to be able to handle subroutines efficiently as programs become more modular. This is true both

***Figure 2:* Example use of the DBcc (test condition Decrement and Branch) instruction.**

```
PROCEDURE typical;
CONST maxcnt = 10000;
VAR i,cat,rat:integer;
BEGIN
 i: = maxcnt;
 REPEAT
      .
      .
      .
  i: = i - 1;
 UNTIL (i < 0) OR (cat = rat);
END; { typical }

       MOVE #MAXCNT - 1,D0       INITIALIZE LOOP COUNTER
LOOP   EQU *
       .
       .
       MOVE.W     CAT,D1         GET CURRENT VALUE OF "CAT"
       CMP.W      RAT,D1         SEE IF "CAT" IS EQUAL TO "RAT"
       DBEQ.S     D0,LOOP        DECREMENT "1" AND LOOP UNTIL "I" =
*                                - 1 OR "CAT" EQUALS "RAT"
```

***Figure 3:* Example Pascal Procedure Call**

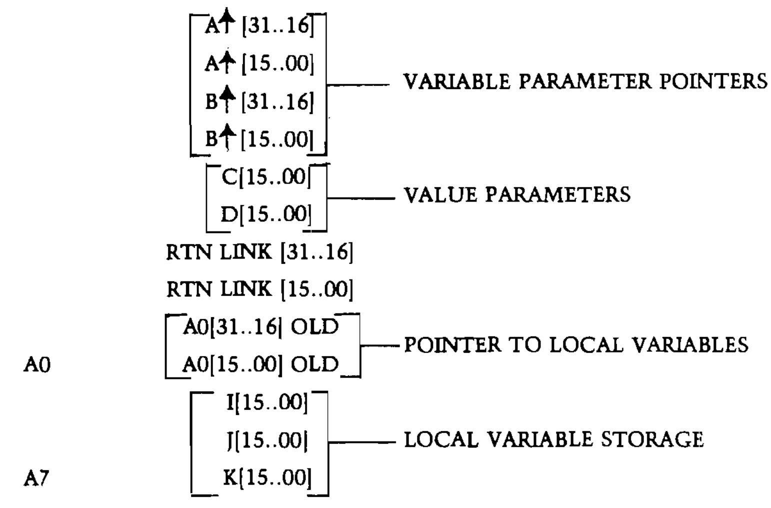
```
          PROCEDURE dojunk(VAR a,b:INTEGER; c,d:INTEGER);
          VAR i,j,k:INTEGER;
          BEGIN
           .
           .
           .
          END;

*
* SET UP FOR PROCEDURE CALL
*
*         INIT POINTERS TO VARIABLE PARAMETERS
*
          PEA       A           PUSH POINTER TO VARIABLE "A"
          PEA       B           PUSH POINTER TO VARIABLE "B"
*
*         MAKE COPY OF VALUE PARAMETERS ON THE STACK
*
          MOVE.W    C, - (A7)   PUSH COPY OF "C"
          MOVE.W    D, - (A7)   PUSH COPY OF "D"
*
* CALL PROCEDURE
*
          JSR       DOJUNK      CALL THE PROCEDURE
*
* CLEAN UP STACK AFTER PROCEDURE AND RESTORE REGISTERS
*
          ADDQ      #6,A7       REMOVE PARAMETER LIST FROM STACK
          .
          .
          .
DOJUNK    EQU *
          LINK      A0,# - 3    MAKE ROOM ON STACK FOR LOCAL
                                VARIABLES
          .
          .
          .
          UNLK      A0          CLEAR OFF STACK AND RESTORE A0
                                BEFORE LEAVING
          RTS
```

at the assembly-language level and for implementing modern HLLs. Figure 3 shows a typical Pascal procedure and how the procedure call might be implemented in MC68000 code.

Procedure "dojunk" is a typical Pascal procedure with variable parameters "a" and "b" and value parameters "c" and "d". For those unfamiliar with Pascal, a value parameter is a copy of a variable passed to a procedure. This copy can be modified by the procedure but the changes are not passed back to the calling procedure when the procedure exits. Variable parameters, unlike value parameters are passed back to the calling procedure as pointers to the variables. Value parameters are copies of the variables. The MC68000 code for calling procedure dojunk begins by initializing the stack with the procedure parameters. The two "PEA" instructions push the pointers to variables "A" and "B". The two "MOVE" instructions push copies of variables "C" and "D" on the stack. The procedure is then called with the JSR instruction. When the procedure returns to the calling routine, the space made on the stack for the parameters is removed with the ADDQ instruction.

When procedure dojunk (figure 4) is called, the LINK instruction provides a clean way to make room for local variables ("i", "j", "k"). The LINK instruction pushes the old value of "A0" on the stack, sets "A0" equal to the stack pointer, and subtracts the offset (in this case three words) from the stack pointer. With this technique, local variables are referenced with negative offsets off "A0" and parameters are referenced with positive offsets. This technique of using the stack supports re-entrant code and recursion with no problems. Before dojunk exits, it uses UNLK to clear off the local variable space it used.

***Figure 4:* Stack Usage for Pascal Procedure Call Example**



Range Checking Using the MC68000

The MC68000 "CHK" instruction provides an efficient mechanism to insure that array bounds are not exceeded in high-level language implementations. As an example of range checking with CHK, consider the problem of enforcing array boundary checking in a simple Pascal program. Figure 5 shows the stack usage for a simple variable declaration using the same scheme described in the "Procedure Call" example. The individual elements of the array "data" will be accessed by indexing off the pointer "A0". The range checking's duty is to insure that we do not attempt to access an element of "data" before its true beginning or after element "MAX". The consequences of exceeding the array bounds range from inadvertently modifying another variable (such as "i", "j", or "misc") to stepping on a subroutine return link! Figure 6 shows a simple Pascal statement to clear the "ith" element of "data" and the MC68000 code to accomplish it with the necessary range check. The code assumes that register "D0" contains the current state of "i". The CHK instruction examines the lower word in the specified register ("D0") and generates an error trap if it is less than 0 or greater than "MAX". Otherwise, the code falls through to the MOVE instruction that clears the "ith" element of "data".

***Figure 5:* Array Storage on the Stack for the Range Checking Example**

```
{ sample variable declaration }
Var i,j  : integer;                          { stuff before data array }
 data    : array[0..MAX] of integer;
 misc.   : integer;                          { stuff after data array }
```

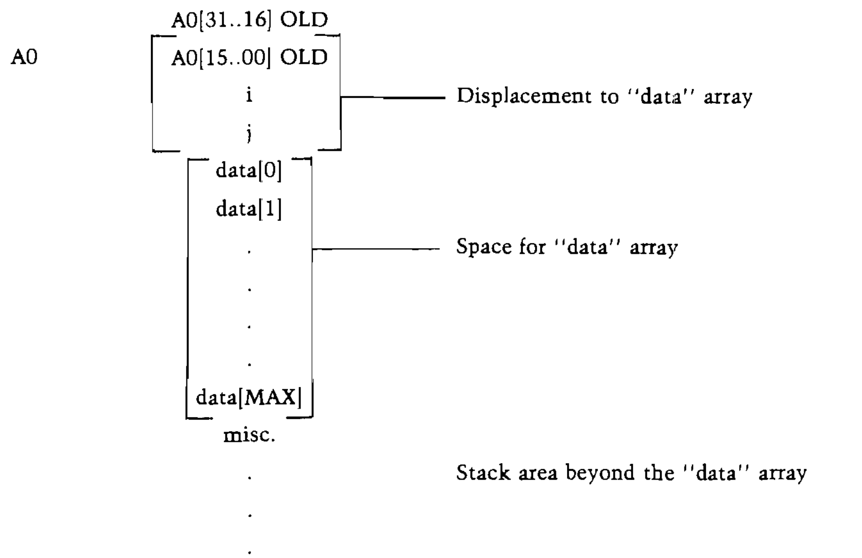


***Figure 6:* Example of Range Checking Using the MC68000**

```
{ Pascal statement to clear the "ith" element of "data" }
data[i]: =0; { note: i < 0 or i > MAX is a range error!!! }

*
* MC68000 CODE TO EXECUTE THE ABOVE STATEMENT
*
*
*    ASSUME D0 CONTAINS THE CURRENT CONTENTS OF "i"
*
*       GENERATE A TRAP IF "i" < 0 OR IF "i" > MAX
*
        CHK #MAX,D0
*
*       HERE IF NO RANGE ERROR OCCURRED
*
        MOVE.W #0,DISP(A0,D0) CLEAR THE "iTH" ELEMENT OF DATA
*
           .
           .
           .
```

Conclusion

We hope this article has given you some insight into the extensive power of the MC68000. The 16-bit data bus, the 16 megabytes of directly addressable memory, the sixteen 32-bit user-accessible registers, the powerful instruction set, numerous addressing modes, and fast execution speed are enough to really get folks excited.

Future Growth

The MC68000 is not the only member of this powerful microprocessor family. Motorola is taking advantage of its modular, microprogrammed structure to develop other processors which are upward and downward compatible to the MC68000.

The MC68008 is a machine and assembly level-compatible version of the MC68000 with an 8-bit data and 20-bit address bus for low-cost systems that need the performance of the MC68000 and can tolerate a slight decrease in throughput for reduced system costs.

With the MC68010, Motorola adds virtual machine capabilities to the MC68000 architecture.

A full 32-bit implementation (32-bit address and data buses) of the architecture with an enhanced instruction set and on-board cache is in design. The part will be designated the MC68020 and will contain all of the additional features of the MC68010.

A full complement of peripherals for the MC68000 family have been introduced or are in development. These include a Direct Memory Access controller (MC68440), a Memory Management Unit (MC68451), a Floating Point Arithmetic Co-processor (MC68881), and several data communication parts.

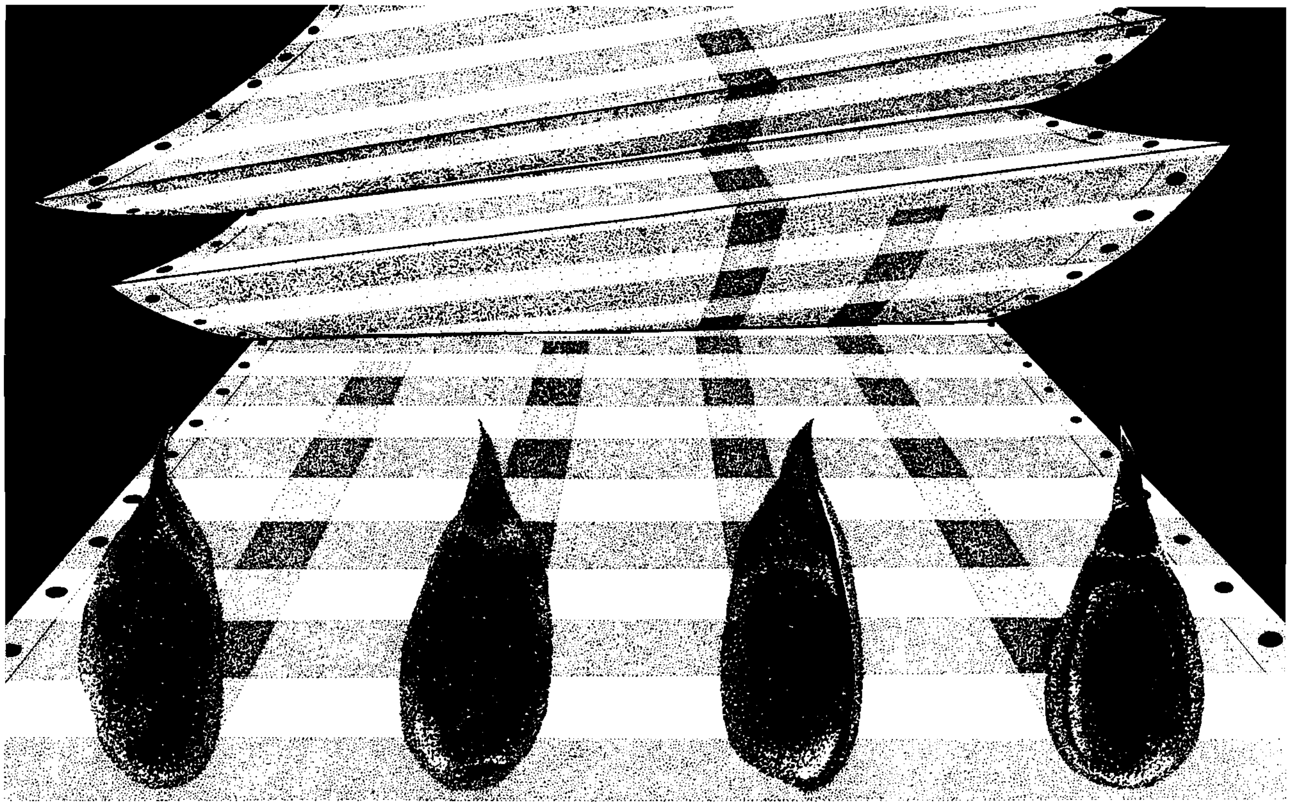
Because of this commitment to the continuation and enhancement of the MC68000 family, systems and software for the MC68000 are insured a long useful life.

References

1. "16-Bit Microprocessor User's Manual," Motorola Inc., 1982.
2. "Motorola Resident Structured Assembler Manual," Motorola Inc., 1982.

MICRO™

FOUR TO GROW



Four software packages, that is, to make your Apple® computer really grow! Versatility is the key to computing success, and with Advanced Operating Systems, it's as easy as one, two, three.

APPLE-AIDS is a collection of 12 Applesoft and machine language programs designed to bring you greater flexibility and control over your microcomputer.

In addition to several programs which allow you to examine and change various track/sector contents, format new disks, and create and edit EXEC files, APPLE-AIDS includes two programs you can't afford to be without. UNDELETE FILES allows you to recover accidentally deleted files which you thought were lost forever. KILL DOS removes DOS from any disk freeing up more space for your programs or data.

Russ Adams, a reviewer for INFOWORLD, writes, "Its (APPLE-AIDS) documentation is the finest I have seen since joining the INFOWORLD Software Review Board." Ease of use and extensive documentation make APPLE-AIDS a valuable addition to every Apple software library. #26066, $49.95

HELLO CENTRAL! changes your Apple II® or Apple II Plus® into a highly versatile communications center.

Establish direct communications with other computers of any make or tap into the wealth of information available from the hundreds of public-base services like THE SOURCE.

HELLO CENTRAL!'s most unique feature is its text buffer. Messages or entire files can be uploaded and downloaded into the text buffer which holds up to 18,000 characters. The buffer text can be scanned, changed, and then saved to disk.

The best thing about HELLO CENTRAL! is that you don't have to be a professional to get professional results. A few simple keystrokes will accomplish any of your text/communications tasks. #26081, $99.00

MUSIC GAMES skillfully combines the sound and high resolution graphics of the popular Apple computer to provide both visual and audible reinforcement to the study of music. Twelve different programs, selected from a menu, cover ear training, note recognition and writing, rhythm practice, and listening enjoyment. These versatile games are written for ages 5 through adults. #26116, $39.95

PEN PAL gives you the power and versatility of higher-priced word processors in an easy-to-use format. With only 29 commands, you can quickly and easily create lengthy reports or short memos. PEN PAL gives you 40- and 80- column screening, joystick control of the cursor for quick editing, complete printer control, plus both upper and lower case capabilities. PEN PAL is the affordable solution to your word processing needs. #25115, $59.95

Advanced Operating Systems has four good ways to make you and your Apple grow and grow!

Now available at your local software retailer, or call 1-800-428-3696 to order. (Indiana residents, call (317) 298-5566.) MasterCard and VISA accepted.

ADVANCED OPERATING SYSTEMS

4300 West 62nd Street
P.O. Box 7092
Indianapolis, IN 46206

A Division of
Howard W. Sams & Co., Inc.

Apple, Apple II and Apple II Plus are registered trademarks of Apple Computers, Inc.

A Monitor for the Color Computer

by Ralph Tenny

This article provides step-by-step instructions to get composite video from the video section of the Color Computer. You will use this signal to drive a standard video monitor instead of a color or black-and-white TV set.

Required:

TRS-80 Color Computer
A monitor-quality CRT

The Radio Shack Color Computer, an excellent low-cost computer, uses a color TV set as the intended display device. Unfortunately, TV sets have relatively low resolution, which limits the clarity of the display. The use of a TV set can be either good or bad, depending on whether color graphics are used in your program. Since I use it as a text processing system, I need a video terminal with maximum display clarity. Even the relatively low character density of the Color Computer's video output isn't very clear on the average TV set — either color or B&W. The signal itself is of high technical quality, and a higher-resolution display can make a great improvement in the readability of the display. If a B&W monitor is substituted for the TV set, this goal is achieved. As you can see from photo 1, my video monitor (which is not a super-high resolution unit) gives an excellent display capability when used with the Color Computer.

Since the remainder of the article describes a process of modifying the Color Computer, you should realize that any internal modifications will probably void the warranty. If the computer is more than three months old, the warranty will have expired. If you modify the computer and it needs repair, you may have to remove the modification before Radio Shack will repair it. I have not had any problems in over a year, and I still feel comfortable with my decision.

The Color Computer's video output is a complete TV signal and is generated by a very low-power TV transmitter mounted within the Color Computer. Part of the display problem is the fact that the TV tuner is deliberately limited, therefore the signal must also be limited. Even if the tuner is bypassed, the TV's display resolution is reduced as a cost-saving measure, since high resolution is not required for normal TV viewing.

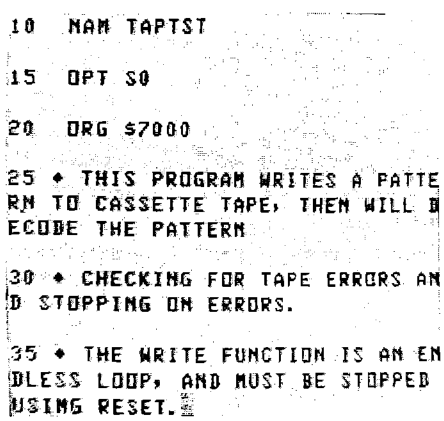


***Photo 1:* A close-up look at a Color Computer display on a video monitor. Note that each pixel is a sharply-defined square.**

The Color Computer uses an MC6847 Video Display Generator to produce all the signals required to drive a TV set; these include composite sync, blanking, video luminance, and color information. The basic approach is to use a buffer amplifier to process the output of the MC6847, enabling the signal to be fed directly to a video monitor. Figure 1 shows the schematic of this buffer amplifier; it was patterned after circuitry suggested by Motorola, the manufacturer of the 6847. This amplifier was built on thin, two-sided, copper-clad board with one side etched into small pads. Figure 2 is a full-size layout of the amplifier board, and figure 3 shows the parts layout.

Access to the Color Computer is easy and requires only that you take out seven screws to remove the top. Note that one screw is located beneath a factory seal. If you remove this screw

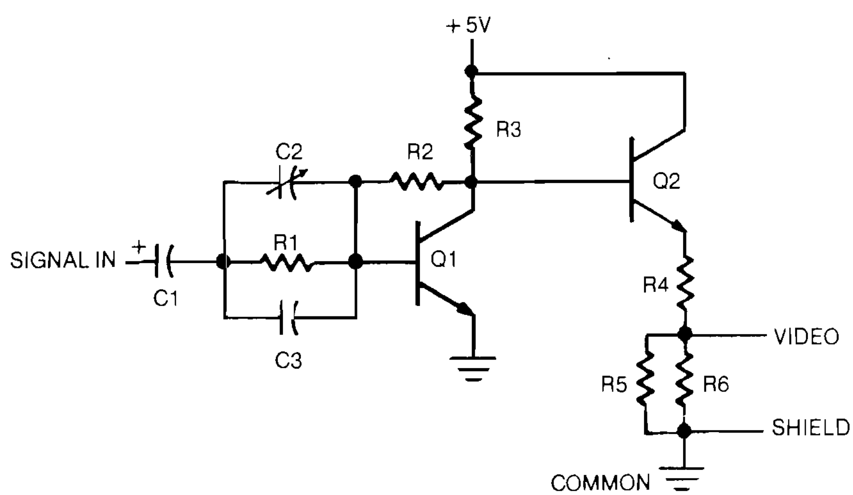


Figure 1: Schematic for the video buffer amplifier, which mounts internal to the computer. It amplifies the video output of the 6847 and turns it into a low-impedence signal suitable for driving a cable to the external monitor.

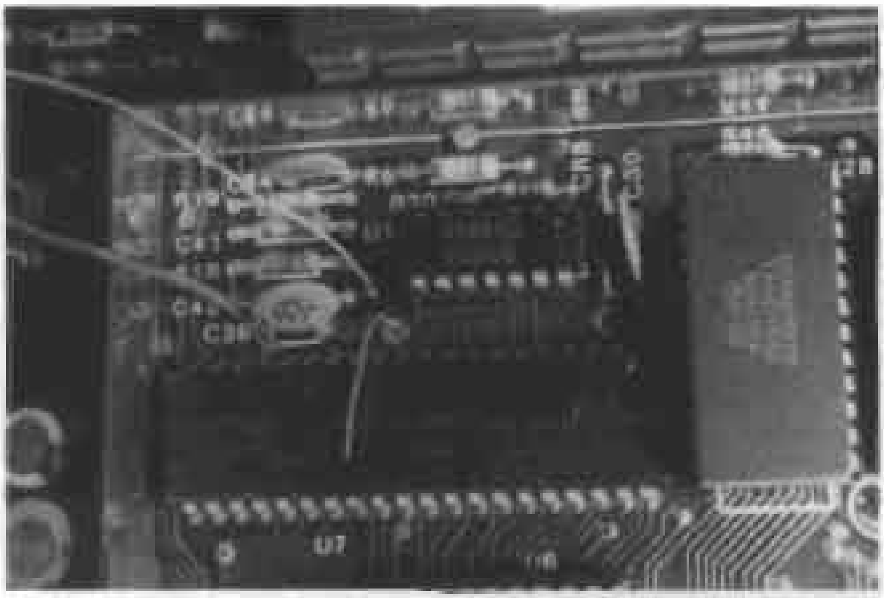


Photo 2: Interior view of the area where the amplifier will be mounted. The long IC is the MC6847, which will be covered by the buffer amplifier as shown in photo 3.

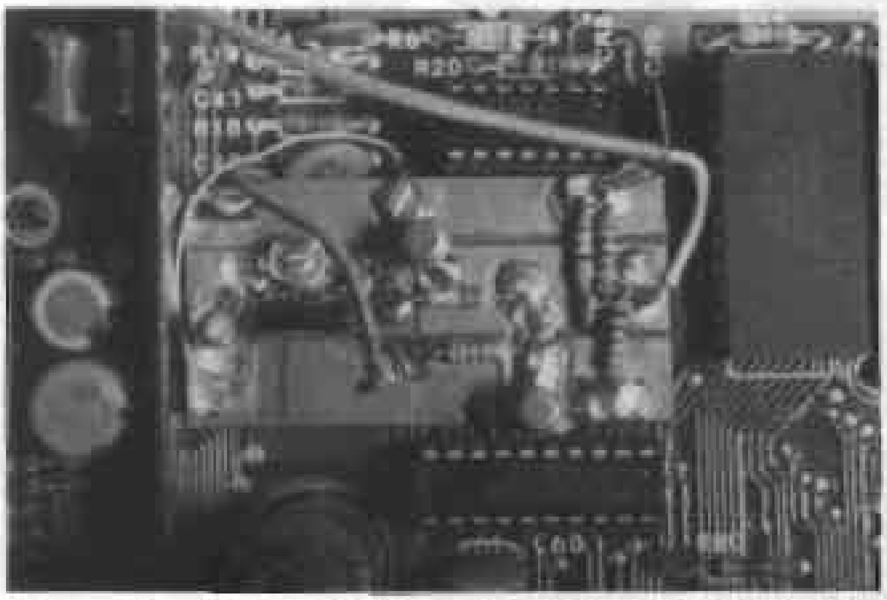


Photo 3: A view of the buffer amplifier mounted, with the output cable attached.

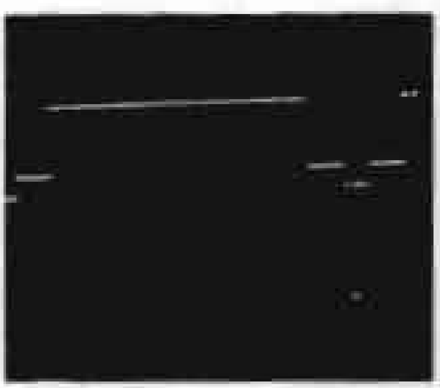


Photo 4: Oscillogram of the output video showing one video frame.

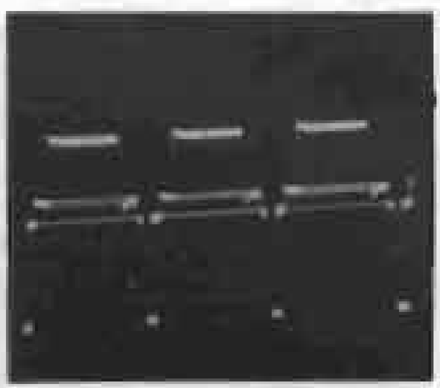


Photo 5: An oscillogram of three lines of horizontal video.

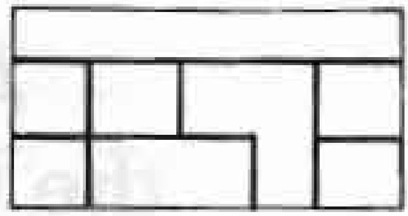


Figure 2: Full-size layout of the amplifier board. Make the board by etching the lines shown to divide the board into eight segments as shown.

you will deface the seal, which voids your warranty.

Once you loosen the screws, put a piece of tape over each hole so the screws will not fall out when you turn the computer upright. With the screws loosened, the top will lift off, leaving the keyboard resting on standoffs and revealing the inner RF shield.

Note that the top of the shield is a friction fit to the shield sides *via* numerous spring-loaded fingers. Work the shield top off by lifting it a little bit all around; keep lifting until it lifts straight off. Once you can see inside, find the MC6847 and MC1372 ICs as shown in photo 2. Also, in photo 2, you will note a solid ground wire curving up over the 6847, plus a heavy wire coming from one side of C26 and smaller wire coming off the circuit board between C42 and the 1372. The solid wire is circuit common, the large wire is +5 volts, and the small wire is the signal input to the buffer board. Photo 3 shows the video buffer installed, resting on the 6847. The large wire has been soldered to the +5 volt input of the buffer board, the ground wire to the amplifier common, and the small wire to the amplifier input. A small 183-ohm cable has its shield soldered to the amplifier common and the center conductor to the amplifier output. The other end of the 183-ohm cable runs across to the corner of the RF shield (the top left corner of the shield as seen from the front of the computer)

Parts List for Video Buffer Amplifier Circuit

C3	68 pF dipped mylar capacitor
R1	33K ohm, ¼-watt resistor
R2	62K ohm, ¼-watt carbon resistor
R3	5.1K ohm, ¼-watt carbon resistor
R4, R5, R6	51 ohm, ¼-watt carbon resistor
Q1, Q2	2N2222 NPN transistors

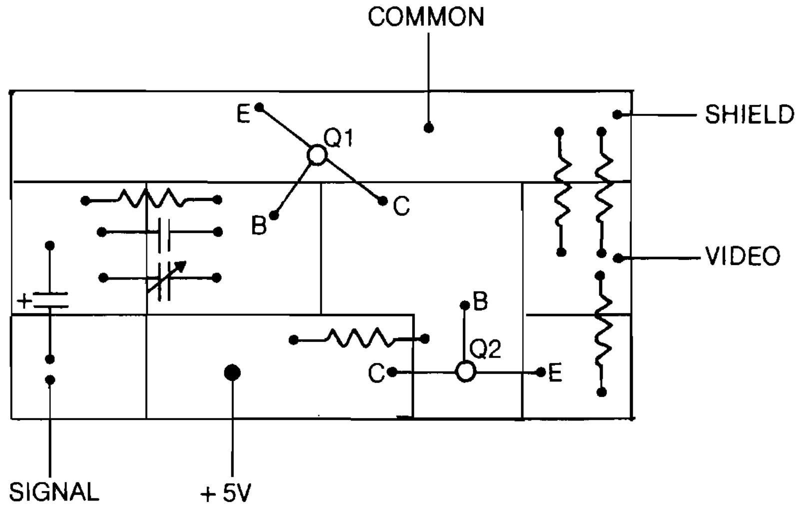


***Figure 3:* Parts layout for the amplifier board. Each part is soldered into place in the approximate location sketched.**

Once the amplifier is in place, turn on the computer and check the video output from the cable. Photo 4 shows one vertical frame of video, and photo 5 shows three horizontal lines of video. If your amplifier does not have similar output, double-check the circuit and adjust C2 until the response is correct.

After testing the amplifier, send its output into a video monitor and make whatever level adjustments are needed to obtain a good picture. Once the monitor is working, re-install the computer cover and enjoy your computer's new display.

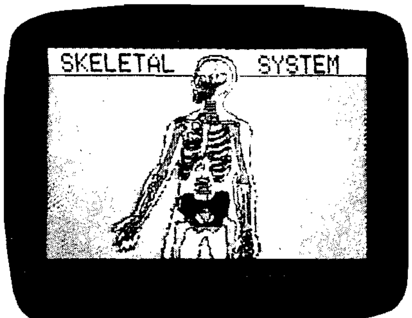
You may contact the author at P.O. Box 545, Richardson, Texas 75080.

MICRO

FOR COMPLETE GRAPHICS:

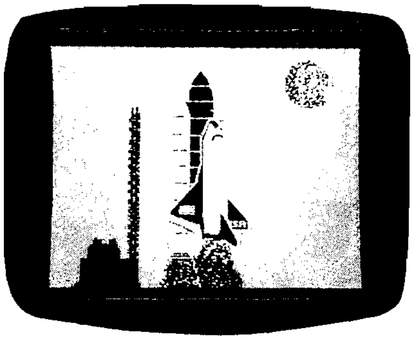
VersaWriter

EDUCATION

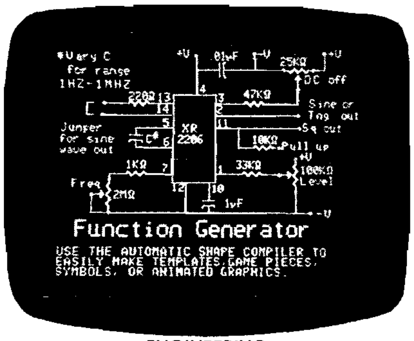


ARTIST

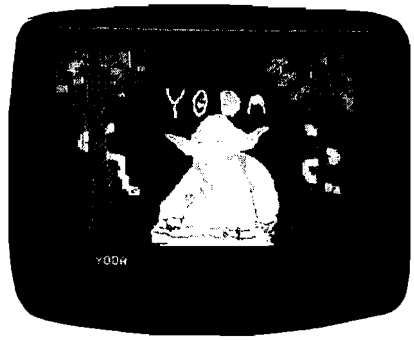
GAME PROGRAMMER



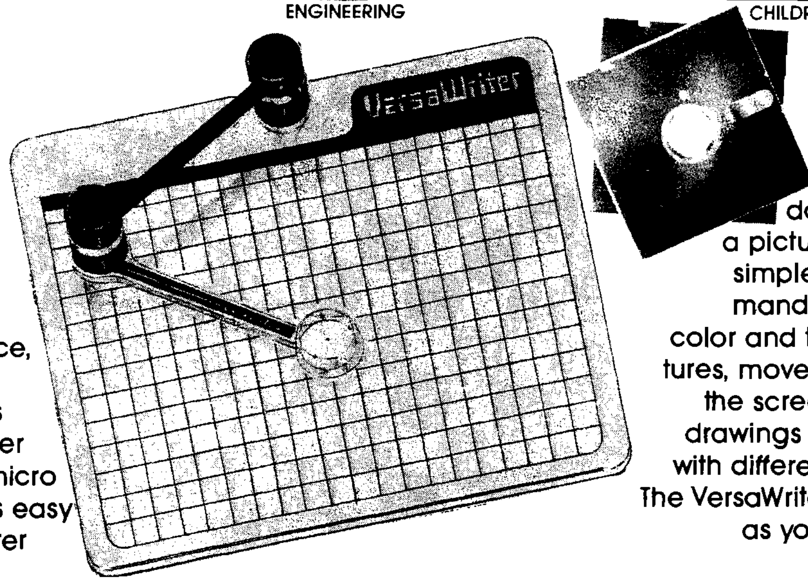
HOBBIEST



ENGINEERING



CHILDREN



Whether you are a teacher, an artist, an engineer, a programmer, or a hobbiest with little or no programming experience, the VersaWriter is the answer to your graphics need. With the VersaWriter exploring the world of micro computer graphics is as easy as tracing. The VersaWriter doesn't just trace a picture though. With simple one key commands you can add color and text to your pictures, move objects across the screen, make scale drawings and even draw with different size brushes. The VersaWriter is as limitless as your imagination.

For complete graphics on your Apple II/II+, or IBM PC, the Versa Writer from Versa Computing, Inc. is your answer. Complete hardware/software system ready for use - $299.

Send for information on the complete line of VERSAWARE & Graphics Products

Dealer Inquiries Welcome

VERSA COMPUTING, INC.

3541 Old Conejo Road, Suite 104 • Newbury Park, CA 91320 • (805) 498-1956

FLEX and the TRS-80 Color Computer

by Ronald W. Anderson

Here is a brief description of the FLEX09 operating system as implemented on the TRS-80 Color Computer.

The configuration described here is available from Frank Hogg Laboratories (130 Midtown Plaza, Syracuse, NY 13210) with 64K or RAM installed ($1275), as well as an assortment of disk drive packages. If you already have a TRS-80, then Frank Hogg Laboratories sells the operating system and modification instructions separately ($99). While it is possible to run the FLEX operating system with one disk drive, two are recommended. You may choose from a minimum package of one single-sided, double-density, 35-track drive, or up to three double-sided, double-density, 40-track drives. The latter will hold up to 375K of data on each drive. You may also use one of the 80-track, double-density drives and have over 700K bytes on one disk, although the 80-track version is not compatible with Radio Shack's disk operating system.

FLEX is the standard operating system for 6800 and 6809 systems. FLEX is so universal that there are versions that will run on the Motorola Exorciser. FLEX is a "Unix-like" operating system and has a nice set of calls that do all the work to interface your assembler program to the disk drivers. There are literally 100 utility programs available for FLEX.

The manual that comes with the operating system is basically the FLEX operating system manual with additional pages that apply to the Color Computer. There is a well-written section on getting your system up and running. First you configure the software to match your hardware. The system boots up expecting 35-track, single-sided drives. To access more tracks or double-sided drives, you must run a utility program that will tell the system you have 40 tracks and double-sided drives, for example. Your drives need not all be alike. You may specify drive 0 as being single-sided and having 35 tracks, and drive 1 as being double-sided and having 40 tracks. Although this might seem like a nuisance, you only need to run the utility program once. This generates a command file that is appended to FLEX containing information on your drives, the terminal, and drivers for your printer.

I did a bit of experimenting with the SETUP command that does all the configuring, and discovered I could specify "reverse video," which is really "normal video" to anyone who has worked in front of a terminal for any time. I find a blinking cursor to be distracting, so I made it a plain block cursor. Now my system will power up just the way I want it.

Another feature is the keyboard's ability to provide all the ASCII codes. The control codes, such as Control C, are generated by holding the shift and up arrow simultaneously (equivalent of the control key on a terminal), and then pressing the desired key, C for example. If you ever program in Pascal and/or "C", you will need several characters that are not included on the Color Computer keyboard.

Without going into detail, I will give an example of the key assignments and how to access them. You may remember that upper case (SHIFT) 8 is a left parenthesis "(", and SHIFT 9 is a right parenthesis ")". Use the CONTROL combination (SHIFT ∧) and type 8 and you get a left square bracket "[". SHIFT ∧ 9 and you get a right square bracket "]". "C" requires the use of curly braces. SHIFT ∧ BREAK 8 will cause a left brace "{" to be generated, and the same combination for a 9 will generate a right curly brace "}". The control and shift keys are depressed first, and are all held simultaneously before keying the 8 or 9. Holding three keys down and typing a fourth is not easy, but it's better than not being able to generate the code for those characters.

Sometimes disk drives run slightly fast, or irregularly, and squeezing the sectors close enough to get 18 on a track results in unreliable disk access. There is a NEWDISK utility that will automatically reduce the number of sectors to 17 if it encounters problems. Another NEWDISKA utility will put 18 sectors on closer together. I have had no trouble using NEWDISKA. A sector in the FLEX system is 256 bytes long, but four bytes are used for system purposes, so a sector actually holds 252 bytes of data. A single-sided, 35-track disk will hold 612 sectors (using NEWDISKA). If your drive has 40-track capability, you will get 702. A double-sided 40-track will result in 1404. If these numbers don't seem to add up correctly, it is because FLEX uses the first track (track 0) for a loader, disk system information record, and file directory sectors. Also, FLEX requires that track 0 always be single density. Therefore, the tracks actually available to the user are one less than the total number on the disk.

Once FLEX is running, make another copy of the system disk. Use NEWDISK or NEWDISKA to format a blank disk, and then use a utility called PUTBOOT.LDR to install the loader on the disk. The loader is readable by the Color Computer disk operating system. It loads a short program that loads and executes a longer loader program that loads all of the FLEX operating system. It sounds complicated but only takes a few seconds to execute. You *must* put the boot loader on your disk first. If you have copied any files onto the newly formatted disk, there is a chance that the system may have allocated you space where the boot loader needs to be. PUTBOOT will inform you that you can only install the loader on a newly formatted disk, and quit.

After installing the loader, you may copy everything from the system disk to your new system disk *except the loader installation utility PUTBOOT.LDR*. An attempt to copy it will get you a DISK SPACE FULL error and a lot of garbage on your disk. If you try to GET PUTBOOT.LDR (for those of you not familiar with FLEX, GET is a utility that reads a disk file to memory but doesn't execute it), you will find that it tries to execute, finds that you haven't specified a drive number, and erases itself from memory before returning to FLEX. Unless you enjoy "puzzles" and are quite familiar with FLEX and the format of binary files, you will have to use your master disk to make all further system disks.

Having copied all the supplied files to your system disk, you must run one further utility, called LINK. LINK will tell the boot loader just where on the disk the file FLEX.SYS is located. My advice is to make two system disks, and hide the master and one of them in a safe place where they won't get hot, bent, or damaged by a magnet. When you wipe out your working system disk you can get the backup and generate another backup for the safe storage area.

The video display looks like a terminal. Most terminals accept control codes to do such things as clear the screen, position the cursor for the next letter to be output, etc. A set of such codes has been provided. You can control the display without getting into the assembler-level CRT driver code. That will make it easier for software suppliers to write compatible software for the Color Computer.

The Color Computer format is 16 lines of 32 characters each, and the character generator displays lower case in reverse video, which is barely usable. Software will soon be available to allow use of any standard ASCII RS-232 terminal on the serial port of the Color Computer. I have a preliminary version of the software which will eliminate the display problem completely, but I imagine that many purchasers will want to use a TV set and the supplied keyboard for some time, before investing in a terminal. The enhanced display software will therefore be welcome.

A SDC (single disk copy) utility allows copying with a single drive. It reads files from the source disk until memory is full or it has read all the files to be copied, then it prompts you to insert the destination disk for a write cycle, etc., until the copy is complete. LINK and PUTBOOT.LDR both prompt you, and therefore give an opportunity to change disks so these operations may be performed with a single disk drive also.

The HELP utility and file help you find or remember FLEX commands. If, for example, you type HELP,NEWDISK you will get a brief description of what the NEWDISK utility does, and a reference to a page in the manual.

If you have a serial printer, you will have no trouble configuring the system so you can use it effectively. A parallel printer will require a serial adaptor of some sort. Epson provides one for their printers, and Computerware of Encinitas, CA, offers one for about $60.

Of course you can run Radio Shack Extended Disk BASIC, which comes in the Color Computer in ROM. The RS disk operating system is well thought out. Both Random Access and Sequential files are implemented. Since the Color Computer comes up running the BASIC from ROM, you might wonder where the operating system is. Actually, it is part of BASIC. While running BASIC you may DIR a disk (prints the directory of a RS disk). You may copy or back up a disk, load and save programs (either BASIC or machine language), and still in BASIC, you may read or write any sector on the disk by specifying the track and sector number, and the ID of two string variables. The first 128 bytes of the sector is read to/written from the first named string variable, and the last 128 bytes to/from the second. That means you can write your own disk system and have fast access! Naturally, the RS BASIC has all the color graphics commands and sound commands, so you can write game programs and/or use graphics.

The Color Computer is a good buy for anyone wanting to get into this expensive hobby with a small investment. If you have the minimum system from RS, which is very inexpensive, you can first have RS upgrade your Color Computer to 32K, and then do the very simple modification to 64K. From that point, you are off into the world of FLEX, with at least five Pascal compilers, three or four BASIC interpreters, several versions of "C", Fortran, Cobol, a couple versions of FORTH, three editors, a good text processor, several assemblers, an excellent debugger program for assembler programs, and much more.

Mr. Anderson is vice president in charge of engineering for Industrial Computer Controls Corp. in Ann Arbor, MI. You may contact him at 3540 Sturbridge Ct., Ann Arbor, MI 48105.

MICRO

68000
Binary Arithmetic Operations

by Joe Hootman

A short discussion of the various instructions for binary arithmetic including addressing and sign information.

The most extensive list of instructions, and some of the most frequently used, are the Binary Arithmetic Operations (table 1). Most of the binary operations are straightforward however, there are several instructions unique to the 68000.

The binary operations usually do not apply to operations on the address registers. However, in the binary arithmetic instruction implementation of the 68000, there are several instructions such as ADDA, CMPA, and SUBA that are designed for operation on the address register. The operations on the address register allow special addressing operations to be carried out and, more importantly, allow the comparison of the magnitude of the address register without using the CHK instruction. Clearly the address registers can be used as data registers or index registers. Three instructions deal with sign extension: ADDX, EXT, and NEGX. All of the sign extension instructions sense the sign bit of the operation and the sign is extended through the length of the word.

When arithmetic instructions are considered, the implementation of the signed multiply and divide and the unsigned multiply and divide must be considered a most worthwhile and powerful addition to the instruction set. The signed and unsigned multiply and divide instructions are all 16-bit instructions. The data to be operated on is a word in length; the result of the operation is 32 bits long (long word). The interpretation of the 32 bits depends on the particular instruction. For example, if D0 contains $8055, D1 contains $0002, and MULS D1, D0 is executed, the result ($FFFF00AA) will be left in data register D0. Since the most significant bit of the word is set, this indicates that the result is negative and the N bit is set in the CCR. If D0 contains $8055, D1 contains $0002, and MULU D1, D0 is executed, then the result ($000100AA) will be in D0.

The signed and unsigned divide have characteristics similar to the multiply instructions. If division by zero is attempted, a trap will occur and overflow is indicated by the state of the V bit in the CCR. If Z is set then the quotient is zero. The N bit follows the most significant bit of the result. In both the signed and unsigned divide the quotient is the lower 16 bits of the destination register and the upper 16 bits is the remainder. The sign is reflected by the most significant bit of the result.

The TST instruction subtracts the designated data from zero and the appropriate bits set in the CCR. This instruction testing of byte, word, and long word data is reflected in the CCR.

***Table 1:* Binary Arithmetic Operations**

*The addressing modes will be covered in future issues.

Mnemonic	Data Size/CCR	Function	Comments
ADD	8, 16, 32 CCR X N Z V C * * * * *	Add Binary	This operation adds the binary data designated by the source to the data designated by the destination and leaves the result in the destination. If the effective address is a source then all addressing modes can be used. Opword Format 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 1 1 0 1 | Register | Op Mode | Effective Address Mode | Register Register — Any of the eight data registers. Op Mode Byte Word Long Word Operation 000 001 010 Dn + EA → Dn 100 101 110 EA + Dn → EA The source can have all the address modes except 13, 14.* The destination can have all the address modes except 1, 2, 10, 11, 12, 13, 14.*
ADDA	16, 32 CCR X N Z V C - - - - -	Add Address	This instruction adds the source data to the designated address register and leaves the result in the address register. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 1 1 0 1 | Register | Op Mode | Effective Address Mode | Register The register field can be any of the 8 address registers. This is always the destination. Op Mode 011 - Word operation the sign will be extended to all 32 bits of the address register. 111 - Long operation All addressing modes are allowed except 13, 14.*

(continued)

Mnemonic	Data Size/CCR	Function	Comments
ADDI	8, 16, 32 CCR X N Z V C * * * * *	Add Immediate	This instruction adds immediate data to the destination data and leaves the result in the destination.

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	0	1	1	0	Size	Effective Address Mode	Register
Word data (16 bits)								Byte data (8 bits)		
Long word (32 bits, including previous word)										

Size field: 00 - Byte
01 - Word
10 - Long word

All addressing modes except 10, 11, 12, 13, 14 can be used as a destination.*

Mnemonic	Data Size/CCR	Function	Comments
ADDQ	8, 16, 32 CCR X N Z V C * * * * *	Add Quick	This instruction allows the binary addition of any data from 1 to 8.

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
0	1	0	1	Data	0	Size	Effective Address Mode	Register

The data field can contain any integer from 1 to 7.
Size field: 00 - Byte
01 - Word
10 - Long word

All addressing modes except 10, 11, 12, 13, and 14 can be used as a destination.*

Mnemonic	Data Size/CCR	Function	Comments
ADDX	8, 16, 32 CCR X N Z V C * * * * *	Add Extended	This instruction adds the source to the destination and leaves the result in the destination. The sign bit of the result is extended to fill the word.

15	14	13	12	11 10 9	8	7 6	5 4	3	2 1 0
1	1	0	1	Register R#	1	Size	0 0	R/M	Register R#
				Destination Register					Source Register

Register field R# designates any one of eight registers.

If R/M = 0 then a data register is specified.
If R/M = 1 then an address register is specified.

Size field: 00 - Byte operation
01 - Word operation
10 - Long word operation

Mnemonic	Data Size/CCR	Function	Comments
CLR	8, 16, 32 CCR X N Z V C - 0 1 0 0	Clear an Operand	This instruction clears the effective address.

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	0	0	1	0	Size	Effective Address Mode	Register

Size field: 00 - Byte operation
01 - Word operation
10 - Long word operation

The following address modes cannot be used as destinations: 2, 10, 11, 12, 13, 14.*

(Continued on page 30)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Function	Comments
CMP	8, 16, 32 CCR X N Z V C - * * * *	Compare	This operation subtracts from the destination the source; the destination is not changed.

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register	Op Mode	Effective Address Mode	Register

Register field defines the destination data register.

Op Mode field defines the size of the data to be compared.

000 - Byte
001 - Word
010 - Long word

All but the Quick Immediate and Implied addressing can be used for an effective address 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
CMPA	16, 32 CCR X N Z V C - * * * *	Compare Addresses	This instruction subtracts the effective address from the destination and leaves the destination unchanged.

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register	Op Mode	Effective Address Mode	Register

Register field defines the destination address register.

Op Mode field specifies the size of the operand.

011 - Word operator
111 - Long word

All effective addressing modes except the implied mode can be used.

Mnemonic	Data Size/CCR	Function	Comments
CMPI	8, 16, 32 CCR X N Z V C - * * * *	Compare Immediate	This instruction subtracts the immediate data from the destination. The condition codes a set consistant with the results of the operation.

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	0	0	0	1	1	0	0	Size	Effective Address Mode	Register
Word data (16 bits)								Byte data (8 bits)		
Long data (32 bits including previous word)										

Size field: 00 - Byte operation
01 - Word operation
10 - Long word operation

The following destination effective addresses cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
CMPM	8, 16, 32 CCR X N Z V C - * * * *	Compare Memory	This operation is used to subtract the source from the destination. The CCR is set in accord with the result. The contents of the destination are not changed. The addressing is always done using postincrement addressing.

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	1	1	Register Rx	1	Size	0	0	1	Register Ry

Rx must be an address register and in the destination.
Ry must be an address register and is always the source.

Only the Post Increment mode can be used.

(continued)

MICRObits

(Continued from page 28)

Elephant Disks

5¼" SS/SD $21/box, SS/DD (48 TPI) $24/box, SS/DD (96 TPI) and DS/DD (48 TPI) $30/box, DS/DD (96 TPI) $36/box. Write for prices on 8". Add $1 per order shipping *via* UPS in USA, NJ add 5% sales tax.

Baker Enterprises
15 Windsor Drive
Atco, NJ 08004

SYM-1 Computer Plus 6502 Books

SYM-1 assembled and tested, never used (original packing). SYM programming manual and reference manual included - $130.00. PMC (EMA 5/6B) power supply - $19.95 (cable connection for SYM included). 6502 software design (L.J. Scanlon) - $8.95 new. Programming and interfacing 6502 with experiments (M.L. DeJong) - $10.95 new. SASE:

Frank Janda
19 Wilson Drive
Framingham, MA 01701
(617) 862-3120 ext.209

OSI Super Defender

Play this great arcade game at home. All machine code includes: scanner; smart bombs; laser fire; moving mountains and more. Save your humanoids from the alien landers. Very smooth (half character moves) graphics. $14.95 for C1,2,4 tape or 5¼" disk.

DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

Commodore 64 Software

Sprite Editor - $12, Dumb Terminal - $10, Disassembler - $15, Cross-Reference Generator for BASIC programs - $10. Descriptive catalogue - 50 cents. Contact by US mail or CompuServe 70140,223.

NEON SYSTEMS
5108 N. 23rd Road
Arlington, VA 22207

The Wrath of Khan

Can you defeat a Superman? Command Enterprise in a tactical battle against the Reliant. C1-P controls Reliant's maneuvers and weapons *via* Artificial Intelligence logic routines. Features full status display, sensors, photon torpedoes, phasers, deflectors, and more. Cassette 8K - $14.95 ppd.

Cygnus Software
791 W. Oakland Park Blvd.
Suite 432
Ft. Lauderdale, FL 33311

(continued)

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Function	Comments
DIVS	16 CCR X N Z V C - • • • 0	Signed Divide	This operation divides the destination by the source. The result is left in the destination. The source is a 16-bit word and the destination is a long word operation.The lower 16 bits are the quotient and the remainder is in the upper 16 bits; the sign of the remainder is the same as dividend unless the remainder is zero. Division by zero causes a trap. Overflow may be detected and flagged but the operation is uneffected.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1 0 0 0	Register #	1 1 1	Effective Address Mode	Register

The register # specifies one of the eight data registers and this is the destination register. The effective address determines the source and all EA modes can be used except 2, 14.*

Mnemonic	Data Size/CCR	Function	Comments
DIVU	16 CCR X N Z V C - • • • 0	Unsigned Divide	The unsigned divide is identical to the signed divide except that unsigned arithmetic is used.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1 0 0 0	Register #	0 1 1	Effective Address Mode	Register

The addressing modes and the definition in the opword are the same as the signed divide.

Mnemonic	Data Size/CCR	Function	Comments
EXT	16, 32 CCR X N Z V C - • • 0 0	Sign Extend	This instruction extends the sign bit of a byte to a word or a word to a long word. The MSB is detected and extended to the proper length. The sign bit is considered to be the most significant bit of the word.

15 14 13 12 11 10 9	8 7 6	5 4 3	2 1 0
0 1 0 0 1 0 0	Op Mode	0 0 0	Register

Op Mode field specifies the size of the extension.
010 - Word sign extension
011 - Long word sign extension

Register field specifies one of eight data registers.

Mnemonic	Data Size/CCR	Function	Comments
MULS	16 CCR X N Z V C - • • 0 0	Signed Multiply	This operation multiplies two signed words together. The destination must be a specified data register. The sign of the operation is reflected in the sign bit.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1 1 0 0	Register	1 1 1	Effective Address Mode	Register

Register field specifies one of the data registers and is a destination register.

All effective addressing modes can be used except Direct, Quick, Immediate, and Implied 2, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
MULU	16 CCR X N Z V C - • • 0 0	Unsigned Multiply	This operation multiplies two 16-bit integers together and leaves the result in the destination register. The operations are similar to the signed multiply except that signed arithmetic is not used. The 32-bit result is left in the destination register.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1 1 0 0	Register	0 1 1	Effective Address Mode	Register

Register field specifies one of eight data registers.

The effective address can be anything but Direct, Quick, Immediate, and Implied 2, 13, 14.*

Table 1 *(continued)*

Mnemonic	Data Size/CCR	Function	Comments
NEG	8, 16, 32 CCR X N Z V C * * * * *	Negate	The destination is subtracted from zero. This changes the sign of the destination. The result of this operation is left in the destination. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 0 1 0 0 | Size | Effective Address Mode | Register Size field specifies the size of the data to be operated on. 00 - Byte operation 01 - Word operation 10 - Long word operation Effective address modes can by anything but 2, 10, 11, 12, 13, 14.*
NEGX	8, 16, 32 CCR X N Z V C * * * * *	Negate with Extend	The destination is subtracted from zero and the result of the operation is left in the destination. The sign bit is extended to the end of the word. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 1 0 0 0 0 0 0 | Size | Effective Address Mode | Register Size field specifies the size of the operation. 00 - Byte operation 01 - Word operation 10 - Long word operation The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*
SUB	8, 16, 32 CCR X N Z V C * * * * *	Subtract Binary	This operation subtracts the source from the destination and leaves the result in the destination. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 1 0 0 1 | Register | Op Mode | Effective Address Mode | Register Register field specifies any one of the eight data registers. Op Mode field defines the way that the operation is to be performed between the data register and the effective address. Byte / Word / Long Word / Operation 000 / 001 / 010 / The data register is the destination and the EA is subtracted from the Register. 100 / 101 / 110 / The EA is the destination and the register is subtracted from the EIA. The only effective address (EA) modes which cannot be used if the EA is a source are 13, 14.* If the EA is a destination then the following effective address modes cannot be used: 1, 2, 10, 11, 12, 13, 14.*
SUBA	16, 32 CCR X N Z V C - - - - -	Subtract Address	This instruction subtracts the effective address from the address register and leaves the result in the address register. 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 1 0 0 1 | Register | Op Mode | Effective Address Mode | Register Register field specifies any one of the eight address registers. The Op Mode specifies the size of the operation. 011 - Word operation 111 - Long word operation All effective address modes can be used except 10, 11, 12, 13, 14.*

(continued)

MICRObits *(continued)*

VisiCalc To Apple Plot

Interface translates from VisiCalc to Apple Plot, prevents erroneous graphs, fits curves to data, and supplements VisiCalc with rank ordering and alphabetizing. Send SASE for details or $30.00 for the copyable program.

Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

VisiCalc To Apple Writer

Veecee-Writer translated VisiCalc (/PF) files for Apple Writer 1. Send $15.00 for the copyable program.

Bill Starbuck
2100 E. Edgewood
Shorewood, WI 53211
(414) 963-9750

AIM-65 FIG-FORTH

Fig-Forth for your Aim-65 with assembler needs 16K memory. Cassette $20.00. With editor and assembler needs 32K memory $25.00.

D. Holmes
466 Palos Verdes Blvd.
Redondo Beach, CA 90277

OSI-C3B

152K static ram, 70 MB disk, 160 1pm printer, dual 8" floppies, software, 1 year old, used 2 months. $12,800.00 US or best offer.

Alan J. Lawson
Toronto, Canada
(416) 576-6508

OSI C1P Chomper

Positively one of the most difficult, challenging, and fun dot-eating games available on the C1P. Progressively faster action, bonus characters, and "commercials" for high scores. Keyboard or joystick, 8K tape. $14.95 includes shipping.

Watts Ware
153 Madrona Drive
Anacortes, WA 98221

Consulting Opportunities

Learn how to become a successful consultant in your own field. Write for a free prospectus:

The Consultant's Library
815 15th Street, NW Dept. M.
Washington D.C. 20005

MICRO

Mnemonic	Data Size/CCR	Function	Comments
SUBI	8, 16, 32 CCR X N Z V C • • • • •	Subtract Immediate	This instruction subtracts the immediate data from the destination. The result of the operation is left in the destination and the proper bits are set in the CCR.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address Mode			Register		
Word data (16 bits)								Byte data (8 bits)							
Long word data (32 bits using previous word)															

Size field defines the size of the operation.
- 00 - Byte operation data is the lower order byte of the immediate word.
- 01 - Word operation data is the entire immediate word.
- 10 - Long word operation data is the next two immediate words.

The following addressing modes cannot be used: 2, 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
SUBQ	8, 16, 32 CCR X N Z V C • • • • •	Subtract Quick	This operation subtracts the immediate data from the destination. The results are left in the destination and the bits are set in the CCR consistent with the results of the operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			1	Size		Effective Address Mode			Register		

Data field contains the immediate data to be subtracted: Any integer from 1 - 7 can be represented and 0 in the data field represents the integer 8.

Size field determines the size of the operation.
- 00 - Byte operation
- 01 - Word operation
- 10 - Long word operation

The following effective address modes cannot be used: 10, 11, 12, 13, 14.*

Mnemonic	Data Size/CCR	Function	Comments
SUBX	8, 16, 32 CCR X N Z V C • • • • •	Subtract with Extension	This instruction subtracts the source from the destination and leaves the results in the destination. The sign is extended.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register Rx			1	Size		0	0	R/M	Register Ry		

Size field specifies the size of the operation.
- 00 - Byte operation
- 01 - Word operation
- 10 - Long word operation

	R/M = 0	R/M = 1
Rx destination register	data register	address register for predecrement mode
Ry source register	data register	address register for predecrement mode
	data register to data register transfer	memory to memory transfer

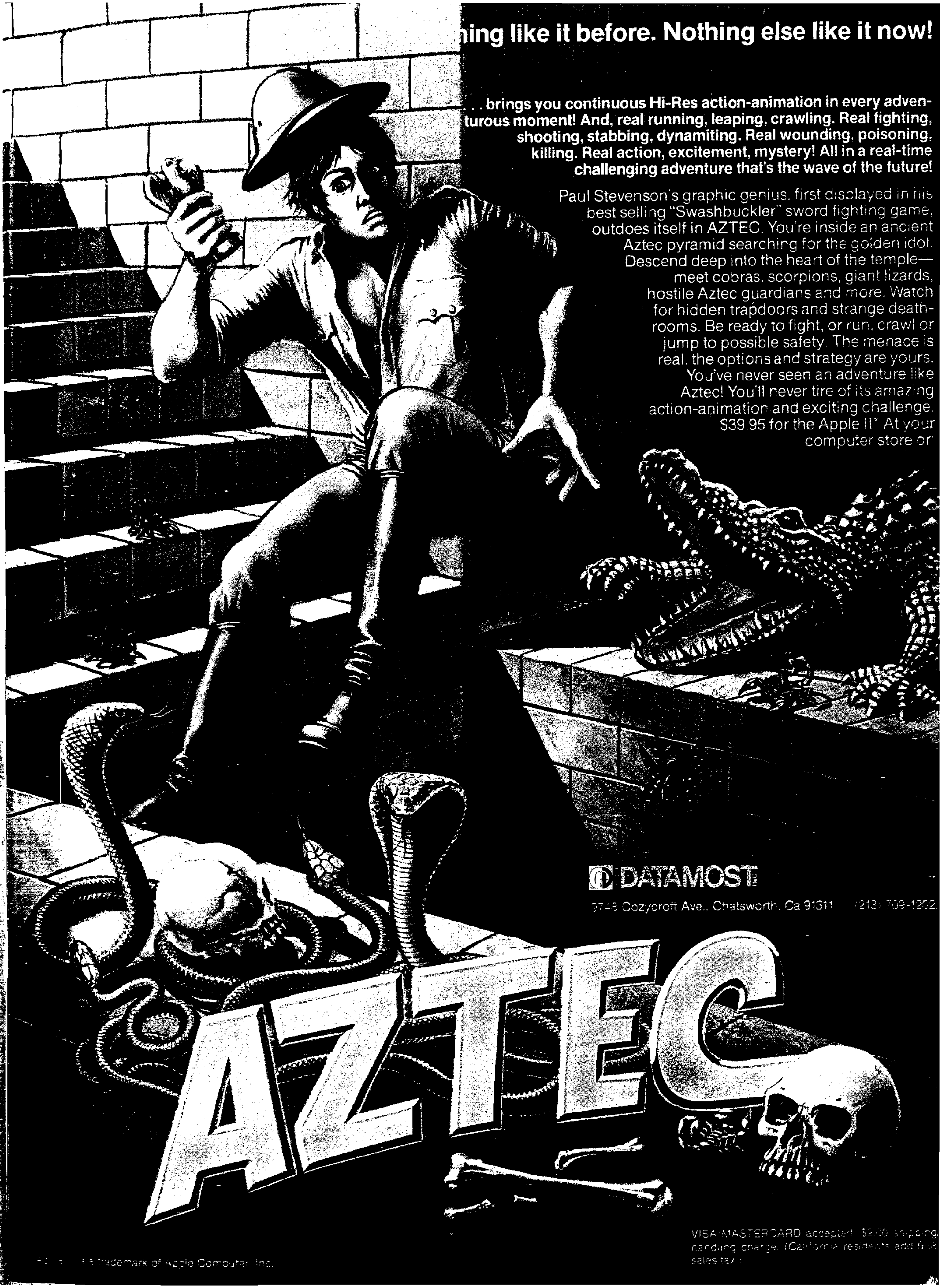
Mnemonic	Data Size/CCR	Function	Comments
TST	8, 16, 32 CCR X N Z V C - • • 0 0	Test an Operand	This instruction compares the data defined by the effective address with zero. The condition code register is set to be consistent with the result of the operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address Mode			Register		

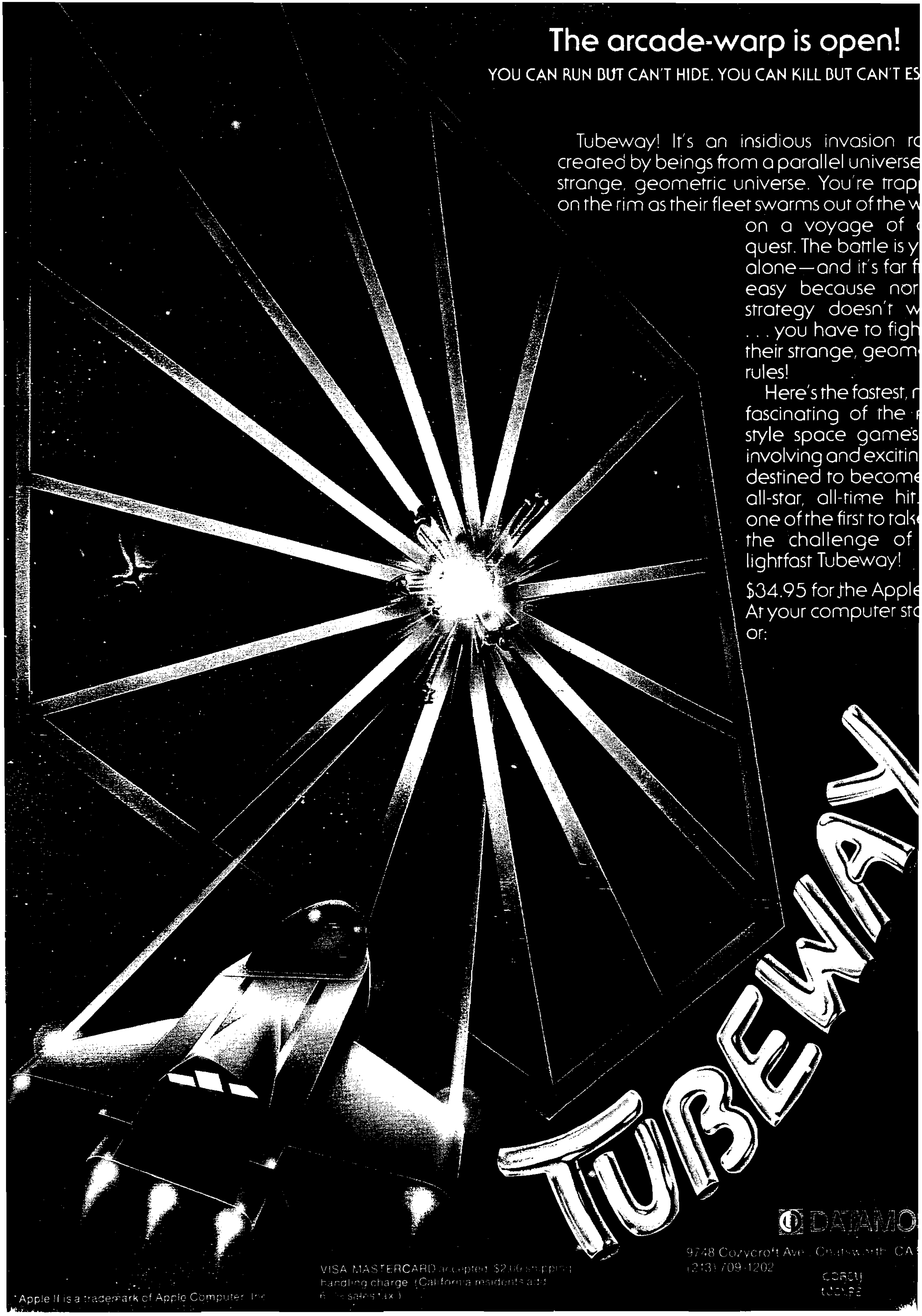
- 00 - Byte operation
- 01 - Word operation
- 10 - Long word operation

The following effective address modes cannot be used: 2, 10, 11, 12, 13, 14.*

MICRO



ing like it before. Nothing else like it now!
. . . brings you continuous Hi-Res action-animation in every adventurous moment! And, real running, leaping, crawling. Real fighting, shooting, stabbing, dynamiting. Real wounding, poisoning, killing. Real action, excitement, mystery! All in a real-time challenging adventure that's the wave of the future!
Paul Stevenson's graphic genius, first displayed in his best selling "Swashbuckler" sword fighting game, outdoes itself in AZTEC. You're inside an ancient Aztec pyramid searching for the golden idol. Descend deep into the heart of the temple—meet cobras, scorpions, giant lizards, hostile Aztec guardians and more. Watch for hidden trapdoors and strange death-rooms. Be ready to fight, or run, crawl or jump to possible safety. The menace is real, the options and strategy are yours. You've never seen an adventure like Aztec! You'll never tire of its amazing action-animation and exciting challenge. $39.95 for the Apple II.* At your computer store or:
DATAMOST INC.
Cozycroft Ave., Chatsworth, Ca 91311
AZTEC
VISA/MASTERCARD accepted.
is a trademark of Apple Computer, Inc.


The arcade-warp is open!
YOU CAN RUN BUT CAN'T HIDE. YOU CAN KILL BUT CAN'T ES
Tubeway! It's an insidious invasion r
created by beings from a parallel universe
strange, geometric universe. You're trap
on the rim as their fleet swarms out of the
on a voyage of
quest. The battle is y
alone—and it's far f
easy because nor
strategy doesn't w
... you have to figh
their strange, geom
rules!
Here's the fastest,
fascinating of the
style space games
involving and excitin
destined to become
all-star, all-time hit
one of the first to tak
the challenge of
lightfast Tubeway!
$34.95 for the Apple
At your computer st
or:
TUBEWAY
DATAMO
9748 Cozycroft Ave., Chatsworth, CA
(213) 709-1202
VISA MASTERCARD accepted. $2.00 shipping
handling charge. (California residents add
6% sales tax.)
*Apple II is a trademark of Apple Computer, Inc.

How to Make a Graphic-80 PET from a 4016

by James Strasma

A "Graphic 80" is an 80-column PET with the graphic-style keyboard. It can be made from a 4016 (Commodore's cheapest PET) by adding some inexpensive, readily available ICs and moving some jumpers. The author provides step-by-step instructions. In addition, instructions are provided to add extra keys to an 8032 or a Graphic 80.

The CBM 8032 offers an 80-column screen and a very business-oriented keyboard. Many PET owners would like to have the 80 columns, yet still maintain the easy access to graphic characters that the "graphic" keyboard offers. This does take some soldering and electronic assembly skill.

This article shows how to upgrade a "Fat Forty" 4016 or 4032 to a graphic keyboard, 80-column machine. There are several options available, including only upgrading to 32K, making a business-keyboard 8032, and adding extra keys to control video functions not previously accessible from a single key.

Before you begin, note that these changes are for ASSY. NO. 8032089, located on the right edge of the board, halfway back, the FCC-approved Universal Dynamic PET main board. Similar changes worked with earlier boards, as long as the computer came with the large, 12-inch screen.

Also note that procedures described below void *any Commodore warrantee, guarantees non-support by them,* and *cannot be guaranteed to work on your particular machine.*

The traces on current PET computer boards are very tiny, and easily destroyed. *Do not attempt this project unless you are skilled with a soldering iron.* Before you start, unplug your machine for your safety and the computer's.

Install sockets where new chips are added. This makes the job easier and makes later repairs more convenient.

To make a 4032 from a 4016

☐ Insert 4116 dynamic RAM chips in the vacant positions of column UA. These include UA4, UA6, UA8, UA10, UA12, UA14, UA16, and UA18, eight in all.

☐ At the right front of the board, find the right end of jumper Y, and reroute that end to the next hole toward the rear, at the right end of a line labeled Z. (See figure 1.) *Your PET will now display "31743 bytes free" when it is powered on. If it doesn't, check your connections.*

To make an 80-column machine from a 4032

☐ Move the right end of all ten jumpers at BA0 one hole to the rear. (See figure 2.)

☐ Just behind jumpers Y and Z, move the right end of the jumper labeled both *3 and 40* one hole to the rear, the right end of the line marked *4* and *80*. (See figure 1.)

☐ Remove the short between pins 10 and 11 of UD2 from the bottom of the board, and repair those pins' connections on the top side of the board. If you damage this chip, it can be inexpensively replaced. (UD2 is the chip in the upper-left of figure 1.)

☐ Remove the jumpers at *6* and *7* between UB2 and UC3. (See figure 3.)

☐ Add jumpers at *5* and at *8* in the same area (figure 3). The jumper at *8* shoud be between the most widely separated of its four holes.

☐ Add 2114 static RAM chips at UC6 and UC7.

☐ Add 74LS244 octal tri-state driver chips at UB6 and UB7.

☐ Add a 74LS373 tri-state octal D flip-flop chip at UB8.

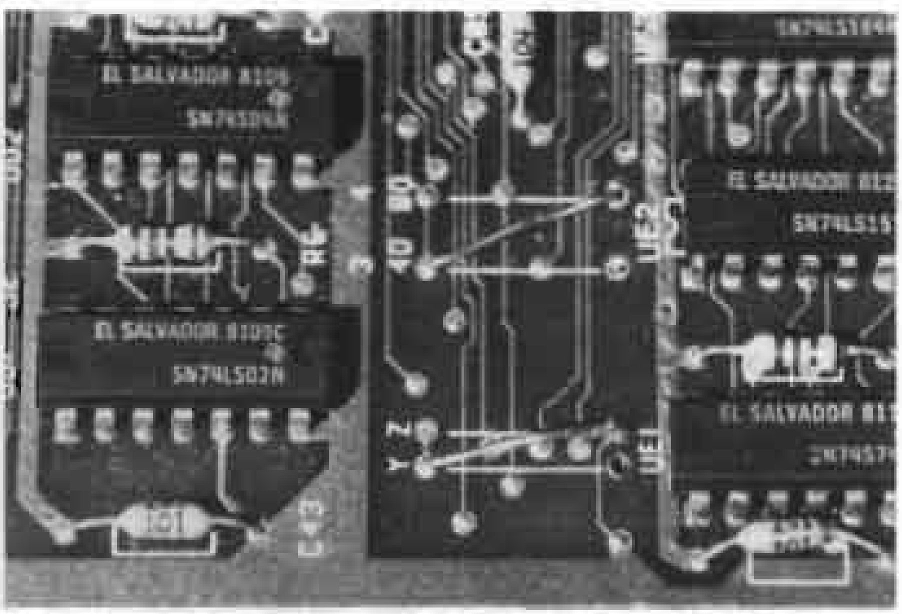
☐ Replace the 2K ROM at UD7 with a suitable replacement, as described in the next section.

To make the result a business-keyboard 8032

☐ Buy or program a ROM for UD7 identical to the one supplied in that position on the 8032. Commodore's part number is 901474-03. The correct EPROM substitute is a single 5-volt supply 2716.

☐ Buy and install a business-keyboard in place of the graphic one

Figure 1: Detail of front of board showing positions of jumpers Y, Z, 3, and 4, and IC UD2.



supplied. Since the cutouts will not match, the lower portion of the hood should be replaced, or you can make your own mounting plate.

To make the result a G80

☐ Create a custom 2716 EPROM for UD7, or obtain one from a user group. The only change is to take the 80-byte keyboard look-up table from locations $E798-$E7E7 in the Fat Forty ROM, and copy it into the functionally equivalent location in the 8032's ROM, starting at $E6D1. With this change, the G80 will lack only the REPEAT, ESCAPE, and TAB keys of the 8032.

However, as long as you have made the decision to modify the ROM, why not improve it? By changing only a few bytes, you may add not only the missing keys, but also up to four others.

Listing 1 shows my keyboard look-up table, which includes keys for TEXT/GRAPHICS, INSERT/DELETE LINE, ERASE TO BEGIN/END, and an optional value for SET TOP/BOTTOM OF SCREEN. The details are described later in the article.

Adjusting the Screen

There is a slight problem with the video adjustment that appears when CHR$(14) is printed. The top and bottom lines disappear! To fix this, adjust the potentiometer labeled HEIGHT, from below the video display board, using a small non-conductive screwdriver. Bear in mind that parts of this board carry over 10,000 volts, even when the computer is unplugged!

Using the program below, adjust the pot so the test pattern just fills the screen:

```
10 PRINT CHR$(14)
20 FOR I=1 TO 1999
30 : PRINT "#";
40 NEXT
50 GOTO 50
```

The same fix works on any Fat Forty.

To add missing and extra keys pull off the keyboard connector at the keyboard. Note that it has 18 separate connections. These correspond to the ten rows and eight columns of the keyboard matrix as shown in listing 1.

There is a small hole at each position in this connector. Using micro test clips available from Radio Shack, you can make temporary connections.

For a slightly more permanent attachment, I soldered tiny loops of wire to the keyboard side of the connector, as shown in figure 4. From the right edge of the keyboard connector, as viewed in place, the ten rows of the keyboard matrix are the first ten wires from the right. The eight columns of

Listing 1

.: E6D1	3D	2E	10	03	3C	20	5B	12
.: E6D9	2D	30	00	3E	19	5D	40	00
.: E6E1	2B	32	0E	3F	2C	4E	56	58
.: E6E9	33	31	0D	3B	4D	42	43	5A
.: E6F1	2A	35	1B	3A	4B	48	46	53
.: E6F9	36	34	15	4C	4A	47	44	41
.: E701	2F	38	16	50	49	59	52	57
.: E709	39	37	5E	4F	55	54	45	51
.: E711	14	11	09	29	5C	27	24	22
.: E719	1D	13	5F	28	26	25	23	21

Figure 2: Detail showing jumpers BA0 - BA10 and position of replacement EPROM at UD7.

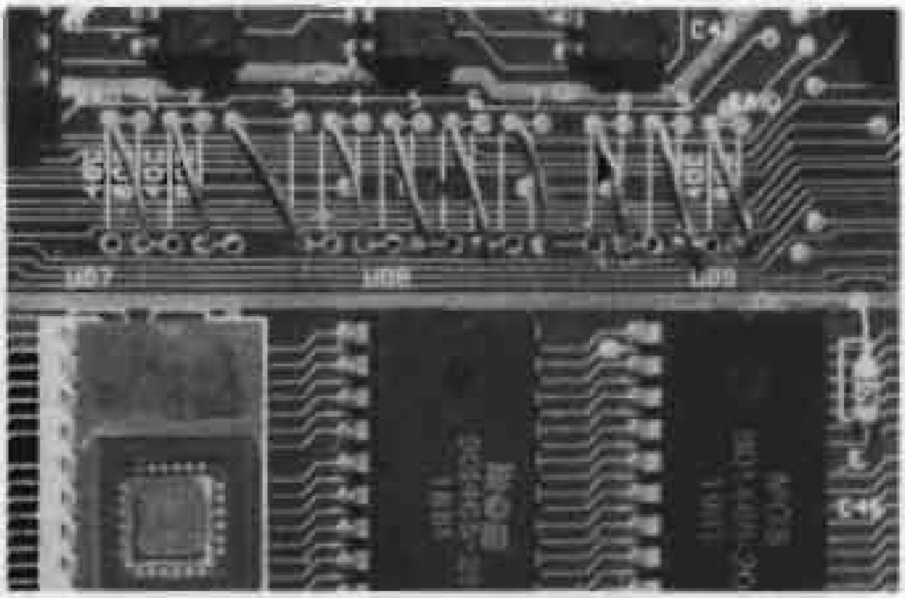
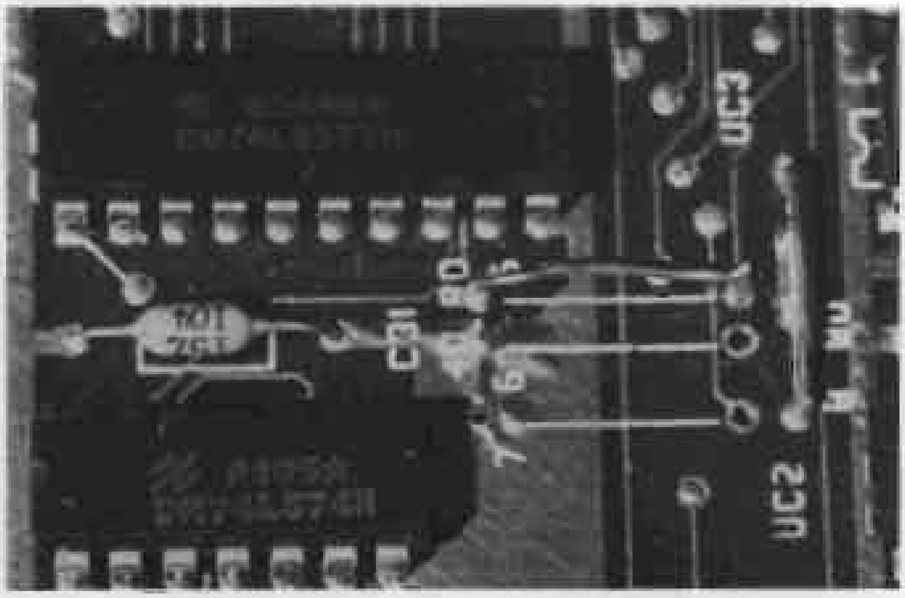


Figure 3: Detail showing jumpers 5, 6, 7, and 8.



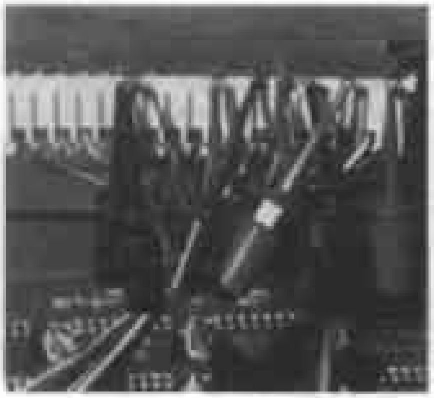


Figure 4: Added loops at keyboard connector and clip connections.

Figure 5: Micro push buttons added near screen. Three additional buttons were installed on the other side.

the matrix follow, again toward the left.

The six keys I have added are all in column 3 of listing 1. This corresponds to the thirteenth wire from the right. It is the one with two micro test clips in figure 4. Each of these wires connects to one pole of an added key (or push button, as I implemented it).

Rows 1, 3, 5, 6, 7, and 9 contain the added keys. Connect each one to the unused pole of the appropriate added "key." One more key could be added by attaching wires to row 2 and column 5. These correspond to positions 2 and 15 from the right end of the connector, and would be mated through another key.

If you prefer to implement another keyboard value, substitute your preferred PET ASCII value into the keyboard look-up table of listing 1. For instance, if you replace the $03 at location $E6D4 with $07, then hitting the STOP key would ring the chime rather than halting running programs.

I chose to mount push buttons through the bezel surrounding the video screen, as shown in figure 5. I used Radio Shack's tiniest push buttons because they are unobtrusive, easier to push than large ones, and I couldn't find regular keyswitches. Regular keyswitches could be used, or a surplus keypad could be wired up.

One method for making the G80 selectable between 40 and 80 columns requires three ICs to switch the necessary lines. This plan offers a simpler solution. It uses a 2732 EPROM, preprogrammed to mimic any two BASIC 4.0 ROM sets, and switches from one to the other by grounding one pin.

The disadvantage of this method is that the 40 columns appear smaller and centered on an 80-column screen, rather than occupying the full width, as on a Fat Forty.

For information on how to obtain this chip, see the box on this page.

Software Compatibility

Nearly everything for the 8032 also works on the G80, especially after adding the missing keys. This includes the 8096 memory expansion board, Silicon Office, COMAL, and VisiCalc 8096. The exceptions are complete languages, including UCSD Pascal, the former PET BASICs supplied with the 8096 board, and A.B. Computer's "Expanded BASIC" for the 8096. By modifying the programs, I have been able to get all but UCSD Pascal to work with the G80.

You may contact the author at 1280 Richland Ave., Lincoln, IL 62656.

MICRO

Obtaining Alternate ROMs

To order alternate ROMs for Commodore 8032 and G80 computers, write:

Jim Russo
Ann Arbor Terminals, Inc.
6175 Jackson Road
Ann Arbor, MI 48103

You may request any two of the following variations of BASIC 4:

Fat Forty (Current model, but centered on an 80-column screen.)
Skinny Forty (Also centered; works with far more games. No repeat function.)
Eighty (Same as the 8032.)

Be sure to specify whether the ROM is for a G80 or an 8032. I think the charge is still $10 per ROM, barely above the cost of the 2732 itself.

MICRO™

CoCo Bits

By John Steiner

Ed. note: This is the first of our Color Computer columns. John will be augmenting our coverage of 6809 computers and helping to recruit authors.

This column will have two major goals: to provide news about the TRS-80 Color Computer, and to provide a clearinghouse for CoCo information. In addition, I hope to pass along any information on the 6809, the 6847 video display generator, and the other major components CoCo has.

Originally, it seems, Tandy developed CoCo to compete with Atari, the VIC 20, and other game machines of similar style. Witness the use of ROM program packs for games, etc. This emphasis seems to be changing slightly. Just recently, Tandy has taken the 4K machine off the market and replaced it with a 16K version, at the same price. With the addition of the 32K upgrade (which uses 64K chips), CoCo has entered a new world. The newer, more powerful machine has been easily converted to 64K and given the capacity to run FLEX. OS-9 will probably be available by the time you read this. These two powerful operating systems allow a vast range of 6809 software to be executed from CoCo. As I learn more details of these CoCo expansions, I will pass them along.

Since this is the games issue, I have some game-oriented information this month. If you have a Color Computer disk system, you have no doubt been frustrated by the fact that the CoCo DOS scratchpad is located at $600, just where many machine-language tape programs like to reside. I am grateful that Tandy has started releasing some software on tape since disk users have trouble getting the disk drive and a ROM pack in the ROMport at the same time. It is a shame, though, to have to spend several minutes loading a 14K adventure game from tape, when that expensive disk system just sits there taking up space.

I purchased "BEDLAM," an adventure game on tape, and decided I would have to transfer it to disk. Being a relative beginner to machine-language programming and only having experience with the 6800, I have been waiting for Tandy's assembler (though others became available, I had a deposit on Tandy's). When it finally arrived, I plugged in the EDTASM + ROM pack (doggone it, there went the disk again), and loaded BEDLAM using an offset of 16384. Using the monitor, I found the start and end addresses, and tacked a little routine at the end. This routine, shown below, moves the program, one byte at a time (I forgot about the D register) down to $600. It then transfers execution to $600, the start address of BEDLAM.

You may enter and assemble the listing yourself, or enter the object code from the assembler listing using a monitor. Once the routine is in place, beginning at $7F02, use CLOADM "BEDLAM",16384 to load in BEDLAM. Transfer the whole thing to disk with the command SAVEM "BEDLAM", &H4600, &H7F15, &H7F02. Notice the execute address is the start address of the memory move routine. To run the program, you can just use LOADM "BEDLAM" : EXEC, or you can write a BASIC load routine, and let BASIC do the work for you with a simple RUN "BEDLAM".

Once the program has been loaded into the region at $600, disk BASIC is essentially gone. The quick and dirty method to restore DOS is to shut the computer off, then turn it on again. A better way is to let RESET do it for you. CoCo BASIC has two options in its reset routine: a cold start and a warm start. When RESET is pressed, memory location 113 is checked for $55. If found, a warm start is done, any program in memory is saved; string memory, number of graphics pages reserved, and other parameters are kept. If, on the other hand, anything but $55 appears, BASIC assumes a cold start and reconfigures the system to power up status. This little trick will recover our disk when we are through with BEDLAM. Before loading the program, from BASIC enter POKE 113,0. Alternatively, you could add a routine at the beginning of the machine-language loader that will store a zero into location 113 before executing BEDLAM. When you are ready to quit BEDLAM, just press RESET. DOS will be reconnected. You will not be missing any fancy end routine by leaving BEDLAM in this manner.

A quirk of this and the other Radio Shack adventure games I've seen is their STOP or QUIT command. I would have expected control to return to BASIC. What happens is that the keyboard will lock up, causing you to turn off the computer or press RESET to regain control. As long as you have to press RESET anyway, you might as well reconnect the disk. Though not particularly fast or fancy, you can use the routine on any 32K machine to transfer programs. Just substitute the correct start, end, and execute addresses where required.

```
                *
                          ORG    $7F02
                *PROGRAM TO MOVE BEDLAM*
                *JOHN STEINER 8/1/82*
7F02 8E    4600  START    LDX    #$4600    Load current start address
7F05 108E  0600           LDY    #$600     Load new start address
7F09 A6    80    MOVE     LDA    ,X+       Load byte from current address
7F0B A7    A0             STA    ,Y+       Store byte to new address
7F0D 8C    7F01           CMPX   #$7F01    Done yet?
7F10 26    F7             BNE    MOVE      If not go do it again
7F12 BD    0600           JSR    $600
                          END
```

CoCo Bits *(continued)*

One more programming hint for this month. Disk users are told by the disk system manual that "COPY" is available for only multi-drive users. This is incorrect. If you have a single disk drive, you can enter COPY "filename/ext". You will be prompted as to when to switch disks. In addition, unlike BACKUP and DSKINI, COPY is non-destructive of the program in memory. If you have a long program in memory and a large file to copy, however, you may have to switch disks a couple of times. COPY can be used to transfer any file type, and easily transfers machine-language or data files. I will have more details on COPY next month, including a small routine that assists in selective disk backups.

I am an electronics instructor by profession, and would like to make a couple of comments on CoCo in school. Tandy is developing courseware that runs on the Color Computer. CoCo will be making appearances in classrooms around the country if Tandy has anything to say about it. By the time you read this, teachers will have access to programs such as Chemistry Simulations. Radio Shack's Talk/Tutor development system is the medium for several recently released educational packages, including Vocabulary Tutor I, and Inventions That Changed Our Lives. Talk/Tutor programs make good use of the high-resolution graphics and audio features of CoCo.

In addition to these and other educational software, Color PILOT will have been released. PILOT has been used by many instructors who wanted to develop computer-assisted instruction, yet did not want to learn the complexities of BASIC. Color LOGO, another popular language with educators, should also be available. Both programs will have disk versions; LOGO will also be on ROM pack.

I am looking forward to comments from readers. I plan on including programming hints, and CoCo- and 6809-related news. In addition, CoCo disk users are probably a distinct minority and I will have information on Color BASIC and Extended BASIC, as well as disk BASIC. I can be reached at 508 Fourth Ave. NW, Riverside, ND 58078, or contact me through MICRO.

MICRO

Castle Adventure for PET and Apple

by David Malmberg

Castle Adventure is a role-playing game that involves traveling throughout a dangerous castle in search of treasure. The goal is to rescue the princess, while avoiding the many dangers about you.

Castle Adventure
requires:
PET/CBM with 32K
or Apple II with 48K
or other Microsoft BASIC computer

In CASTLE ADVENTURE you play the role of Godfrey de Goodheart, a bold, but impoverished knight. King Fredrick III has dispatched you to rescue his only daughter, the beautiful Princess Fatima, from the dungeons of Baron von Evil's castle. You have also been asked to capture the Baron's treasures of gold, silver, and gems, which he enmassed by cruelly exploiting his serfs. If you can rescue the princess and return with all of the Baron's ill-gotten treasures, King Fredrick has promised you Princess Fatima's hand in marriage.

Your quest will be filled with peril. The seven lone knights who were previously sent on this crusade all vanished without a trace. If you are to succeed where so many others have failed, you must use all your strength and cunning — and be very lucky!

During your quest the computer will be your guide. You take action by giving the computer a series of one- or two-word commands, such as: GO SOUTH (or just "S"), OPEN DOOR, GET KEYS, LEAVE CHEST, SWIM. The computer has a vocabulary of only about 100 words. If it does not understand your command, try something else. A complete list of this vocabulary is purposely not included. At least half the fun will be establishing the computer's lexicon. Several commands will be particularly useful:

- INVENTORY (or just "I") will give you a list of all the items you are carrying.
- LOOK (or just "L") may reveal significant details that may help you in your quest.
- HELP (or just "H") may result in a valuable hint.
- SAVE will cause the current status of the game to be saved on tape or disk.
- LOAD will enable you to resume a previously saved game.
- SCORE will show you the total number of points you have earned so far.
- QUIT will end the game and show you your final score.

Remember that everything you encounter in your adventure has a purpose. There are plenty of clues, but it will take imagination, perseverance, cunning, skill, and most of all luck to win the Princess's hand.

Converting to Other Computers

CASTLE ADVENTURE, as shown in listing 1, is written for a 32K PET/CBM computer. Listing 2 shows changes required for an Apple II. However, the program is written in "standard" Microsoft BASIC, so conversion to other Microsoft machines should be a relatively trivial task. There are only three areas where changes to the program will have to be made.

First, you will have to replace the screen and cursor control commands of the PET. These are shown in the listing within square brackets in their "English equivalents" so their meaning should be fairly obvious; i.e., CLR means clear the screen, 3 DOWN means move the cursor down three rows, etc. CASTLE ADVENTURE is written for a 40-column screen, so no spacing changes will be required for the Atari (with Microsoft BASIC) or other 40-column systems.

The second change is to convert the LOAD game (lines 24 to 36) and SAVE game (lines 219 to 233) routines so they will be compatible with your machine's tape and/or disk command formats. The variables you want to SAVE and LOAD are: SF, LX, DF, R, and the array IA(.), which has IL elements (including a zero-th element).

The last thing to change is the reference to the PET ROMs in line 390.

Acknowledgement

Many of the ideas in CASTLE ADVENTURE, as well as other adventures that are widely available, owe a tremendous debt to Scott Adams. In the specific case of CASTLE, it uses a data-base structure and table-driven logic similar to those first described by Adams in several articles. These articles are a *must* for the true adventurephile:

1."An Adventure in Small Computer Game Simulation," *Creative Computing*, (August 1979). Describes the data-base structure.
2."Adventureland," *Softside*, (July 1980). Describes the table-driven logic.
3."Pirate's Adventure," *BYTE*, (December 1980). Also describes the table-driven logic.

Castle Adventure listing begins on page 42. The changes for Apple II (listing 2) are on page 46.

David Malmberg is the author or co-author of several personal computer packages; the most recent is *VIC Turtle Graphics* published by Human Engineered Software. You may contact him at 43064 Via Moraga, Fremont, CA 94539.

Listing 1: Castle Adventure for PET/CBM

```
1 PRINT "[CLR][ 4 DOWN]      WELCOME TO CASTLE ADVENTURE"
2 PRINT "[DOWN]          BY DAVID MALMBERG"
3 PRINT "[ 3 DOWN]  BASED ON THE IDEAS, DATA STRUCTURE"
4 PRINT "  AND DRIVER PROGRAM OF SCOTT ADAMS"
5 PRINT "[ 3 DOWN]READING DATA TABLES -- JUST A MOMENT"
6 REM DATA STRUCTURE EXPLAINED IN CREATIVE COMPUTING AUGUST 1979
7 REM TRS-80 VERSION OF DRIVER PROGRAM GIVEN IN BYTE DECEMBER 1980
8 REM AND IN SOFTSIDE JULY 1980
9 GOSUB 235:TQ=0:TD=0
10 FOR X=1 TO NL:IF NV$(X,0)="GET" THEN TQ=X
11 IF NV$(X,0)="DRO" THEN TD=X
12 NEXT X
13 DIM L1%(NL),U1%(NL)
14 FOR X=0 TO NL:L1%(X)=CL:U1%(X)=0:NEXT X
15 FOR X=0 TO CL:V=INT(C0%(X)/150)
16 IF X>U1%(V)THEN U1%(V)=X
17 IF X<L1%(V)THEN L1%(V)=X
18 NEXT X
19 FOR X=0 TO NL:IF L1%(X)=CL AND U1%(X)=0 THEN U1%(X)=CL
20 NEXT X
21 PRINT "[CLR]"
22 R=AR:LX=LT:DF=0:SF=0:INPUT "USE SAVED GAME (Y OR N)   N[ 3 LEFT]";K
   $
23 IF LEFT$(K$,1)<>"Y" THEN PRINT "[CLR]":GOTO 37
24 INPUT "[CLR][RVS]T[OFF]APE OR [RVS]D[OFF]ISK  D[ 3 LEFT]";K$
25 IF K$<>"T" AND K$<>"D" THEN 24
26 IF K$="T" THEN 32
27 INPUT "DRIVE [RVS]0[OFF] OR [RVS]1[OFF]  0[ 3 LEFT]";K$
28 IF K$<>"0" AND K$<>"1" THEN 27
29 DN$=CHR$(34)+K$+":"
30 INPUT "[DOWN]TITLE  *[ 3 LEFT]";LT$:IF LT$="*" THEN 30
31 OPEN 1,8,0,DN$+LT$+",S,R":GOTO 34
32 INPUT "IS SAVED GAME TAPE POSITIONED";K$:IF LEFT$(K$,1)<>"Y" THEN 2
   2
33 OPEN 1,1,0,"ADVENTURE GAME"
34 INPUT#1,SF,LX,DF,R
35 FOR X=0 TO IL:INPUT#1,IA(X):NEXT X
36 CLOSE 1
37 GOSUB 66:GOTO 44
38 PRINT:INPUT "WHAT DO YOU WANT TO DO  ▚[ 3 LEFT]";TP$:PRINT:GOSUB 45
39 IF F THEN PRINT "YOU USE WORD(S) I DON'T KNOW!":GOTO 38
40 GOSUB 82:IF IA(9)=-1 THEN LX=LX-1:GOTO 42
41 GOTO 44
42 IF LX<0 THEN PRINT "LIGHT HAS RUN OUT!":IA(9)=0:GOTO 44
43 IF LX<25 THEN PRINT "LIGHT RUNS OUT IN";LX;"TURNS!"
44 NV(0)=0:GOSUB 82:GOTO 38
45 IF LEN(TP$)>1 THEN 55
46 IF TP$="U" THEN TP$="GO UP":GOTO 55
47 IF TP$="D" THEN TP$="GO DOWN":GOTO 55
48 IF TP$="E" THEN TP$="GO EAST":GOTO 55
49 IF TP$="W" THEN TP$="GO WEST":GOTO 55
50 IF TP$="N" THEN TP$="GO NORTH":GOTO 55
51 IF TP$="S" THEN TP$="GO SOUTH":GOTO 55
52 IF TP$="I" THEN TP$="INVENTORY":GOTO 55
53 IF TP$="L" THEN TP$="LOOK":GOTO 55
54 IF TP$="H" THEN TP$="HELP"
```

Listing 1 *(continued)*

```
55 K=0:NT$(0)="":NT$(1)="":AA=0
56 FOR X=1 TO LEN(TP$):K$=MID$(TP$,X,1):IF K$=" " THEN K=1:GOTO 58
57 NT$(K)=LEFT$(NT$(K)+K$,LN)
58 NEXT X:FOR X=0 TO 1:NV(X)=0:IF NT$(X)="" THEN 64
59 FOR Y=0 TO NL:K$=NV$(Y,X):IF LEFT$(K$,1)="*" THEN K$=MID$(K$,2)
60 IF X=1 AND Y<7 THEN K$=LEFT$(K$,LN)
61 IF NT$(X)=K$ THEN NV(X)=Y:GOTO 63
62 NEXT Y:GOTO 64
63 IF LEFT$(NV$(NV(X),X),1)="*" THEN NV(X)=NV(X)-1:GOTO 63
64 NEXT X:IF(NV(0)=TQ OR NV(0)=TD)AND NV(1)<1 THEN AA=1:F=0:RETURN
65 F=NV(0)<1 OR LEN(NT$(1))>0 AND NV(1)<1:RETURN
66 IF DF THEN IF IA(9)<>-1 AND IA(9)<>R THEN PRINT "IT IS TOO DARK TO
   SEE!":RETURN
67 K=-1:IF LEFT$(RS$(R),1)="*" THEN PRINT MID$(RS$(R),2);:GOTO 69
68 PRINT "YOU ARE IN ";RS$(R);
69 FOR Z=0 TO IL:IF K THEN IF IA(Z)=R THEN PRINT:PRINT "[DOWN]VISIBLE
   ITEMS HERE:":K=0
70 GOTO 75
71 TP$=IA$(Z):IF RIGHT$(TP$,1)<>"/" THEN RETURN
72 FOR W=LEN(TP$)-1 TO 1 STEP-1:IF MID$(TP$,W,1)="/" THEN TP$=LEFT$(TP
   $,W-1):RETURN
73 NEXT W
74 RETURN
75 IF IA(Z)<>R THEN 78
76 GOSUB 71:IF POS(0)+LEN(TP$)+3>39 THEN PRINT
77 PRINT TP$;".  ";
78 NEXT:PRINT
79 K=-1:FOR Z=0 TO 5:IF K THEN IF RM(R,Z)<>0 THEN PRINT:PRINT "OBVIOUS
   EXITS: ":K=0
80 IF RM(R,Z)<>0 THEN PRINT NV$(Z+1,1);" ";
81 NEXT:PRINT:PRINT:RETURN
82 F2=-1:F=-1:F3=0:IF NV(0)=1 AND NV(1)<7 THEN 134
83 L2=L1%(NV(0)):U2=U1%(NV(0))
84 IF NV(0)<>0 THEN 89
85 V=INT(RND(1)*100+1)
86 FOR X=L2 TO U2
87 IF V<=C0%(X)THEN 95
88 NEXT X:RETURN
89 IF AA=1 THEN GOSUB 199:RETURN
90 N=NV(1)+150*NV(0):V=150*NV(0):F4=0
91 IF NV(0)=TQ OR NV(0)=TD THEN F2=0
92 FOR X=L2 TO U2
93 IF N=C0%(X)OR V=C0%(X)THEN 95
94 NEXT X:GOTO 179
95 F4=1:F2=-1:F=0:F3=-1:FOR Y=1 TO 5:ON Y GOTO 96,97,98,99,100
96 W=C1%(X):GOTO 101
97 W=C2%(X):GOTO 101
98 W=C3%(X):GOTO 101
99 W=C4%(X):GOTO 101
100 W=C5%(X):GOTO 101
101 LL=INT(W/20):K=W-LL*20:F1=-1
102 ON K+1 GOTO 121,109,111,113,115,116,117,118,119,120,105,107
103 IF K<12 THEN 105
104 ON K-11 GOTO 110,112,114
105 F1=-1:FOR Z=0 TO IL:IF IA(Z)=-1 THEN 121
106 NEXT:F1=0:GOTO 121
```

Listing 1 *(continued)*

```
107 F1=0:FOR Z=0 TO IL:IF IA(Z)=-1 THEN 121
108 NEXT:F1=-1:GOTO 121
109 F1=IA(LL)=-1:GOTO 121
110 F1=IA(LL)<>-1 AND IA(LL)<>R:GOTO 121
111 F1=IA(LL)=R:GOTO 121
112 F1=IA(LL)<>0:GOTO 121
113 F1=IA(LL)=R OR IA(LL)=-1:GOTO 121
114 F1=IA(LL)=0:GOTO 121
115 F1=R=LL:GOTO 121
116 F1=IA(LL)<>R:GOTO 121
117 F1=IA(LL)<>-1:GOTO 121
118 F1=R<>LL:GOTO 121
119 F1=SF AND INT(2↑LL+.5):F1=F1<>0:GOTO 121
120 F1=SF AND INT(2↑LL+.5):F1=F1=0
121 F2=F2 AND F1:IF F2 THEN NEXT Y:GOTO 123
122 NEXT X:GOTO 179
123 IP=0:FOR Y=1 TO 4:K=INT((Y-1)/2+6):ON Y GOTO 124,125,126,127
124 AC=INT(C6%(X)/150):GOTO 128
125 AC=C6%(X)-INT(C6%(X)/150)*150:GOTO 128
126 AC=INT(C7%(X)/150):GOTO 128
127 AC=C7%(X)-INT(C7%(X)/150)*150
128 IF AC>101 THEN 133
129 IF AC=0 THEN 176
130 IF AC<52 THEN PRINT MS$(AC):GOTO 176
131 ON AC-51 GOTO 144,148,149,151,152,153,154,151,156,158,159
132 ON AC-62 GOTO 160,162,163,168,172,173,174,175,219,150
133 PRINT MS$(AC-50):GOTO 176
134 L=DF:IF L THEN L=DF AND IA(9)<>R AND IA(9)<>-1:GOTO 136
135 GOTO 137
136 IF L THEN PRINT "DANGEROUS IN THE DARK!"
137 IF NV(1)<1 THEN PRINT "GIVE ME A DIRECTION TOO.":GOTO 187
138 K=RM(R,NV(1)-1)
139 IF K>=1 THEN 142
140 IF L THEN PRINT "YOU FELL DOWN AND BROKE YOUR NECK.":K=RL:DF=0:GOTO
    142
141 PRINT "YOU CAN'T GO IN THAT DIRECTION!|":GOTO 187
142 IF NOT L THEN PRINT "[CLR]"
143 R=K:GOSUB 66:GOTO 187
144 L=0:FOR Z=1 TO IL:IF IA(Z)=-1 THEN L=L+1
145 NEXT Z
146 IF L>=MX THEN PRINT "YOU'VE TOO MUCH ALREADY!":GOTO 177
147 GOSUB 188:IA(P)=-1:GOTO 176
148 GOSUB 188:IA(P)=R:GOTO 176
149 GOSUB 188:R=P:GOTO 176
150 GOSUB 188:L=P:GOSUB 188:Z=IA(P):IA(P)=IA(L):IA(L)=Z:GOTO 176
151 GOSUB 188:IA(P)=0:GOTO 176
152 DF=-1:GOTO 176
153 DF=0:GOTO 176
154 GOSUB 188
155 SF=INT(.5+2↑P)OR SF:GOTO 176
156 GOSUB 188
157 SF=SF AND NOT INT(.5+2↑P):GOTO 176
158 PRINT "YOU'RE DEAD...":R=RL:DF=0:GOTO 162
159 GOSUB 188:L=P:GOSUB 188:IA(L)=P:GOTO 176
160 INPUT "THIS GAME IS NOW OVER.  ANOTHER GAME";K$:IF LEFT$(K$,1)="N"
    THEN END
```

Listing 1 *(continued)*

```
161 FOR X=0 TO IL:IA(X)=I2(X):NEXT:PRINT "[CLR]":GOTO 22
162 GOSUB 66:GOTO 176
163 L=0:FOR Z=1 TO IL:IF IA(Z)=TR THEN IF LEFT$(IA$(Z),1)="*" THEN L=L+
    1
164 NEXT Z:PRINT "YOU'VE STORED";L;"TREASURES. ON A SCALE"
165 PRINT "OF 0 TO 100 THAT RATES A";INT(L/TT*100)
166 IF L=TT THEN PRINT "WELL DONE. ":GOTO 160
167 GOTO 176
168 PRINT "YOU HAVE:":K$="NOTHING":FOR Z=0 TO IL:IF IA(Z)<>-1 THEN 171
169 GOSUB 71:IF LEN(TP$)+POS(0)>39 THEN PRINT
170 PRINT TP$;".",;:K$=""
171 NEXT:PRINT K$:GOTO 176
172 P=0:GOTO 155
173 P=0:GOTO 157
174 LX=LT:IA(9)=-1:GOTO 176
175 PRINT "[CLR]"
176 NEXT Y
177 IF NV(0)<>0 THEN 179
178 NEXT X
179 IF NV(0)=0 THEN 187
180 IF NV(0)<>TQ AND NV(0)<>TD AND F4=0 THEN PRINT "I DON'T UNDERSTAND
    YOUR COMMAND"
181 IF F2 THEN 187
182 IF NV(0)<>TQ AND NV(0)<>TD AND NOT F2 THEN PRINT "NOTHING HAPPENED"
    :RETURN
183 IF NV(0)=TQ THEN GOSUB 197
184 IF NV(0)=TD THEN GOSUB 197
185 IF F=0 THEN 187
186 IF NOT F2 THEN PRINT "YOU CAN'T DO THAT YET."
187 RETURN
188 IP=IP+1
189 ON IP GOTO 190,191,192,193,194
190 W=C1%(X):GOTO 195
191 W=C2%(X):GOTO 195
192 W=C3%(X):GOTO 195
193 W=C4%(X):GOTO 195
194 W=C5%(X):GOTO 195
195 P=INT(W/20):M=W-P*20:IF M<>0 THEN 188
196 RETURN
197 IF NV(0)=0 THEN 218
198 IF NV(1)=0 THEN PRINT "WHAT?":GOTO 211
199 IF NV(0)<>TQ THEN 202
200 L=0:FOR Z=0 TO IL:IF IA(Z)=-1 THEN L=L+1
201 NEXT:IF L>=MX THEN PRINT "YOU'VE TOO MUCH ALREADY!":GOTO 211
202 K=0:FOR X=0 TO IL:IF RIGHT$(IA$(X),1)<>"/" THEN 212
203 LL=LEN(IA$(X))-3:TP$=MID$(IA$(X),LL,3)
204 IF TP$<>NT$(1)THEN 212
205 IF NV(0)=TQ THEN 208
206 IF IA(X)<>-1 THEN K=1:GOTO 212
207 IA(X)=R:K=3:GOTO 210
208 IF IA(X)<>R THEN K=2:GOTO 212
209 IA(X)=-1:K=3
210 PRINT "OK":PRINT
211 F=0:RETURN
212 NEXT X
213 IF K=1 THEN PRINT "YOU DO NOT HAVE IT!"
214 IF K=2 THEN PRINT "I DON'T SEE IT HERE."
```

Listing 1 *(continued)*

```
215 IF K=0 AND NV(1)=0 THEN PRINT "IT'S BEYOND YOUR POWER TO DO THAT.":
    F=0
216 IF K=0 AND NV(1)<>0 THEN F=1
217 IF K<>0 THEN F=0
218 RETURN
219 PRINT "[CLR]"
220 INPUT "[RVS]T[OFF]APE OR [RVS]D[OFF]ISK  D[ 3 LEFT]";K$
221 IF K$<>"T" AND K$<>"D" THEN 220
222 IF K$="T" THEN 228
223 INPUT "DRIVE [RVS]0[OFF] OR [RVS]1[OFF]  0[ 3 LEFT]";K$
224 IF K$<>"0" AND K$<>"1" THEN 223
225 DN$=CHR$(34)+K$+":"
226 INPUT "[DOWN]TITLE  *[ 3 LEFT]";LT$:IF LT$="*" THEN 226
227 OPEN 1,8,1,DN$+LT$+",S,W":GOTO 230
228 INPUT "OUTPUT TAPE READY TO SAVE GAME";K$:IF LEFT$(K$,1)<>"Y" THEN
    228
229 OPEN 1,1,1,"ADVENTURE GAME"
230 C$=CHR$(13):PRINT#1,SF;C$;LX;C$;DF;C$;R;C$;
231 FOR W=0 TO IL:PRINT#1,IA(W);C$;
232 POKE 59411,53:FOR Z9=1 TO 10:NEXT Z9:POKE 59411,61
233 NEXT W:CLOSE 1
234 GOTO 176
235 READ TL$
236 READ IL,CL,NL,RL,MX,AR,TT,LN,LT,ML,TR
237 CC=CL+3:NN=NL+12
238 DIM NV(1),C0%(CC),C1%(CC),C2%(CC),C3%(CC),C4%(CC),C5%(CC),C6%(CC),C
    7%(CC)
239 DIM NV$(NN,1),IA$(IL),IA(IL),RS$(RL),RM(RL,5),MS$(ML),NT$(1),I2(IL)
240 FOR X=0 TO CL STEP 2:Y=X+1
241 READ C0%(X),C1%(X),C2%(X),C3%(X),C4%(X),C5%(X),C6%(X),C7%(X)
242 READ C0%(Y),C1%(Y),C2%(Y),C3%(Y),C4%(Y),C5%(Y),C6%(Y),C7%(Y)
243 NEXT X
244 FOR X=0 TO NL STEP 5
245 READ NV$(X+0,0),NV$(X+0,1)
246 READ NV$(X+1,0),NV$(X+1,1)
247 READ NV$(X+2,0),NV$(X+2,1)
248 READ NV$(X+3,0),NV$(X+3,1)
249 READ NV$(X+4,0),NV$(X+4,1)
250 NEXT X
251 FOR X=0 TO RL:READ RM(X,0),RM(X,1),RM(X,2),RM(X,3),RM(X,4),RM(X,5),
    RS$(X)
252 NEXT X
253 FOR X=0 TO ML:READ MS$(X):NEXT X
254 FOR X=0 TO IL:READ IA$(X),IA(X):I2(X)=IA(X)
255 NEXT X:MS$(69)=MS$(69)+" WITH YOUR HEAD!":RETURN
256 DATA CASTLE,65,171,55,42,5,1,11,3,100,73,12
257 DATA 5,281,280,0,0,0,6655,0,8,883,0,0,0,0,16076,0
258 DATA 20,883,564,0,0,0,16108,16200,25,484,265,260,0,0,7983,0
259 DATA 25,344,265,260,0,0,7983,0,40,902,1032,0,0,0,10613,9150
260 DATA 40,224,262,280,20,200,6352,8754,50,784,265,1165,260,0,7983,0
261 DATA 50,584,905,900,0,0,8065,0,50,1183,0,0,0,0,17400,0
262 DATA 50,264,322,0,0,0,10501,5761,100,264,134,120,140,0,10800,0
263 DATA 100,124,0,0,0,0,8550,0,100,264,121,120,140,0,10800,0
264 DATA 100,883,324,0,0,0,16162,0,100,28,660,200,20,0,9364,9960
265 DATA 100,664,0,0,0,0,9813,0,100,264,181,180,40,0,10800,0
266 DATA 100,883,624,0,0,0,16159,0,100,883,804,0,0,0,16160,0
```

Listing 1 *(continued)*

```
267 DATA 100,262,581,260,220,0,6812,0,150,281,0,0,0,0,1243,0
268 DATA 161,224,722,620,0,0,8170,9600,161,824,1222,780,0,0,8170,9600
269 DATA 161,1062,800,0,0,0,8170,9600,168,682,440,0,0,0,8170,9600
270 DATA 170,44,60,0,0,0,8170,9600,170,24,60,0,0,0,8170,9600
271 DATA 171,104,240,0,0,0,8170,9600,172,104,120,0,0,0,8170,9600
272 DATA 172,124,140,0,0,0,8454,10564,173,284,260,0,0,0,8170,9600
273 DATA 173,84,260,0,0,0,8170,9600,174,562,300,0,0,0,8170,9600
274 DATA 174,662,304,280,660,280,8162,10564,174,662,284,300,660,300,816
    2,10564
275 DATA 174,662,224,200,660,200,8162,10564,174,761,0,0,0,0,1200,0
276 DATA 174,662,204,220,660,220,8162,10564,174,63,0,0,0,0,4500,0
277 DATA 176,462,800,0,0,0,8170,9600,177,204,0,0,0,0,5100,0
278 DATA 179,204,0,0,0,0,5100,0,179,284,0,0,0,0,5100,0
279 DATA 185,302,84,300,380,380,9354,10564,185,302,384,300,80,80,9354,1
    0564
280 DATA 185,222,0,0,0,0,1200,0,193,842,560,0,0,0,8170,9600
281 DATA 201,0,0,0,0,0,1200,0,1050,604,0,0,0,0,900,0
282 DATA 1050,724,0,0,0,0,900,0,1050,260,0,0,0,0,7806,9900
283 DATA 1200,262,0,0,0,0,3750,0,1200,281,0,0,0,0,1243,0
284 DATA 1213,1162,1054,1040,0,0,7815,0,1215,0,0,0,0,0,9900,0
285 DATA 1216,0,0,0,0,0,3323,0,1223,84,0,0,0,0,1211,0
286 DATA 1224,662,660,60,0,0,8302,2250,1230,462,634,620,0,0,2452,0
287 DATA 1230,484,442,440,460,600,10852,150,1242,704,963,960,520,0,9402
    ,0
288 DATA 1246,544,880,0,0,0,7907,16200,1248,1062,1060,1080,1000,0,10852
    ,17100
289 DATA 1249,0,0,0,0,0,10616,9150,1818,682,141,0,0,0,4650,0
290 DATA 1808,43,141,0,0,0,1231,0,1808,43,132,0,0,0,1246,0
291 DATA 1808,43,123,40,180,0,4872,10350,1810,183,40,180,0,0,2322,0
292 DATA 1818,682,132,0,0,0,1246,0,1818,682,121,680,700,0,10870,9620
293 DATA 1800,0,0,0,0,0,3600,0,2428,84,381,400,320,380,10855,2250
294 DATA 2437,281,0,0,0,0,3324,3450,2439,503,500,0,0,0,8373,1800
295 DATA 3174,204,61,60,660,0,4405,7950,3174,63,724,60,0,0,8352,15450
296 DATA 3174,284,61,60,660,0,4405,7950,3174,63,604,60,0,0,8352,15450
297 DATA 3174,63,60,0,0,0,2303,0,3178,84,381,400,320,380,10855,2250
298 DATA 3181,526,0,0,0,0,1350,0,3181,521,265,520,0,0,2303,0
299 DATA 3181,521,262,520,0,0,18053,0,3191,983,724,980,0,0,8352,15450
300 DATA 3191,604,920,0,0,0,8352,15450,3300,0,0,0,0,0,9813,0
301 DATA 3499,0,0,0,0,0,17550,0,3462,0,0,0,0,0,9750,0
302 DATA 3468,0,0,0,0,0,764,0,3469,0,0,0,0,0,763,0
303 DATA 3471,0,0,0,0,0,764,0,3472,0,0,0,0,0,764,0
304 DATA 3476,0,0,0,0,0,764,0,3480,0,0,0,0,0,871,0
305 DATA 3484,0,0,0,0,0,871,0,3492,0,0,0,0,0,763,0
306 DATA 3450,0,0,0,0,0,10564,0,3900,0,0,0,0,0,9750,0
307 DATA 4050,0,0,0,0,0,9900,0,4207,0,0,0,0,0,10650,0
308 DATA 4350,262,0,0,0,0,3750,0,4361,1082,1052,0,0,0,1203,0
309 DATA 4361,224,265,641,720,240,10815,600,4361,224,265,646,0,0,450,0
310 DATA 4361,1082,1043,1060,1080,0,10815,600,4361,1041,1220,1100,820,0
    ,10854
311 DATA 10564,4361,824,1046,0,0,0,1203,0,4369,161,134,120,0,0,7985,0
312 DATA 4369,161,133,0,0,0,1500,0,4369,161,154,140,0,0,7985,0
313 DATA 4380,0,0,0,0,0,2250,0,4392,0,0,0,0,0,10619,9150
314 DATA 4393,822,820,840,0,0,10815,600,4525,784,1160,260,0,0,10849,765
    0
315 DATA 4525,262,820,0,0,0,8149,7500,4500,0,0,0,0,0,3600,0
316 DATA 4963,1162,1040,0,0,0,2302,0,4962,0,0,0,0,0,9750,0
```

Listing 1 *(continued)*

```
317 DATA 4963,1174,0,0,0,0,5400,0,4950,0,0,0,0,0,2700,0
318 DATA 5111,1062,1043,1060,1080,0,10815,450,5111,824,1041,1220,1100,0
    ,10815
319 DATA 450,5250,84,0,0,0,0,5850,0,5250,404,0,0,0,0,5400,0
320 DATA 5250,724,49,40,0,0,8805,0,5250,764,0,0,0,0,1050,0
321 DATA 5250,604,49,40,0,0,8805,0,5250,484,0,0,0,0,6150,0
322 DATA 5250,464,0,0,0,0,6150,0,5250,262,0,0,0,0,15600,0
323 DATA 5250,744,0,0,0,0,1050,0,5250,281,0,0,0,0,7050,0
324 DATA 5250,48,40,0,0,0,9106,0,5250,0,0,0,0,0,5550,0
325 DATA 5400,0,0,0,0,0,2850,0,5550,0,0,0,0,0,3150,0
326 DATA 5708,183,40,180,0,0,2322,0,5711,1082,1052,0,0,0,1246,0
327 DATA 5711,1082,1043,1060,1080,0,10815,600,5711,224,265,646,0,0,450,
    0
328 DATA 5711,824,1046,0,0,0,1246,0,5711,1041,1220,1100,820,0,10854,105
    64
329 DATA 5718,702,0,0,0,0,1322,0,5850,483,480,0,0,0,3055,0
330 DATA 5888,483,480,0,0,0,8265,3000,6000,503,500,0,0,0,3055,0
331 DATA 6023,0,0,0,0,0,900,0,6039,503,500,0,0,0,2270,0
332 DATA 6161,824,521,1100,1220,0,10815,6064,6161,824,526,0,0,0,1246,0
333 DATA 6450,84,260,0,0,0,8170,9600,6450,284,260,0,0,0,8170,9600
334 DATA 6632,422,654,640,0,0,7965,150,6770,521,24,0,0,0,6000,0
335 DATA 6770,521,44,0,0,0,6000,0,6770,526,0,0,0,0,1246,0
336 DATA 6774,562,521,560,300,220,10840,8250,7080,621,0,0,0,0,4200,0
337 DATA 7080,601,753,0,0,0,7200,0,7080,601,754,740,0,0,5303,0
338 DATA 7084,464,0,0,0,0,3927,0,7236,763,760,0,0,0,7815,0
339 DATA 7536,761,760,0,0,0,7965,0,7650,824,780,0,0,0,8170,9600
340 DATA 7650,744,760,0,0,0,8170,9600,7650,764,740,0,0,0,8170,9600
341 DATA 7845,1183,1180,1200,0,0,10820,9666,7840,1203,0,0,0,0,3000,0
342 DATA 7800,0,0,0,0,0,3600,0,8260,702,0,0,0,0,1322,0
343 DATA AUT,ANY,GO,NORTH,*ENT,SOUTH,*RUN,EAST,*WAL,WEST
344 DATA*CLI,UP,*CRA,DOWN,JUM,GAM,GET,TOR,*TAK,*ON
345 DATA*PIC,OFF,*CAT,DOO,LIG,SCO,*TUR,KEY,*TOR,HEL
346 DATA*BUR,INV,DRO,SIG,*REL,ARO,*SPI,FIR,*LEA,KNA
347 DATA*GIV,TRE,THR,HIL,QUI,CAV,LOO,WAT,*SHO,ROP
348 DATA*SEE,GUA,SCO,PAS,INV,OPE,SAV,PIL,OPE,WAL
349 DATA ATT,BOO,*KIL,AXE,*FIG,PIA,FIN,*MUS,LOC,GRA
350 DATA HEL,DRA,SAY,HOR,STO,ARM,UNL,FOO,EAT,WIN
351 DATA DRI,PRI,BRE,LAD,*SMA,CHE,SWI,COF,PLA,*STA
352 DATA CUT,TOA,*CHO,SKU,REA,BAT,RID,BAR,*MOU,COB
353 DATA DIS,*WEB,IVA,LED,KIS,"",*LOV,"",*FUC,""
354 DATA FIR,"","","","","","","","",""
355 DATA 0,0,0,0,0,0,""
356 DATA 1,4,1,2,0,0,A THICK WOODS
357 DATA 2,4,1,5,0,0,A DARK FOREST
358 DATA 0,0,0,0,0,1,*YOU ARE ON A BRANCH NEAR THE TOP OF A    TREE
359 DATA 1,0,0,0,0,0,*YOU ARE BY THE WATER'S EDGE OF THE MOAT
360 DATA 0,0,2,0,0,0,A CLEARING BY THE SIDE OF A  SMALL HILL
361 DATA 0,0,5,0,0,0,*YOU ARE BY THE ENTRANCE TO A CAVE
362 DATA 0,7,6,8,7,0,A LARGE CAVERN WITH TUNNELS  GOING OFF IN ALL DIRE
    CTIONS
363 DATA 8,8,7,8,9,8,A SMALL CAVERN WITH MANY     OPENINGS LEADING FROM
     IT
364 DATA 8,8,8,10,9,8,A LOW PASSAGE-WAY
365 DATA 0,0,9,0,0,0,A LARGE CAVERN WITH HIGH     SHEER WALLS
366 DATA 0,0,0,0,0,0,A SMALL ROOM CARVED INTO THE CAVE WALL
```

Listing 1 *(continued)*

```
367 DATA 2,1,0,0,0,5
368 DATA*YOU ARE ON THE TOP OF A SMALL HILL--THE CASTLE IS SOUTH
369 DATA 4,14,0,0,0,0,THE WATER--IT IS ICY COLD
370 DATA 13,0,0,0,0,0,*YOU ARE BY THE OUTER CASTLE WALLS
371 DATA 0,0,18,0,16,0,*YOU ARE ON A CATWALK NEAR THE WEST TOWER
372 DATA 0,0,0,17,0,15,A SMALL CHAMBER IN THE WEST  TOWER
373 DATA 0,0,16,0,0,0,THE GUARDS' QUARTERS
374 DATA 0,0,20,15,0,19,THE WEST END OF A GREAT HALL
375 DATA 23,0,0,0,18,0,THE CASTLE COURTYARD
376 DATA 21,0,0,18,24,0,THE EAST END OF THE GREAT    HALL
377 DATA 0,20,0,19,0,0,THE KITCHEN
378 DATA 18,0,0,0,0,25,A SECRET PASSAGE BEHIND THE  FIREPLACE
379 DATA 0,19,0,0,0,0,THE CASTLE'S STABLES
380 DATA 0,0,0,0,26,20,THE BARON'S PRIVATE CHAMBERS
381 DATA 0,0,0,0,22,27,A LONG LOW TUNNEL SLOPING     DOWNWARD
382 DATA 0,0,0,0,0,24,THE TURRET ROOM OF THE EAST  TOWER
383 DATA 0,0,0,0,25,0,THE BARON'S FAMILY CRYPT
384 DATA 0,0,29,0,27,0,A CAVERN WITH IRIDESCENT     GLOWING WALLS
385 DATA 0,30,0,28,0,0,AN ODD-SHAPED ROOM WITH A    STRANGE SMELL
386 DATA 29,0,0,0,0,0
387 DATA*YOU ARE ON A LEDGE ON THE NORTH SIDE OF A DEEP AND WIDE CHASM
388 DATA 32,11,0,34,0,0,THE EAST END OF A LARGE       CAVERN
389 DATA 33,31,0,0,0,35,*YOU ARE AT A CROSS-ROADS IN A LONG      TUNNEL
390 DATA 0,0,0,0,0,0,*YOU ARE LOST IN THE PET ROMS
391 DATA 35,0,31,0,0,0,THE WEST END OF A LARGE       CAVERN
392 DATA 0,34,36,0,32,0,THE BARON'S SECRET STRONGHOLD
393 DATA 0,0,0,35,0,0
394 DATA*YOU ARE ON A LEDGE ON THE SOUTH SIDE OF A DEEP AND WIDE CHASM
395 DATA 0,0,0,0,0,0,THE BARON'S TREASURE ROOM
396 DATA 38,39,0,0,38,39,A TWISTING TUNNEL IN THE     DUNGEON
397 DATA 38,39,0,0,38,39,A LONG PASSAGE-WAY WITH ROWS OF CELLS
398 DATA 0,0,37,24,0,0,A SECRET PASSAGEWAY
399 DATA 0,0,0,0,0,0,A DUNGEON CELL
400 DATA 33,3,33,33,33,12,*PICK A DIRECTION AND YOU MIGHT RETURN TOLIFE
401 DATA "",THERE'S A STRANGE SOUND,NOTHING HAPPENS,IT'S LOCKED,IT'S OP
    EN
402 DATA THERE'S SOMETHING THERE-MAYBE YOU SHOULD
403 DATA THAT'S NOT VERY SAFE,YOU MAY NEED MAGIC HERE,SORRY...YOU CAN'T
404 DATA YOU DON'T HAVE IT,IT'S EMPTY,YOU HAVE NO CONTAINER
405 DATA WHAT A WASTE...,OPEN IT?,GO THERE?,OK...,YOU FOUND SOMETHING!
406 DATA YOU DIDN'T FIND ANYTHING
407 DATA YOU DON'T SEE IT HERE,USE ONE WORD,OH...THAT FEELS GOOD!
408 DATA TO STOP GAME SAY QUIT,A VOICE BOOMS OUT...,LEAVE IT ALONE!
409 DATA DON'T BE SILLY!,THE GUARD WON'T LET YOU!,THE BARON SUCKS EGGS!
410 DATA READ SCOTT FOR MAGIC WORD,IVANHOE!
411 DATA THE HOOK CATCHES AND THE ROPE HANGS DOWN
412 DATA YOU HAVE TO THROW IT FIRST!,THE MATCHES ARE TOO WET,IT IS BLAZ
    ING
413 DATA AN ANGRY-LOOKING GUARD APPEARS,IT'S TOO HIGH AND STEEP
414 DATA SOMETHING FELL OUT
415 DATA TRY THE PIANO,BE ADVENTEROUS!,YOU MADE A TASTY MEAL!
416 DATA YOU NEED SOME EXERCISE,CRASH...IT'S DOWN!,READING IMPROVES THE
    MIND
417 DATA THE GUARD THROWS YOU OVER THE EDGE,WITH A BROKEN ARM
418 DATA YOUR ARM HAS HEALED,THE GUARD MISTAKES YOU FOR THE BARON ANDLE
    AVES
```

Listing 1 *(continued)*

```
419 DATA YOU DON'T HAVE THE NECESSARY RESOURCES,TIME HEALS ALL THINGS
420 DATA TRY ANOTHER BOOK,IT WAS A FIERCE FIGHT,BUT YOU LOST
421 DATA BUT YOU FINALLY WON,IT DISAPPEARS,INTO THE CHASM
422 DATA LEAVE PEACEFULLY!,THROW SOMETHING,SORRY ABOUT THAT
423 DATA THE SKULL SPEAKS---,BEWARE THE VAMPIRE,BEWARE THE CHEST
424 DATA BEWARE THE GOLD BARS,YOU JUST LOST YOUR CURSOR AND THE GAME  I
    S OVER
425 DATA LEAVE THE GUARD IN HIS QUARTERS,THE BAT JUST BITE YOU
426 DATA THE DOOR SLAMS SHUT!,YOU HEAR THE FLAP OF WINGS
427 DATA THE TOAD SAYS--RIB-BIT!
428 DATA YOU SEE A POISONOUS SPIDER CRAWLING      TOWARD YOU
429 DATA YOU WERE JUST BITTEN BY A POISONOUS      SPIDER
430 DATA THE BARON SUDDENLY APPEARS WITH A LARGE SWORD AND IT'S OFF
431 DATA IT MISSED!,READ IT?,YOU HAVE NO WATER,THE BOTTLE SHATTERS!
432 DATA TREES,1,TREES,2,UNLITE TORCH/TOR/,6
433 DATA ROPE WITH GRAPPLING HOOK ON END/ROP/,9,CAVE IN THE SIDE OF THE
     HILL,5
434 DATA SIGN SAYING--NO SWIMMING--DANGER!,4,MATCHES/MAT/,0,SOGGY MATCH
    ES/MAT/
435 DATA 18,KNAPSACK/KNA/,-1,BLAZING TORCH/TOR/,0,THE CASTLE ACROSS THE
     WATER,4
436 DATA THE DRAWBRIDGE IS UP,4,A HEAVY OAK DOOR,11,AN ANGRY-LOOKING GU
    ARD,11
437 DATA A BROKEN ARM,0,THE DRAWBRIDGE IS DOWN,0
438 DATA A SCHOOL OF MAN-EATING PIRANHA,13
439 DATA SIGN -- LEAVE TREASURE HERE--SAY 'SCORE',12
440 DATA OPENING IN THE WALL--HIGH UP NEAR THE   TOP OF THE CAVERN,10
441 DATA*MAGIC SLEEPING PILLS*/PIL/,12,A SCHOOL OF SLEEPY PIRANHA,0
442 DATA A GRAND PIANO,20,A BOOKCASE FILLED WITH CLASSICS,24
443 DATA A SECRET PASSAGE BEHIND THE BOOKCASE,0,FOOD/FOO/,21,WINE/WIN/,
    21
444 DATA A BATTLE AXE/AXE/,17,GRAFFITI ON THE WALLS,23
445 DATA A THICK ROPE HOLDING THE DRAWBRIDGE UP,19,*SILVER SUIT OF ARMO
    R*/ARM/
446 DATA 24,A BOOK OF HORROR STORIES/BOO/,0,*RARE 1ST EDITION OF SCOTT*
    /BOO/,0
447 DATA SOME PIANO KEYS/KEY/,0,A ROPE HANGING DOWN,0,A FIREPLACE,18
448 DATA A FIREPLACE WITH A BLAZING FIRE,0,AN OPEN DOOR,0
449 DATA*A LARGE DIAMOND*/DIA/,0,*THE BARON'S BEST HORSE*/HOR/,23,MUDDY
     WATER
450 DATA 4,*JEWELED URN*/URN/,27,COFFIN,27,AN OPEN COFFIN WITH STAIRS I
    NSIDE,0
451 DATA BONES/BON/,27,SKULL/SKU/,27,A VAMPIRE BAT,0,*BAG OF RUBIES*/RU
    B/,29
452 DATA SIGN--DANGER--GO BACK!,32,*TREASURE CHEST*/CHE/,35,A LONG LADD
    ER/LAD/
453 DATA 36,*GOLD BARS*/BAR/,37,*SILVER CROSS*/CRO/,37,GUARD'S KEYS/KEY
    /,0
454 DATA AN OPEN DOORWAY,37,A LOCKED STEEL DOOR,0,A LOCKED OAK DOOR,41
455 DATA ANOTHER LEDGE ON THE NORTH SIDE,36,A NARROW LEDGE ON THE OTHER
     SIDE
456 DATA 30,AN UNCONSCIOUS GUARD,0,A HORNY TOAD/TOA/,41
457 DATA*BEAUTIFUL PRINCESS*/PRI/,0,AN OPEN DOORWAY,0,THE CASTLE TO THE
     SOUTH
458 DATA 3,COBWEBS,27,COBWEBS,35,COBWEBS,38
```

***Listing 2:* Changes to Listing 1 for Apple II**

```
1 HOME : VTAB 4: HTAB 6: PRINT "Welcome to CASTLE ADVENTURE
2 HTAB 10: PRINT "By David Malmberg
3 VTAB 8: HTAB 3: PRINT "Based on the Ideas, Data Structure
4 HTAB 3: PRINT "and Driver Program of Scott Adams
5 VTAB 12: PRINT "Reading Data Tables — Just a moment
21 HOME
22 R = AR:LX = LT:DF = 0:SF = 0: INPUT "Use Saved Game(Y or N)? ";K$
23 IF LEFT$ (K$,1) < > "Y" THEN  HOME : GOTO 37
24 REM  Delete lines 24-29
30 INPUT "Filename? ";LT$: IF  LEN (LT$) = 0 THEN 30
31 D$ =  CHR$ (4): PRINT D$"OPEN"LT$
32 REM  Delete 32-33
34 PRINT D$"READ"LT$: INPUT SF: INPUT LX: INPUT DF: INPUT R
35 FOR X = 0 TO IL: INPUT IA(X): NEXT
36 PRINT D$"CLOSE"
38 PRINT : INPUT "What do you want to do? ";TP$: PRINT : GOSUB 45
69 FOR Z = 0 TO IL: IF K AND IA(Z) = R THEN  PRINT : PRINT "Visible items here:":
   K = 0
76 GOSUB 71: IF  PEEK (36) +  LEN (TP$) + 3 > 39 THEN  PRINT
142 IF  NOT L THEN  HOME
161 FOR X = 0 TO IL:IA(X) = I2(X): NEXT : HOME : GOTO 22
169 GOSUB 71: IF  PEEK (36) +  LEN (TP$) > 39 THEN  PRINT
175 HOME
219 HOME
220 REM  Delete 220-225
226 INPUT "Filename? ";LT$: IF  LEN (LT$) = 0 THEN 226
227 D$ =  CHR$ (4): PRINT D$"OPEN"LT$
228 REM Delete 228-229
230 PRINT D$"WRITE"LT$: PRINT SF: PRINT LX: PRINT DF: PRINT R
231 FOR W = 0 TO IL: PRINT IA(W)
232 REM  Delete
233 NEXT : PRINT D$"CLOSE"
```

ROCKWELL Microcomputers from Excert, Inc.

THE AIM 65/40
Single Board or Smorgasbord

- A full size terminal style keyboard w/8 special function keys
- A smart, 40 character display with its own microprocessor
- A 40 column printer w/text and graphic output
- Up to 64K of on-board RAM and ROM
- On-board interfaces include RS232, dual audio cassette and 2 user I/O R6522 devices
- Firmware includes interactive monitor and text editor w/options of Assembler, BASIC, FORTH and PL/65

THE AIM 65
Take-Out Order

- A full size terminal style keyboard w/3 special function keys
- A 20 character display
- A 20 column printer w/text and graphic output capability
- Up to 4K RAM and 20K ROM on-board
- On-board interfaces include 20MA TTY, dual audio cassette and 1 user I/O R6522 device
- Firmware includes interactive monitor and text editor w/options of Assembler, BASIC, FORTH, PASCAL, & PL/65

And if the above isn't enough,
Try the RM65 — a product line filled with embellishments including:

32K DRAM Board	ACIA Board	Prototype cards
CRT Controller	IEEE-488Board	Adaptor Buffer Modules
Floppy Disk Controller	CPU/SBC Board	General Purpose I/O Board
PROM Programmer	4-16 Slot Card Cages	PROM/ROM Board

NEW LOWER PRICES AND A CASH DISCOUNT* TO BOOT!

A65/40-16 (16K RAM)	$1225
A65/40-32 (32K RAM)	$1295
A65/40-A (Assembler)	$ 85
A65/40-B (BASIC)	$ 65

A65-1 (1K RAM)	$420
A65-4 (4K RAM)	$445
A65-4B (4K RAM w/BASIC)	$495
A65-PS (PASCAL)	$100
A65-F (FORTH)	$ 65
A65-A (Assembler)	$ 35

Mail Order to:

Educational Computer Division
EXCERT INCORPORATED

- SALES
- SERVICE
- INSTALLATION
- CONSULTING

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

Higher quantities quoted upon request, COD's accepted, shipping will be added. *Deduct 5% cash discount on prepaid orders. Minnesota residents add 5% sales tax. Prices subject to change without notice.

SYM 23 Matches

by Matt Ganis

Two players alternate, removing matches from a pile of 23; the player taking the last match loses. On each turn a player may take 1, 2, or 3 matches.

23 Matches
requires:
1K SYM

In this version of the game the human challenges the computer. Key in G 0200 to begin the game. The LEDs will go blank and the SYM will wait for you to press any key on the keypad. Once this is done the computer informs the player that it is his turn. At this point the player enters either a 1, 2, or 3 (taking away 1, 2, or 3 matches). The SYM will inform the player of the number of matches it is taking by scrolling the message 'I SELECT....'

Modifications to the program are very simple:

1. To change the speed of the messages that scroll on the display, decrement the value at location $02FA.
2. To make the computer move first, change the 0 at location $0204 to 1.
3. To alter the number of matches used, change the $17 (hex $17, decimal 23) to the number of matches desired.

The only part of the program that might be used in another program is the 'display-a-message' routine at location $02EE. This routine will scroll a message across the SYM displays starting at the right-most display and ending at any desired display. Just set the zero page pointer (PTR) to the location of the message (lo, hi), and load the Y register with the number of displays to be used minus one. Then do a JSR MESSAGE.

Mr. Ganis studies computer science at PACE University (Pleasantville/Briarcliff). He may be contacted at Sheridan Road R.D. #3, Lebanon, NJ 08833.

23 Matches

```
                         ;THIS IS A GAME OF 23 MATCHES.
                         ;  EACH PERSON (THE COMPUTER AND
                         ;  THE PLAYER TAKE TURNS TAKING
                         ;  AWAY MATCHES FROM THE PILE.
                         ;
                         ;ON EACH TURN YOU MAY TAKE ONLY
                         ;  1,2 OR 3 MATCHES.  THE PLAYER
                         ;  TAKING THE LAST MATCH LOSES !!
                         ;
                         ;BASIC VERSION FROM -
                         ;
                         ;101 BASIC COMPUTER GAMES
                         ;   EDITED BY DAVID H. AHL
                         ;
                         ;
                         ;TO START GAME KEY ' G 0200 '
                         ;
                         ;
                         ;***ZERO PAGE LOCATIONS USED
                         ;
                         ;
                         ;
 80:   0200              MOVE     =    $00        ;WHOSE MOVE IT IS
 81:   0200              MATCHES  =    $01        ;NUMBER OF MATCHES
 82:   0200              DISPLAY  =    $02        ;LAST DISPLAY USED
 83:   0200              RND      =    $04        ;RANDOM NUMBER
 84:   0200              PTR      =    $05        ;POINTER FOR MESSAGES
 85:   0200              COUNT1   =    $07        ;DELAY COUNTER 1
 86:   0200              COUNT2   =    $08        ;DELAY COUNTER 2
 87:   0200              TEMP     =    $09        ;TEMP. STORAGE
 88:   0200              REG      =    $0A        ;1ST REG. COMPUTER MOVE
 89:   0200              SYMMOVE  =    $0B        ;COMPUTER'S MOVE
 90:   0200              REG2     =    $0C        ;2ND REG. COMPUTER MOVE
 91:   0200              TEMP2    =    $0D        ;TEMP. STORAGE 2
                         ;
                         ;
                         ;***MONITOR LOCATIONS USED***
                         ;
                         ;
 93:   0200              GETKEY   =    $88AF      ;GET A KEY FROM KEYPAD


 94:   0200              LED      =    $A640      ;LOCATION OF 1ST DISPLAY
 95:   0200              SCAND    =    $8906      ;SCAN THE DISPLAY
 96:   0200              ACCESS   =    $8B86      ;ENABLE SYSTEM RAM
 97:   0200              SEGCODES =    $8C29      ;TABLE OF SEGMENT COEES
 98:   0200              KEYSTAT  =    $896A      ;IF A KEY IS DOWN C=1
                         ;
                         ;
100:   0200                       *=   $0200

110:   0200 20 86 8B MAIN         JSR  ACCESS     ;ENABLE RAM
121:   0203 A9 00                 LDA  #0
121:   0205 85 00                 STA  MOVE       ;PLAYER FIRST
130:   0207 A9 17                 LDA  #23
130:   0209 85 01                 STA  MATCHES    ;MATCHES=23
140:   020B A9 42                 LDA  #$42
140:   020D 85 02                 STA  DISPLAY
140:   020F A9 A6                 LDA  #$A6
140:   0211 85 03                 STA  DISPLAY+1
150:   0213 E6 04    RANDOM       INC  RND        ;GENERATE RANDOM #
160:   0215 20 6A 89              JSR  KEYSTAT    ;WAIT FOR KEY DOWN
165:   0218 90 F9                 BCC  RANDOM     ;IF NO KEY,BUMP RND
                         ;
```

(continued)

MICRO™ New Publications

40 Computer Games from Kilobaud Microcomputing, edited by Emily A. Gibbs and Jim Perry. Wayne Green, Inc. (Peterborough, NH), 1980, 148 pages, paperback. $7.95

40 Computer Games offers you some of the best game programs from recent issues of *Kilobaud Microcomptuing.* Nine game categories offer something for everyone. Accompanying articles explain how to play the games and increase the odds to beat the computer. The games are written in different languages for various computer systems.

CONTENTS: Gambling; Racing; Space; Board Games; Card Games; Guessing Games; Puzzles; Calculators; Odds and Ends.

Science and Engineering Sourcebook by Cass Lewart. Micro Text Publications Inc. (One Lincoln Plaza, Suite 27C, New York, NY 10023), 1982, 95 pages, 6 × 9 inches, paperback.
ISBN: 0-942412-02-8 $9.95

A book of professional applications programs for the TRS-80 Pocket Computer. The programs cover problems in the field of electrical engineering, statistics, queuing theory, reliability, graph generation, artificial intelligence, and related technical disciplines. A table of conversions make the translation of these programs applicable to other BASIC computers.

CONTENTS: Foreword; Introduction; Electrical Engineering; Data Transmission; Number Theory; Computer Programming; Computer Generated Plotting; Probability and Statistics; Mathematics; Operations Research; Miscellaneous; Appendix.

Games for the ATARI, by S. Roberts. Elcomp Publishing, Inc. (53 Redrock Lane, Pomona, CA 91766), 1982, 115 pages, paperback.
ISBN: 3-911682-84-3 $7.95

Games for the Atari provides ideas on how to create your own computer games. This booklet deals primarily with BASIC examples with only one example in machine language. Atari programs show the possibilities of using both graphics and sound features.

CONTENTS: Drawing Figures on the Screen; Movements in BASIC; Movements in Machine Language; Movements of Missiles; Overlapping Detection; Sound-

(Continued on next page)

23 Matches *(continued)*

```
021A A9 00    PRINT    LDA  #0           ;PRINT MATCHES
021C 85 09             STA  TEMP
021E A5 01             LDA  MATCHES
0220 38       PNT2     SEC               ;DETERMINE TENS
0221 E9 0A             SBC  #$0A         ;BY SUCCESSIVE
0223 30 04             BMI  PNT3         ;SUBTRACTIONS OF 10
0225 E6 09             INC  TEMP         ;BUMP TENS SPOT
0227 D0 F7             BNE  PNT2         ;BRANCH ALWAYS
0229 18       PNT3     CLC
022A 69 0A             ADC  #$0A         ;ADD 10 BACK
022C AA                TAX               ;TO GET UNITS
022D BD 29 8C          LDA  SEGCODES,X   ;GET SEGMENT CODE
0230 8D 41 A6          STA  LED+1        ;2ND DISPLAY
0233 A6 09             LDX  TEMP         ;GET TENS
0235 BD 29 8C          LDA  SEGCODES,X   ;  GET THE CODE
0238 8D 40 A6          STA  LED          ;1ST DISPALY
023B A5 01             LDA  MATCHES      ;TEST FOR ZERO
023D D0 03             BNE  OVER         ;MATCHES, BRANCH IF N
023F 4C D0 02          JMP  LOST         ;SOMEONE LOST !!
                       ;
0242 A9 01    OVER     LDA  #1           ;DETERMINE WHOSE
0244 38                SEC               ;MOVE IT IS BY
0245 E5 00             SBC  MOVE         ;COMPUTING MOVE=1-MOVE
0247 85 00             STA  MOVE
0249 F0 27             BEQ  COMPUTER     ;IF 0 COMPUTER'S MOVE
024B A9 1B             LDA  #<PLYRMOVE   ;PRINT THE
024D 85 05             STA  PTR          ;  'YOUR MOVE'
024F A9 03             LDA  #>PLYRMOVE   ;  MESSAGE
0251 85 06             STA  PTR+1
0253 20 EE 02          JSR  MESSAGE      ;DISPLAY IT
0256 20 AF 88 GET      JSR  GETKEY       ;WAIT FOR PLAYERS
0259 C9 34             CMP  #$34         ;MOVE
025B B0 F9             BCS  GET          ;BRANCH IF KEY>'4'
025D C9 31             CMP  #$31
025F 90 F5             BCC  GET          ;BRANCH IF KEY<'1'
0261 38                SEC               ;CONVERT FROM ASCII
0262 E9 30             SBC  #$30         ;TO INTEGER
0264 85 09             STA  TEMP         ;KEEP IN TEMP
0266 A5 01             LDA  MATCHES      ;MAKE SURE PLAYER
0268 38                SEC               ;DIDN'T TAKE MORE
0269 E5 09             SBC  TEMP         ;THAN WHAT'S IN PILE
026B 30 E9             BMI  GET          ;BRANCH IF <0
026D 85 01             STA  MATCHES      ;ELSE STORE RESULT
026F 4C 1A 02          JMP  PRINT        ; AND CONTINUE.
                       ;
                       ;***COMPUTER'S MOVE***
                       ;
0272 A5 01    COMPUTER LDA  MATCHES      ;IF MATCHES=1
0274 C9 01             CMP  #1           ;COMPUTER LOST
0276 D0 03             BNE  CMOVE
0278 4C AD 02          JMP  FINISH       ;JUMP WITH ACC=1
027B 4A       CMOVE    LSR               ;COMPUTE -
027C 4A                LSR               ;R=4*INT(MATCHES/4)
027D 0A                ASL
027E 0A                ASL
027F 85 0A             STA  REG          ;KEEP IT IN REG
0281 38                SEC               ;COMPUTE -
0282 A5 01             LDA  MATCHES      ;R=MATCHES-REG
0284 E5 0A             SBC  REG
0286 85 0A             STA  REG          ;KEEP IT IN REG
0288 C9 01             CMP  #1           ;IF REG=1 THEN
028A D0 13             BNE  CMOVE3       ;MAKE RANDOM MOVE
                       ;
                       ;***RANDOM NUMBER GENERATOR***;
                       ;
028C D8       CMOVE2   CLD               ;BINARY ADDITION
028D A5 04             LDA  RND          ;GET RND
028F 0A                ASL               ;COMPUTE -
0290 0A                ASL               ; RND=4*RND
0291 18                CLC
0292 65 04             ADC  RND          ;RND=5*RND
0294 18                CLC
0295 69 01             ADC  #1           ;RND=5*RND+1
0297 85 04             STA  RND
0299 29 03             AND  #%00000011   ;MAKE SURE 1<RND<3
029B F0 EF             BEQ  CMOVE2       ;NO ZERO'S ALLOWED
029D D0 12             BNE  DONE         ;BRANCH ALWAYS
029F 18       CMOVE3   CLC
02A0 A5 0A             LDA  REG          ;COMPUTE -
02A2 69 03             ADC  #3           ;REG2=REG+3
02A4 85 0C             STA  REG2
02A6 4A                LSR               ;COMPUTE -
02A7 4A                LSR               ; 4*INT(REG2/4)
02A8 0A                ASL
02A9 0A                ASL
02AA 85 0D             STA  TEMP2        ;AND KEEP IN TEMP2
02AC 38                SEC               ;DETERMINE MOVE BY
```

23 Matches *(continued)*

```
02AD A5 0C     FINISH   LDA REG2          ;COMPUTING -
02AF E5 0D              SBC TEMP2         ;  REG2-TEMP2
02B1 85 0B     DONE     STA SYMMOVE       ;THIS IS COMPUTER'S MOVE
02B3 38                 SEC
02B4 A5 01              LDA MATCHES       ;TAKE AWAY FROM PILE
02B6 E5 0B              SBC SYMMOVE
02B8 85 01              STA MATCHES       ;PUT RESULT BACK
02BA A6 0B              LDX SYMMOVE       ;GET SEGMENT CODE
02BC BD 29 8C           LDA SEGCODES,X    ;OF COMPUTER'S MOVE
02BF 8D 51 03           STA COMPBYTE+1    ;STORE IN MESSAGE
02C2 A9 44              LDA #<COMPMOVE    ;PRINT THE
02C4 85 05              STA PTR           ;'I SELECT X'
02C6 A9 03              LDA #>COMPMOVE    ;MESSAGE
02C8 85 06              STA PTR+1
02CA 20 EE 02           JSR MESSAGE       ;DISPLAY IT
02CD 4C 1A 02           JMP PRINT         ;CONTINUE
02D0 A5 00     LOST     LDA MOVE          ;SOMEONE LOST
02D2 C9 01              CMP #1            ;WHO IS IT PRINT
02D4 F0 0A              BEQ PLAYER        ;IF MOVE=1 THEN PLAYER
02D6 A9 57              LDA #<COMPLOSE    ;PRINT THE
02D8 85 05              STA PTR           ;'I LOSE ..'
02DA A9 03              LDA #>COMPLOSE    ;MESSAGE
02DC 85 06              STA PTR+1
02DE D0 08              BNE DISMES        ;BRANCH ALWAYS!
02E0 A9 2D     PLAYER   LDA #<PLYRLOSE    ;PRINT THE
02E2 85 05              STA PTR           ;'YOU LOSE ..'
02E4 A9 03              LDA #>PLYRLOSE    ;MESSAGE
02E6 85 06              STA PTR+1
02E8 20 EE 02  DISMES   JSR MESSAGE       ;DISPLAY IT
02EB 4C 00 02           JMP MAIN          ;START ALL OVER
                ;
                ;***DISPLAY A MESSAGE***
                ;
02EE A0 03     MESSAGE  LDY #3            ;NUMBER OF DISPLAYS-1
02F0 B1 05     MESS1    LDA (PTR),Y       ;GET A BYTE
02F2 30 20              BMI BYE           ;IF NEG, STOP
02F4 91 02              STA (DISPLAY),Y   ;  DISPLAY IT
02F6 88                 DEY
02F7 10 F7              BPL MESS1         ;KEEP GOING
02F9 A9 02     DELAY    LDA #2            ;DELAY A LITLE BIT
02FB 85 07              STA COUNT1
02FD A9 FF     D1       LDA #$FF
02FF 85 08              STA COUNT2
0301 20 06 89  D2       JSR SCAND         ;SCAN THE DISPLAY
0304 C6 08              DEC COUNT2
0306 D0 F9              BNE D2
0308 C6 07              DEC COUNT1
030A D0 F1              BNE D1
030C E6 05              INC PTR           ;BUMP PTR
030E D0 02              BNE D3
0310 E6 06              INC PTR+1         ;DOUBLE BUMP
0312 D0 DA     D3       BNE MESSAGE       ;BRANCH ALWAYS
0314 60        BYE      RTS
                ;
                ;
031B                    *=   $31B
031B 00 00 00  PLYRMOVE.BYTE $00,$00,$00  ;DATA FOR THE
031E 00 6E 5C          .BYTE $00,$6E,$5C  ;'YOUR MOVE'
0321 1C 50 00          .BYTE $1C,$50,$00  ;  MESSAGE
0324 78 1C 50          .BYTE $78,$1C,$50
0327 54 00 00          .BYTE $54,$00,$00
032A 00 00 80          .BYTE $00,$00,$80
                ;
032D 00 00 00  PLYRLOSE.BYTE $00,$00,$00  ;DATA FOR THE
0330 00 6E 5C          .BYTE $00,$6E,$5C  ;'YOU LOSE..'
0333 1C 00 38          .BYTE $1C,$00,$38  ;  MESSAGE
0336 3F 6D 79          .BYTE $3F,$6D,$79
0339 00 76 77          .BYTE $00,$76,$77
033C 00 76 77          .BYTE $00,$76,$77
033F 00 00 00          .BYTE $00,$00,$00
0342 00 80             .BYTE $00,$80
                ;
0344 00 00 00  COMPMOVE.BYTE $00,$00,$00  ;DATA FOR THE
0347 00 06 00          .BYTE $00,$06,$00  ;'I SELECT X'
034A 6D 79 38          .BYTE $6D,$79,$38  ;  MESSAGE
034D 79 39 78          .BYTE $79,$39,$78
0350 00 FF 00  COMPBYTE.BYTE $00,$FF,$00
0353 00 00 00          .BYTE $00,$00,$00
0356 80                .BYTE $80
                ;
0357 00 00 00  COMPLOSE.BYTE $00,$00,$00  ;DATA FOR THE
035A 00 06 00          .BYTE $00,$06,$00  ;'I LOSE..'
035D 38 3F 6D          .BYTE $38,$3F,$6D  ;MESSAGE
0360 79 00 7C          .BYTE $79,$00,$7C
0363 5C 5C 00          .BYTE $5C,$5C,$00
0366 78 06 6D          .BYTE $78,$06,$6D
0369 6D 00 00          .BYTE $6D,$00,$00
036C 00 00 80          .BYTE $00,$00,$80
```

MICRO

New Publications *(continued)*

features; Programming the Joystick; Backgammon; SMARTY; BOMBER; ROBOT ATTACK; BALL; SMART; BARRIER; KNIGHT-BATTLE; CALENDAR; GUNFIGHT; Appendix; The Video Processor "ANTIC" and the Atari 400/800; Display List Interrupts and the Atari, Atari 400/800 and CTIA/GTIA; The Atari 400/800 and its Character Set.

Apple II Assembly Language, by Marvin L. De Jong. Howard W. Sams & Co., Inc. (4300 West 62nd St., Indianapolis, IN 46268), 1982, 334 pages, 5¼ × 8½ inches, paperback.
ISBN: 0-672-21894-1 $15.95

This is a 6502 assembly-language manual written for the beginning assembly-language programmer on the Apple II. Dr. De Jong introduces each topic in a building-block concept, starting with a description of a microcomputer and continuing through interrupt programming and real-time applications. The book is carefully written and well illustrated with programming examples. The only possible flaw with the book is that it quickly gets technical after the first half dozen chapters. If the reader were truly a beginner, he would find the material difficult.

CONTENTS: The Microcomputer System; Writing and Executing Simple Assembly-Language Programs; Branches and Loops; Logical Operations and Shift and Rotate Operations; Arithmetic Operations; Addressing Modes: Indexed Addressing; Subroutines, The Stack and Interrupts; Additional Programming Topics; Programming with the 6522; Applications; Decimal, Binary, and Hexadecimal Number Systems; Additional Circuits and Programs; Pin Diagrams of Some Integrated Circuits; Index.

Phil Daley
MICRO Staff

Kids and the Apple, by Edward H. Carlson. Reston Publishing Company, Inc. (Reston, VA), 1982, 218 pages, paperback.
ISBN: 0-8359-3669-4 $19.95

This book teaches Applesoft BASIC on both disk-based or cassette Apple systems to children from 10-14 years old. The book is intended for self-study, but may also be used in a classroom setting. The lessons contain explanations, examples, exercises, and review questions. Notes for the instructor summarize the lesson material, provide helpful hints, and give good review questions.

CONTENTS: Introduction; Graphics, Games, and All That; Advanced Programming.

MICRO™

Chances are, when you bought your first disk drive, it was an Apple. Now that you're ready for a second, take a look at Quentin.

Our Apple*-Mate™ 5¼" Disk Drive is fully software transparent with Apple's DOS 3.3 operating system in full and half track operation.

Add it to your present drive for greater capacity and faster access. Just plug it in and go to work.

And the Apple-Mate has these High Performance advantages:

ON TRACK HEAD SEEK

A precision lead screw positions the head onto the correct track. Time-consuming retries and disk-to-disk copying errors are virtually eliminated.

SIEMENS† DISK DRIVE

The apple-beige unit is built around the highly reliable Siemens system with over 10,000 lifetime hours. Shielded connecting cable also attached.

LONG TERM DEPENDABILITY

MTBF (Mean Time Between Failures)—8,500 power-on hours, and the unit has a one-year warranty.

COUNT ON QUENTIN FOR QUALITY

Quentin Research was building disk systems for the computer industry when Apple was a little bud on the big computer tree. We're known for product reliability and stand behind every system we sell you.

But the best news may be the price—only $335.00 (40 tracks).

A special introductory offer when you order Apple-Mate directly from us.

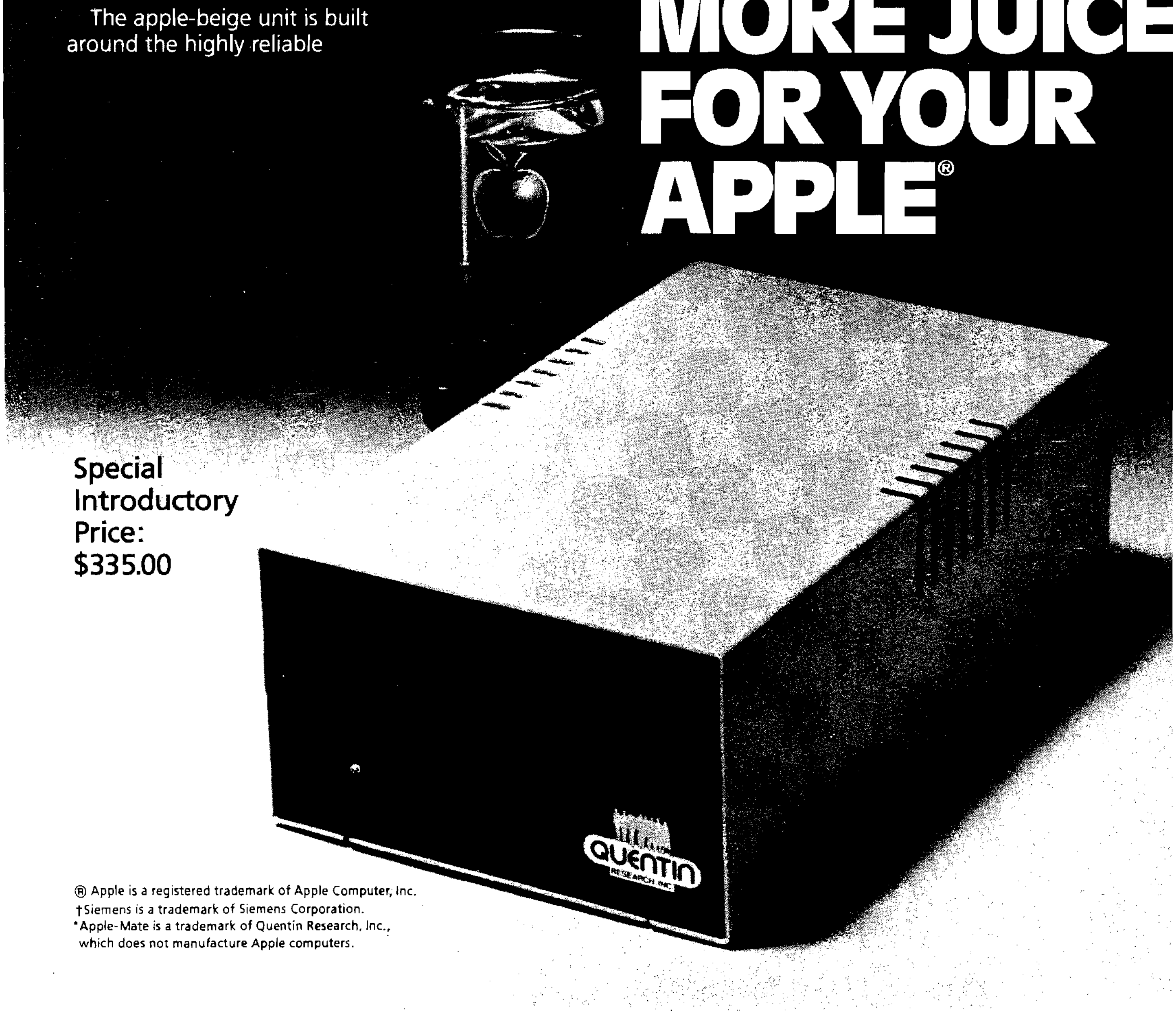
So when you're ready to boost the juice on your Apple, add-on the Quentin Apple-Mate.

To order: Check, money order, Visa or Mastercard number. Calif. residents add 6% sales tax. Allow one week delivery.

QUENTIN RESEARCH, INC.

19355 Business Center Drive
Northridge, California 91324
(213) 701-1006

MORE JUICE FOR YOUR APPLE®



Special Introductory Price: $335.00

® Apple is a registered trademark of Apple Computer, Inc.
†Siemens is a trademark of Siemens Corporation.
*Apple-Mate is a trademark of Quentin Research, Inc., which does not manufacture Apple computers.

Solve the Pagoda Puzzle Using Recursive Assembly

by Sherwood Hoyt

This routine solves the "Pagoda" or "Tower of Hanoi" puzzle, using a recursive subroutine — one that calls itself. Four stacks are maintained for passing parameters.

Pagoda Solver
requires:

6502 computer with 2K (uses 13 page-zero locations, and character OUTPUT and INPUT routines — provided for PET, Apple, and OSI OS-65D).

The Pagoda Puzzle is a game using a rectangular platform with three pegs sticking vertically out of it. The peg sticking out of the left side of the board has a number (usually eight) of discs stacked on it. The discs get smaller as you near the top of the stack (see figure 1). The object is to move the discs, which could be called a tower, from the peg on the left of the board to the peg on the right. There are only two rules to the game; move only one disc at a time, and never put a disc on top of another one so that the larger one is on top.

According to Peter Grogono's book *Programming in Pascal* (where I first read about this game and the algorithm to solve it), the game was accompanied with literature saying that priests in the Temple of Bramah played the game. When they finished their game, it signified the end of the world. Apparently the priests were playing with 64 discs. It would take roughly 18.4 billion billion moves to solve a 64-disc game. According to Peter Grogono, and some rough calculations that I made, it would take a powerful computer about a million years (if it could run that long) just to compute, not to mention print, the moves for a 64-disc game. It would take my OSI computer about 60 million years.

One function that can be used to calculate the number of moves for n discs is ((2^n) − 1). A 4-disc game would take 15 moves; eight discs, 255. The number of moves the game takes as the number of discs goes up is exponential.

Recursion and the Tower

The recursive procedure to move the tower is quite simple; see listing 1. Let's assume that the pegs are numbered from one to three. The tower is stacked on peg 1, and is supposed to be moved to peg 3. The procedure is called initially with four parameters. NMDISC is the number of discs on the tower, FROM is set to peg 1, TO is set to peg 3, and USING is set to peg 2.

and USING receives the value of TO. The same thing is done in the third statement with FROM and USING. The second statement prints a move; a disc is taken off peg FROM and put onto peg TO. The algorithm in this procedure is used in the assembly-language program to solve the puzzle.

The trick to a recursive procedure is in maintaining distinct values for the local variables of the procedure, so that when the procedure calls itself the variables have separate values, though the names of the variables may be the same. To accomplish this, the variable values can be put on a stack (a block of memory pointed to by a stack pointer) in much the same way that return addresses for a JSR are put on the 6502 stack. In a procedure, this is done to be

Figure 1

```
          I                   I                   I
          I                   I                   I
          I                   I                   I
         DDD                  I                   I
        DDDDD                 I                   I
       DDDDDDD                I                   I
      DDDDDDDDD               I                   I
     DDDDDDDDDDD              I                   I
    DDDDDDDDDDDDD             I                   I
   DDDDDDDDDDDDDDD            I                   I
  DDDDDDDDDDDDDDDDD           I                   I
-------------------------------------------------------------
I          THE   PAGODA   PUZZLE                            I
-------------------------------------------------------------
```

The first operation of the procedure is to check the number of discs left on the tower. If the number is not greater than zero, control returns to the calling procedure. If the condition is true (NMDISC > 0), then the three statements between the IF...ENDIF are executed. The first statement is a recursive call. You can see that the number of discs passed is equal to the current number of discs minus 1, and parameters TO and USING are passed so that TO receives the value of USING

able to pull the values out of the "deep freeze" when control returns to the procedure. When a procedure is called, and there may be indefinitely many nested calls, its values are pushed on top of the stack. For the program to remember where the top of the stack is, a pointer is used that always points to the top. In my assembly-language program, the pointer is register Y, and it is incremented before a value is pushed to the stack, and decremented to pop a value off of the stack.

Listing 1: Recursive Procedure In Pseudo-Code

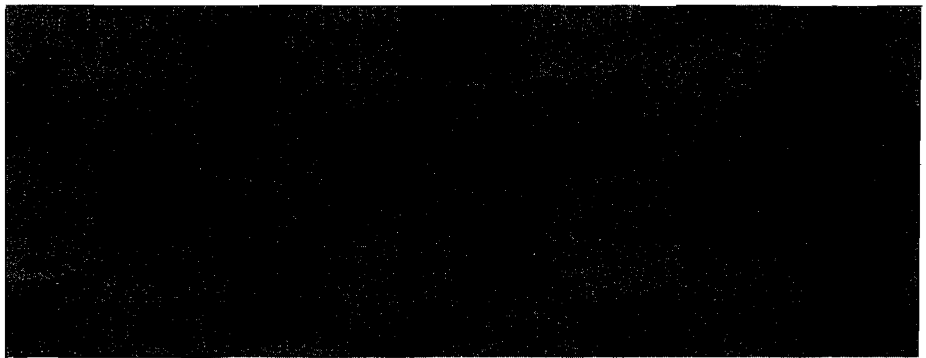
```
PROCEDURE MOVE RECEIVE
(NMDISC,FROM,TO,USING)

  IF NMDISC > 0 THEN

    CALL MOVE WITH(NMDISC – 1,
    FROM,USING,TO)
    PRINT FROM," → ",TO
    CALL MOVE WITH(NMDISC – 1,
    USING,TO,FROM)

  ENDIF

END PROCEDURE
```

The Recursive Assembly-Language Program

In the assembly-language program (listing 2), there are four zero-page locations (lines 130-160) for passing values to the recursive procedure labeled MOVE (lines 530-820). Values are passed from recursive procedural calls *via* these locations, as well as initially from the calling routine. The calling routine (lines 470-490) jumps to lines 1590-2360 to initialize the parameters and stack pointers. Register Y is used as the stack index, and is initialized to $FF. When MOVE is called, register Y is incremented before the values passed are actually pushed on the stack, so that the procedure, when first called, will increment register Y to $00, then push the values on the stack indexed by register Y.

The stack pointers are initialized next. There are four stacks, one stack for each value that is passed. When the procedure is called, register Y is incremented so that it points to a new stack position. The total number of bytes used in each stack for any given game is equal to the number of discs on the tower. Each stack occupies one page ($00-$FF) of memory. Actually, the stack size only needs to be one half of a page because the number of JSRs that can be executed, and therefore the number of recursive calls and separate values, is limited by the 6502's one-page stack, which can only hold $80 two-byte return addresses. The largest tower that this program can move has approximately $70 discs on it, which allows part of the stack to have been used already by the calling routine. However, having a maximum game of $70 discs is obviously no problem if it would take a computer a million years just to solve a $40 (decimal 64) disc game!

After the initialization routine stores zero in the location that holds



Listing 2

```
            0010 ;********************************
            0020 ;*                              *
            0030 ;*      PAGODA  SOLVER          *
            0040 ;*                              *
            0050 ;*     by Sherwood Hoyt         *
            0060 ;*                              *
            0070 ;********************************
            0080 ;
            0090 ;--- parameters passed ---
            0100 ;
            0110            .ba $54               ;PET
            0120 ;OS-65D--$10; Apple--$f0
            0130 nmdisc     .de =
            0140 from       .de nmdisc+1
            0150 to         .de from+1
            0160 using      .de to+1
            0170 ;
            0180 ;--- stack ptr's to local values ---
            0190 ;
            0200 nptr       .de using+1
            0210 fptr       .de nptr+2
            0220 tptr       .de fptr+2
            0230 uptr       .de tptr+2
            0240 ;
            0250 ;--- stack location ---
            0260 ;
            0270            .ba $1800             ;$4000--OS-65D; $800--Appl
            0280 nstack     .de =
            0290 fstack     .de nstack+$100
            0300 tstack     .de fstack+$100
            0310 ustack     .de tstack+$100
            0320 ;
            0330 ;--- miscellaneous ---
            0340 ;
            0350 nmoves     .de uptr+2            ;current number of moves
            0360 input      .de $ffcf             ;get character
            0370 ;Apple--$fd0c; OS-65D--$2339
            0380 output     .de $ffd2             ;output character
            0390 ;Apple--$fded; OS-65D--$2343
            0400 return     .de $0d
            0410 lf         .de $0a
            0420 ;
            0430            .ba ustack+$100       ;OS-65D--$317e
            0440 ;
            0450 ;
            0460 ;--- call major subroutines ---
            0470 ;
1C00- 20 BB 1C  0480            jsr init              ;init. ptr's & parameters
1C03- 20 09 1C  0490            jsr move              ;call proc. to move tower
1C06- 4C 70 1D  0500            jmp done
            0510 ;
            0520 ;--- recursive procedure to move tower ---
            0530 ;
1C09- 20 A9 1C  0540 move       jsr push              ;push values to stack
1C0C- B1 58     0550            lda (nptr),y          ;get # of discs left
1C0E- F0 2D     0560            beq ret               ;if = 0 then pop procedure
            0570 ;
1C10- B1 58     0580            lda (nptr),y          ;get values to be passed
1C12- 85 54     0590            sta *nmdisc
1C14- C6 54     0600            dec *nmdisc           ;de. # of diiscs on tower
1C16- B1 5A     0610            lda (fptr),y
1C18- 85 55     0620            sta *from
1C1A- B1 5C     0630            lda (tptr),y
1C1C- 85 57     0640            sta *using            ;store 'to' in 'using'
1C1E- B1 5E     0650            lda (uptr),y
1C20- 85 56     0660            sta *to               ;store 'using' in 'to'
1C22- 20 09 1C  0670            jsr move
            0680 ;
1C25- 20 3F 1C  0690            jsr prmove            ;display a move
            0700 ;
1C28- B1 58     0710            lda (nptr),y          ;pass values to next proc.
1C2A- 85 54     0720            sta *nmdisc
1C2C- C6 54     0730            dec *nmdisc           ;dec. # of discs on tower
1C2E- B1 5A     0740            lda (fptr),y
1C30- 85 57     0750            sta *using            ;store 'from' in 'using'
1C32- B1 5C     0760            lda (tptr),y
1C34- 85 56     0770            sta *to
1C36- B1 5E     0780            lda (uptr),y
1C38- 85 55     0790            sta *from             ;store 'using' in 'from'
1C3A- 20 09 1C  0800            jsr move              ;do recursive call
            0810 ;
1C3D- 88        0820 ret        dey                   ;pop procedure from stack
1C3E- 60        0830            rts
```

(continued)

***Figure 2:* Sample Runs**

```
ENTER TOWER SIZE: 3

0001   1 → 3
0002   1 → 2
0003   3 → 2
0004   1 → 3
0005   2 → 1
0006   2 → 3
0007   1 → 3

*** DONE ***


ENTER TOWER SIZE: 4

0001   1 → 2
0002   1 → 3
0003   2 → 3
0004   1 → 2
0005   3 → 1
0006   3 → 2
0007   1 → 2
0008   1 → 3
0009   2 → 3
000A   2 → 1
000B   3 → 1
000C   2 → 3
000D   1 → 2
000E   1 → 3
000F   2 → 3

*** DONE ***
```

the current number of moves (lines 1830-1850), it stores the initial values for the FROM, USING, and TO pegs (lines 1890-1940). Then in lines 1980-2030 the prompt "ENTER TOWER SIZE:" is displayed, and lines 2080-2360 input the user response. The input routine will only accept a one- or two-digit hexadecimal number. If a one-digit number is entered, the return key must follow it. If two digits are entered, the program will continue execution and no return needs to be entered.

The next call that the calling routine makes, after initialization, is to the recursive procedure; MOVE. The first operation is to push the variables it receives onto the stack. This allows the variables for each successive calling of the procedure to be maintained separately. When the procedure is finished, the stack index will be decremented (line 810), effectively popping the variables off of the stack. The only other thing done with the variables is for the procedure to call itself with the variables (in slightly modified form) as parameters. This the procedure performs twice. Also, between the two recursive calls, the procedure calls PRMOVE from line 860-1030 to display a move. The variables FROM and TO are passed to it. FROM is the peg to

Listing 2 *(continued)*

```
                 0840 ;
                 0850 ;--- print a move ---
                 0860 ;
1C3F- 20 6D 1C   0870 prmove      jsr prnum          ;print current # of moves
1C42- B1 5A      0880             lda (fptr),y
1C44- 09 30      0890             ora #$30
1C46- 20 D2 FF   0900             jsr output         ;print peg # to take disc
1C49- A2 01      0910             ldx #$01           ;print ' -> '
1C4B- BD 67 1C   0920 p1          lda p3,x
1C4E- F0 06      0930             beq p2
1C50- 20 D2 FF   0940             jsr output
1C53- E8         0950             inx
1C54- D0 F5      0960             bne p1
1C56- B1 5C      0970 p2          lda (tptr),y
1C58- 09 30      0980             ora #$30
1C5A- 20 D2 FF   0990             jsr output         ;print peg# to put disc on
1C5D- A9 0D      1000             lda #return        ;new line
1C5F- 20 D2 FF   1010             jsr output
1C62- A9 0A      1020             lda #lf
1C64- 20 D2 FF   1030             jsr output
1C67- 60         1040 p3          rts
1C68- 20 2D 3E   1050             .by ' -> ' 00
1C6B- 20 00
                 1060 ;
                 1070 ;--- print current number of moves ---
                 1080 ;
1C6D- E6 60      1090 prnum       inc #nmoves        ;bump # of moves up one
1C6F- D0 02      1100             bne prnum1
1C71- E6 61      1110             inc #nmoves+1
1C73- A5 61      1120 prnum1      lda #nmoves+1
1C75- 4A         1130             lsr a              ;shift high nibble right
1C76- 4A         1140             lsr a
1C77- 4A         1150             lsr a
1C78- 4A         1160             lsr a
1C79- 20 9A 1C   1170             jsr dspbyt         ;display first digit
1C7C- A5 61      1180             lda #nmoves+1
1C7E- 20 9A 1C   1190             jsr dspbyt         ;display second digit
1C81- A5 60      1200             lda #nmoves
1C83- 4A         1210             lsr a              ;shift high nibble right
1C84- 4A         1220             lsr a
1C85- 4A         1230             lsr a
1C86- 4A         1240             lsr a
1C87- 20 9A 1C   1250             jsr dspbyt         ;display third digit
1C8A- A5 60      1260             lda #nmoves
1C8C- 20 9A 1C   1270             jsr dspbyt         ;display fourth digit
1C8F- A9 20      1280             lda #$20           ;output 2 blanks
1C91- 20 D2 FF   1290             jsr output
1C94- A9 20      1300             lda #$20
1C96- 20 D2 FF   1310             jsr output
1C99- 60         1320             rts
                 1330 ;
                 1340 ;--- convert byte to ascii and display
                 1350 ;
1C9A- 29 0F      1360 dspbyt      and #$0f           ;clear high nibble
1C9C- 09 30      1370             ora #$30           ;convert to ascii
1C9E- C9 3A      1380             cmp #$3a           ;check for 'a'-'f'
1CA0- 90 03      1390             bcc dspb1          ;branch if not
1CA2- 18         1400             clc
1CA3- 69 07      1410             adc #$07
1CA5- 20 D2 FF   1420 dspb1       jsr output         ;display digit
1CA8- 60         1430             rts
                 1440 ;
                 1450 ;--- push procedure values on stacks ---
                 1460 ;
1CA9- C8         1470 push        iny                ;bump index for new values
1CAA- A5 54      1480             lda #nmdisc        ;push # of discs to stack
1CAC- 91 58      1490             sta (nptr),y
1CAE- A5 55      1500             lda #from          ;push 'from' peg to stack
1CB0- 91 5A      1510             sta (fptr),y
1CB2- A5 56      1520             lda #to            ;push 'to' peg to stack
1CB4- 91 5C      1530             sta (tptr),y
1CB6- A5 57      1540             lda #using         ;push 'using' peg to stack
1CB8- 91 5E      1550             sta (uptr),y
1CBA- 60         1560             rts
                 1570 ;
                 1580 ;--- initialize stack ptr's ---
                 1590 ;
1CBB- A0 FF      1600 init        ldy #$ff           ;set stack index
                 1610 ;gets bumped to 0 for 1st values
1CBD- A9 00      1620             lda #l,nstack      ;set 'nmdisc' stack ptr.
1CBF- 85 58      1630             sta #nptr
1CC1- A9 18      1640             lda #h,nstack
1CC3- 85 59      1650             sta #nptr+1
                 1660 ;
1CC5- A9 00      1670             lda #l,fstack      ;set 'from' stack ptr.
1CC7- 85 5A      1680             sta #fptr
1CC9- A9 19      1690             lda #h,fstack
1CCB- 85 5B      1700             sta #fptr+1
                 1710 ;
1CCD- A9 00      1720             lda #l,tstack      ;set 'to' stack ptr.
1CCF- 85 5C      1730             sta #tptr
1CD1- A9 1A      1740             lda #h,tstack
1CD3- 85 5D      1750             sta #tptr+1
                 1760 ;
1CD5- A9 00      1770             lda #l,ustack      ;set 'using' stack ptr.
1CD7- 85 5E      1780             sta #uptr
1CD9- A9 1B      1790             lda #h,ustack
1CDB- 85 5F      1800             sta #uptr+1
                 1810 ;
                 1820 ;--- set number of moves to zero ---
                 1830 ;
```

(continued)

Listing 2 *(continued)*

```
1CDD- A9 00     1840            lda #$00
1CDF- 85 60     1850            sta #nmoves
1CE1- 85 61     1860            sta #nmoves+1
                1870 ;
                1880 ;--- initialize beginning parameters ---
                1890 ;
1CE3- A9 01     1900            lda #$01
1CE5- 85 55     1910            sta #from          ;set 'from' to peg #1
1CE7- A9 03     1920            lda #$03
1CE9- 85 56     1930            sta #to            ;set 'to' to peg #3
1CEB- A9 02     1940            lda #$02
1CED- 85 57     1950            sta #using         ;set 'using' to peg #2
                1960 ;
                1970 ;--- display prompt ---
                1980 ;
1CEF- A2 00     1990 prompt     ldx #$00
1CF1- BD FC 1C  2000            lda pro3,x
1CF4- F0 1C     2010            beq enter
1CF6- 20 D2 FF  2020            jsr output
1CF9- E8        2030            inx
1CFA- D0 F5     2040            bne prompt+2
1CFC- 0A 0A 0D  2050 pro3       .by lf lf return 'ENTER TOWER SIZE: ' 00
1CFF- 45 4E 54
1D02- 45 52 20
1D05- 54 4F 57
1D08- 45 52 20
1D08- 53 49 5A
1D0E- 45 3A 20
1D11- 00
                2060 ;
                2070 ;--- input number of discs on tower ---
                2080 ;
1D12- 20 CF FF  2090 enter      jsr input
1D15- C9 0D     2100            cmp #return
1D17- F0 2B     2110            beq ent3           ;done inputting
1D19- 20 54 1D  2120            jsr chno
1D1C- B0 F4     2130            bcs enter          ;not a hex digit
1D1E- 20 D2 FF  2140            jsr output         ;display valid digit
1D21- 20 66 1D  2150            jsr strip          ;strip ascii from digit
1D24- 85 54     2160            sta #nmdisc        ;store first digit
1D26- 20 CF FF  2170 ent2       jsr input
1D29- C9 0D     2180            cmp #return
1D2B- F0 17     2190            beq ent3           ;done inputting
1D2D- 20 54 1D  2200            jsr chno
1D30- B0 F4     2210            bcs ent2           ;not a hex digit
1D32- 20 D2 FF  2220            jsr output         ;display valid digit
1D35- 20 66 1D  2230            jsr strip          ;strip ascii from digit
1D38- 06 54     2240            asl #nmdisc        ;sift 1st digit left
1D3A- 06 54     2250            asl #nmdisc
1D3C- 06 54     2260            asl #nmdisc
1D3E- 06 54     2270            asl #nmdisc
1D40- 05 54     2280            ora #nmdisc        ;pack 2nd digit with 1st
1D42- 85 54     2290            sta #nmdisc        ;store it
1D44- A9 0A     2300 ent3       lda #lf            ;space 2 lines down
1D46- 20 D2 FF  2310            jsr output
1D49- A9 0A     2320            lda #lf
1D48- 20 D2 FF  2330            jsr output
1D4E- A9 0D     2340            lda #return
1D50- 20 D2 FF  2350            jsr output
1D53- 60        2360            rts
                2370 ;
                2380 ;--- check for valid hex number ---
                2390 ;
1D54- C9 30     2400 chno       cmp #'0
1D56- 90 0C     2410            bcc invlid
1D58- C9 3A     2420            cmp #':
1D5A- 90 09     2430            bcc valid
1D5C- C9 41     2440            cmp #'A
1D5E- 90 04     2450            bcc invlid
1D60- C9 47     2460            cmp #'G
1D62- 90 01     2470            bcc valid
1D64- 38        2480 invlid     sec
1D65- 60        2490 valid      rts
                2500 ;
                2510 ;strip ascii from input
                2520 ;
1D66- C9 41     2530 strip      cmp #'A
1D68- 90 03     2540            bcc st3
1D6A- 18        2550            clc
1D6B- 69 09     2560            adc #$09           ;add 9 if digit 'a'-'f'
1D6D- 29 0F     2570 st3        and #$0f           ;strip ascii
1D6F- 60        2580            rts
                2590 ;
                2600 ;--- finished moving tower ---
                2610 ;
1D70- A2 01     2620 done       ldx #$01           ;print that it is done
1D72- 8D 7D 1D  2630            lda do3,x
1D75- F0 06     2640            beq do3
1D77- 20 D2 FF  2650            jsr output
1D7A- E8        2660            inx
1D7B- D0 F5     2670            bne done+2
1D7D- 60        2680 do3        rts                ;return to operating syste
                2690 ;
1D7E- 0A 0A 0D  2700            .by lf lf return '*** DONE *** lf return 00
1D81- 2A 2A 2A
1D84- 20 44 4F
1D87- 4E 45 20
1D8A- 2A 2A 2A
1D8D- 0A 0D 00
                2710 ;
                2720            .en
```

take the disc from, and TO is the peg to put the disc onto.

Trial Run

In figure 2 there are two trial runs. The first one solves the puzzle for three discs and the second for four discs. Remember, the number of discs that you enter at the prompt 'ENTER TOWER SIZE:' is assumed to be a hexadecimal number. A carriage return is required if the number entered has only one digit. [Ed. Note: The routine is written to echo the character received to the screen. If your machine's INPUT routine does this automatically, the character will appear twice on the screen. You may want to skip the echoes in lines 2130 and 2210.] A four-digit hexadecimal number in the first column of output shows the number of moves. This number will turn over to '0000' after it reaches 'FFFF'. The number in the second column tells which peg to take the disc off of and the number in the fourth column tells which peg to put the disc onto.

The author may be contacted at Rt. 1, Box 56, Sand Springs, OK 74063.

MICRO

VIC/PET GOMOKU

by David Malmberg

A fast, machine-language version of the popular oriental game, with three user-selectable styles for the computer's play.

GOMOKU
requires:
VIC with 3K or more extra RAM and joystick
or PET with 8K or more

GOMOKU is an ancient oriental game of strategy. According to Edward Lasker's book *Go and GOMOKU* (Dover, 1960), GO means "five" and MOKU means "stones." To play, you and your opponent alternate placing stones on the intersections of the lines of a square grid. One person plays with white stones and the other uses black stones. As the name implies, the object of the game is to have your stones occupy five adjacent points on a grid — vertically, horizontally, or diagonally — before your opponent gets five of his stones in a row. You must get exactly five in a row to win. Connecting two chains of stones less than five long to form a chain greater than five stones long will not win the game.

The most obvious way a player can force a win is to create a situation where he has four stones in a row with both ends of the chain open for potential stones. The opponent will be able to block only one end of the chain, so the player can win by placing a stone at the other end. To reach a forced win with an "open-four," the player must first create an "open-three." Whenever a player threatens a win by making an open-three, the opponent must immediately play a stone at one end of this chain of three in order to avoid losing the game two moves later.

As GOMOKU is played in both Japan and China, only one type of move is illegal: a participant cannot play a stone that creates more than one open-three simultaneously. The oriental originators of GOMOKU felt that the game would not be enough of a challenge if this type of easy win were permitted.

VIC/PET GOMOKU

The computerized version of GOMOKU allows the enforcement of the no-multiple open-threes rule to be optional. However, remember that if you elect to play without this restriction, the computer will play accordingly.

VIC/PET GOMOKU plays an excellent game. Its main counting and evaluation routines are written in machine language for increased speed and enhanced logic. The program takes 10 to 30 seconds between moves, depending on the density of stones on the

Listing 1

```
1 RL=22:S=256*PEEK(648)-1:A=30720:IFPEEK(648)=16THENA=33792
2 FH=RND(-TI):POKE36879,25:DD=37154:P1=37151:P2=37152:V=36878:E=36876
3 FORI=1TO8:READX:POKE951+I,X:NEXTI:DATA1,21,22,23,129,149,150,151:FH=10
4 ML=PEEK(43)+256*PEEK(44)+3839:B=ML/256:DIMD(8),V%(14,14)
5 FORI=1TO13:READX:POKEML+X,B:NEXT
6 DATA2,22,51,54,75,91,104,125,136,161,170,215,256
7 B=B+1:FORI=1TO4:READX:POKEML+X,B:NEXT:DATA19,306,351,384
8 MV$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒":WH$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
9 FORI=1TO8:READD(I):NEXT:DATA-22,-21,1,23,22,21,-1,-23
10 H=87:C=81:NP$="C":M=0:W=0:PRINT"▒   ▒MICRO   GOMOKU":IN$="N"
11 PRINT"▒  BY DAVID MALMBERG"
12 PRINT"▒▒WANT TO MOVE FIRST ";:GOSUB85:PRINT
13 IFLEFT$(IN$,1)="Y"THENNP$="H"
14 IN$="W":PRINT"WHITE OR BLACK ";:GOSUB85:PRINT:IFIN$="B"THENH=81:C=87
15 POKE950,C:POKE951,H:IN$="2":PRINT"▒HOW SHOULD I PLAY:":PRINT"  1: CAUTIOUS"
16 PRINT"  2: BALANCED":PRINT"  3: AGGRESSIVE ";
17 GOSUB85:PRINT:X=VAL(LEFT$(IN$,1))
18 IFX>3ORX<1THENPRINT:PRINT"1,2, OR 3 ONLY ";:GOTO17
19 ZZ=0:PRINT"▒PLAY OPEN 3'S RULE ";:GOSUB85:PRINT:IFIN$<>"Y"THENZZ=1
20 FC=6+2*X:N=8:G=7:MT=14*14:FORI=1TO14:FORJ=1TO14:V%(I,J)=0:NEXTJ:NEXTI:O=1
21 PRINTCHR$(28)"▒   ▒MICRO   GOMOKU"
22 PRINTTAB(O+2)"▒ABCDEFGHIJKLMN":PRINTTAB(O)"▒1  ┬┬┬┬┬┬┬┬┬┬┬┬┬┬":FORI=2TO13
23 PRINTTAB(O-1)RIGHT$(STR$(I),2);"  ┼┼┼┼┼┼┼┼┼┼┼┼┼┼"
24 NEXTI:PRINTTAB(O-1)"14  ┴┴┴┴┴┴┴┴┴┴┴┴┴┴":PRINTTAB(O+3);"▒MY MOVE:"
25 PRINTCHR$(144);WH$;CHR$(128+H);"▒▒▒▒▒▒"CHR$(128+C)
26 IFNP$<>"H"THEN31
27 PRINTCHR$(30);WH$;"▒▒YOU▒▒▒▒▒▒▒▒ME":T=H:POKEV,15:FORK=195TO240:U=TI
28 POKEE,K:IFTI<U+1THEN28
29 NEXT:POKEE,0:GOSUB96:M=M+1:V%(I,J)=-30000:MT=MT-1:IFW=HTHEN76
30 IFMT=0THEN75
31 PRINTCHR$(30);WH$;"▒YOU▒▒▒▒▒▒▒▒ME▒":IFM=0THENI=7:J=8:GOTO36
32 QM=C:GOSUB46:MX=-1E15:FORMI=1TO14:FORMJ=1TO14:IFV%(MI,MJ)<MXTHEN35
33 IFV%(MI,MJ)=MXANDRND(1)>.2THEN35
34 XJ=MJ+O+2:XI=MI+4:L=S+RL*(XI-1)+XJ:MX=V%(MI,MJ):I=MI:J=MJ
35 NEXTMJ:NEXTMI
36 B$="":IFI<10THENB$=" "
37 PRINTCHR$(28);MV$TAB(O+12);B$;CHR$(J+64)MID$(STR$(I),2):N=I:G=J:CL=J+O+2
38 R=I+4:L=S+RL*(R-1)+CL
39 P=PEEK(L):POKEV,15:FORK=1TO6:POKEL,P:U=TI:POKEE,135
40 IFTI<U+8THEN40
41 POKEL,C:POKEL+A,0:U=TI:POKEE,240
42 IFTI<U+8THEN42
43 NEXTK:POKEV,0:POKEE,0:T=C:GOSUB68:MT=MT-1:IFW=CTHEN76
44 IFMT=0THEN75
45 QM=H:GOSUB46:V%(I,J)=-30000:GOTO27
46 II=I:JJ=J:POKE991,QM:FORIN=-1TO1:FORJN=-1TO1:IFIN=0ANDJN=0THENNEXTJN
47 FORKN=1TO4:I=II+KN*IN:J=JJ+KN*JN:IFI<1ORI>14THENKN=4:GOTO67
48 IFJ<1ORJ>14THENKN=4:GOTO67
49 CL=J+O+2:R=I+4:L=S+RL*(R-1)+CL:P=PEEK(L):IFP=CORP=HTHEN67
50 POKEL,P+128:V%(I,J)=0:LH=INT(L/256):LL=L-LH*256:POKE962,LL:POKE963,LH
51 POKEL,P:SYS(ML):ZP=PEEK(973):IFZP=0THEN55
52 FORY=1TOZP:NC=PEEK(974+Y):NH=PEEK(982+Y):F=FH:IFNH=0THENNH=NC:F=FC
53 V%(I,J)=V%(I,J)+INT(F*NH*NH):NEXTY:NE=PEEK(993)
54 IFNE<>0THENV%(I,J)=V%(I,J)-NE*NE
55 NH=0:IFPEEK(992)=0THEN67
56 FORY=1TO4:Q=0:Z=D(Y):T=PEEK(L+Z):IFT<>PEEK(L-Z)THEN66
57 IFT=HORT=CTHEN59
58 GOTO66
```

(continued)

Listing 1 *(continued)*

```
59 Z=D(Y):GOSUB91:IFPEEK(L)<>TTHENQ=Q-1
60 IFQ>4THENQ=0:GOTO66
61 IFQ<4THEN64
62 IFT=CTHENQ=2
63 Q=Q+11:GOTO66
64 IF(N1=HORN1=C)ANDN1=N2THENQ=0
65 IFQ=3AND(N1<>HANDN1<>C)AND(N2<>HANDN2<>C)THENQ=8
66 NH=NH+Q:NEXTY:IFNH>0THENV%(I,J)=INT(FH*NH*NH)+V%(I,J)
67 NEXTKN:NEXTJN:NEXTIN:I=II:J=JJ:RETURN
68 N3=0:FORY=1TO4:Z=D(Y):GOSUB91
69 IFQ=3THENIFN1<>CANDN1<>32ANDN2<>CANDN2<>32THENN3=N3+1
70 IFQ=5THENW=T:GOTO72
71 NEXTY
72 IFQ=5ANDW=HTHENFH=FH*1.2
73 IFZZTHENN3=0
74 RETURN
75 PRINTMV$"▓STALEMATE":GOTO83
76 IFNP$="C"ANDW=CTHENM=M+1
77 IFW=HTHENPRINTMV$"▓YOU WON IN"M" MOVES"
78 IFW=CTHENPRINTMV$"▓I WON IN"M" MOVES"
79 IFM<10THENPRINT"GOOD GAME!"
80 IFM>9ANDM<20THENPRINT"CLOSE GAME!"
81 IFM>19ANDM<30THENPRINT"GREAT GAME!"
82 IFM>29THENPRINT"FANTASTIC GAME!"
83 PRINT"WANT TO PLAY AGAIN";:GOSUB85:IFLEFT$(IN$,1)="Y"THEN10
84 END
85 ZC=1:ZT=0
86 GETZ$:IFZ$<>""THEN89
87 IFTI>=ZTTHENPRINTMID$("? ",ZC,1);"▌";:ZT=TI+30:ZC=3-ZC
88 GOTO86
89 IFZ$<>CHR$(13)THENIN$=Z$
90 PRINT"? ";IN$;:RETURN
91 Q=1:LN=L+Z
92 IFPEEK(LN)=TTHENQ=Q+1:LN=LN+Z:GOTO92
93 N1=PEEK(LN):LN=L-Z
94 IFPEEK(LN)=TTHENQ=Q+1:LN=LN-Z:GOTO94
95 N2=PEEK(LN):RETURN
96 XJ=G+O+2:XI=N+4:L=S+RL*(XI-1)+XJ:P=PEEK(L):FS=T:IFFS=PTHENFS=128+FS
97 POKEL,FS:FZ=P:ZT=TI+9
98 POKEE,0:DX=0:DY=0:FB=0:GOSUB112:IFDXORDYORFBTHEN101
99 IFTI<ZTTHEN98
100 Z=FZ:FZ=PEEK(L):POKEL,Z:ZT=TI+9:GOTO98
101 IFFBTHEN109
102 POKEL,P:N=N+DY:G=G+DX:POKEE,135:U=TI
103 IFTI<U+9THEN103
104 IFN>14THENN=1
105 IFN<1THENN=14
106 IFG>14THENG=1
107 IFG<1THENG=14
108 GOTO96
109 IFP=CORP=HTHEN98
110 GOSUB68:IFN3>1THEN98
111 POKEL,T:POKEL+A,0:POKEV,0:POKEE,0:I=N:J=G:RETURN
112 POKEDD,127:B=PEEK(P2)AND128:J0=-(B=0):POKEDD,255:B=PEEK(P1)
113 J2=-((BAND16)=0):J3=-((BAND4)=0):FB=-((BAND32)=0):J1=-((BAND8)=0)
114 IFJ0THENDX=1
115 IFJ1THENDY=1
116 IFJ2THENDX=-1
117 IFJ3THENDY=-1
118 RETURN
READY.
```

grid. By comparison, the first version of GOMOKU I wrote was entirely in BASIC and took from three to six minutes between moves — and it did not play half as well as the current version. VIC/PET GOMOKU should be capable of holding its own against even very good human players — especially if the computer moves first.

VIC/PET GOMOKU has three styles of play: cautious, balanced, and aggressive. These styles refer to how much the computer will weigh defensive *vs.* offensive moves in its evaluation of the merits of its next move. If you select a balanced style, both offense and defense have equal weights. You will find the computer will win more games when it is playing aggressively. However, when you select the cautious style for the computer, you may not be able to win often because many games will end up in stalemate; the computer will unrelentlessly block your chances for any kind of a winning pattern. When the computer loses a game, it will automatically play a more aggressive game the next time.

The VIC Version

The VIC version of GOMOKU needs at least 3K of additional memory. This can be obtained by using any of the standard VIC Super Expander, 3K, 8K, or 16K memory cartridges.

The program is written in two parts. Listing 1, the main part, is written in BASIC and should be keyed-in exactly as shown in the listing (i.e., with no unnecessary spaces to conserve memory). Listing 2, a hex-dump of the machine-language routines, shows the machine code being located from \$1300 to \$14AB. These locations assume you are using either the Super Expander or the 3K memory cartridges, which cause the start of BASIC to be located at \$0400 (just like the PET). If you are using either the 8K or 16K memory expanders, this machine code should be located from \$2100 to \$22AB because the start of BASIC is relocated to \$1200 by the VIC's operating system. When you use a monitor to enter and save the code in the hex listing, there is no need to worry about relocating the two-byte addresses since lines 4-6 of the BASIC program automatically adjust them to correspond to the memory configuration of your VIC.

To create a working copy of GOMOKU for the VIC, first load the BASIC portion (listing 1). Then load the machine-language code you had previously saved with the monitor by

Listing 2

```
.:1300 20 ED 13 A9 00 85 5C 85
.:1308 5D 85 48 85 49 A5 48 85
.:1310 42 20 02 14 20 4D 13 A5
.:1318 44 C9 00 F0 0C A6 49 A5
.:1320 3C 95 4B A5 3D 95 53 E6
.:1328 49 A5 48 C9 07 F0 05 E6
.:1330 48 4C 0D 13 20 38 13 60
.:1338 A2 00 B5 32 9D B6 03 BD
.:1340 84 03 95 32 E0 2B F0 04
.:1348 E8 4C 3A 13 60 A5 3C C9
.:1350 03 F0 09 A5 3D C9 03 F0
.:1358 10 4C 89 13 A5 45 C5 33
.:1360 D0 07 A9 00 85 3D 4C C5
.:1368 13 A5 45 C5 33 F0 1A C5
.:1370 32 F0 16 A5 5B C5 4A D0
.:1378 05 A9 0A 4C 80 13 A9 08
.:1380 85 3C A9 00 85 3D 4C C5
.:1388 13 A5 3C C9 04 D0 1C A5
.:1390 46 C5 32 F0 0D A5 47 C5
.:1398 32 F0 07 A9 0D 85 3C 4C
.:13A0 C5 13 A9 00 85 3C 85 3D
.:13A8 4C C5 13 A5 3D C9 04 D0
.:13B0 14 A5 46 C5 33 F0 EB A5
.:13B8 47 C5 33 F0 E5 A9 0B 85
.:13C0 3C A9 00 85 3D A9 00 85
.:13C8 44 A5 3C C9 00 D0 09 A5
.:13D0 3D C9 00 D0 09 4C E2 13
.:13D8 A5 3D C9 00 D0 04 A9 01
.:13E0 85 44 A5 4A C5 47 D0 04
.:13E8 A9 01 85 5C 60 A2 00 B5
.:13F0 32 9D 84 03 BD B6 03 95
.:13F8 32 E0 2B F0 04 E8 4C EF
.:1400 13 60 A5 3E 85 40 A5 3F
.:1408 85 41 A6 42 B5 34 85 43
.:1410 A2 00 86 3C 86 3D 86 45
.:1418 86 46 86 47 86 4A A5 43
.:1420 38 C9 80 B0 0E A5 40 38
.:1428 E5 43 B0 02 C6 41 85 40
.:1430 4C 3E 14 E9 80 18 65 40
.:1438 90 02 E6 41 85 40 A0 00
.:1440 B1 40 85 47 A5 3E 85 40
.:1448 A5 3F 85 41 E8 A5 43 38
.:1450 C9 80 B0 0C 18 65 40 90
.:1458 02 E6 41 85 40 4C 6F 14
.:1460 E9 80 85 44 A5 40 38 E5
.:1468 44 B0 02 C6 41 85 40 A0
.:1470 00 B1 40 C9 20 D0 0A 38
.:1478 E0 03 B0 2F E6 5D 4C AB
.:1480 14 C5 32 D0 02 E6 3C C5
.:1488 33 D0 02 E6 3D E0 01 D0
.:1490 02 85 4A E0 04 D0 02 85
.:1498 45 E0 05 D0 AF 85 46 C5
.:14A0 32 D0 02 C6 3C C5 33 D0
.:14A8 02 C6 3D 60 AA AA AA AA
```

using the command LOAD "name", 1,1. The 1's at the end of the LOAD will load the code at its absolute location (i.e., $1300-$14AB or $2100-$22AB), rather than automatically being relocated to the start of BASIC. After both parts have been loaded, issue the command SAVE "VIC GOMOKU" and you will save a copy that combines both parts and can be used independently of whatever memory configuration is operating in the VIC.

The VIC version of GOMOKU uses the joystick to indicate your move. Once you have moved the cursor to your desired location, just hit the fire-button.

The PET Version

The machine-language code in the hex-dump (listing 2) will work without any changes in the PET. However, several lines of the BASIC program must be changed. Specifically, listing 3 contains all the lines that must be changed in listing 1 to get GOMOKU to work on the PET. In addition to these differences, lines 7, 116, 117, and 118 of listing 1 should not be included in the PET version. The BASIC instructions should be typed without any unnecessary spaces to assure that the BASIC part does not run into the machine-code part. If you have an 80-column screen, change RL in line 1 to 80, the DATA values in line 3 to 1, 79, 80, 81, 129, 207, 208, 209, and the DATA values in line 9 to −80, −79, 1, 81, 80, 79, −1, −81.

To create a working PET copy of GOMOKU, first load the BASIC part, then load the machine code previously saved by the monitor (it will always be located from $1300 to $14AB in the PET version), then issue the command SAVE "PET GOMOKU" to save a copy that combines both parts.

The PET version uses the numeric keyboard to indicate your move. Once the cursor is where you wish to move, just hit RETURN to enter the move.

Listing 3

```
1 RL=40:S=32767:E=12*4096:V=E:A=5*4096
2 FH=RND(-TI)
3 FORI=1TO8:READX:POKE951+I,X:NEXTI:DATA1,39,40,41,129,167,168,169:FH=10
4 ML=PEEK(40)+256*PEEK(41)+3839:DIMD(8),V%(14,14),DR(9),DC(9)
5 FORI=1TO9:READDR(I),DC(I):NEXTI
6 DATA1,-1,1,0,1,1,0,-1,0,0,0,1,-1,-1,-1,0,-1,1
9 FORI=1TO8:READD(I):NEXT:DATA-40,-39,1,41,40,39,-1,-41
10 H=81:C=87:NP$="C":M=0:W=0:PRINT"▯    ▮MICRO    GOMOKU":IN$="N"
14 IN$="W":PRINT"WHITE OR BLACK ";:GOSUB85:PRINT:IFIN$="B"THENH=87:C=81
112 GETF$:IFF$=""THENRETURN
113 IFF$=CHR$(13)THENF8=1:RETURN
114 IFF$<"1"ORF$>"9"THENRETURN
115 QQ=ASC(F$)-48:DX=DC(QQ):DY=DR(QQ):RETURN
```

Acknowledgement

I published an earlier version of PET GOMOKU in the December, 1979 issue of *CURSOR* (Box 550, Goleta, CA 93116). That version benefitted from many improvements suggested by Glen Fisher. To the extent that his suggestions survived to the current VIC/PET version, his help is gratefully acknowledged.

The text for this article was submitted by the author as a WordPro file. The edited file was then transmitted to our Compugraphic through our FOCUS CompuPlus system.

MICRO

Number Shuffle on the Atari

by Frank Roberts

A computer version of the popular game Magic Square.

Number Shuffle
requires:
Atari 400 or 800, 8K

In the days of yore, before Rubik's Cube, there existed a two-dimensional puzzle called Magic Square. It was played by rearranging a random set of numbers within a four-by-four matrix until the numbers were in numerical ascending or descending order.

This program is a computer simulation of that puzzle, and is built around a six-by-six matrix. All positions surrounding the central matrix are set to a value of −1 and used only for comparison and validation of moves. One position of the four-by-four center matrix, set to zero, is graphically represented by a blank space. Only the numbers horizontally or vertically adjacent to the space may be moved into that vacant space.

The game begins after the numbers in the NBR array are shuffled 100 times (lines 210-250) and set into the playfield array, BRD (300-370). Once the board is displayed on the screen, the user is prompted for a choice of numbers to move into the empty space. If the number is valid, the number and the space are switched. The X and Y arrays hold the screen coordinates for each number in the central matrix. The switch is made in lines 530-560 and subroutine 20-25. Lines 600-650 check the board after every switch. If all numbers are in proper ascending order, the program terminates with a congratulations and some fanfare. It will also display the total number of moves made to solve the puzzle. To check for a solution in descending order, just change line 610 to FOR J=29 TO 8.

Note: If you wish to see display of finish without working through the whole puzzle, just type "GOTO 700."

```
1 REM ************************************
2 REM NUMBER SHUFFLE
3 REM
4 REM Frank Roberts
5 REM 3736 Ferndale Drive
6 REM Ft. Wayne, IN 4615
7 REM ************************************
8 REM
10 GOTO 30
20 BRD(P)=MOVE:POSITION X(P),Y(P):IF BRD(P)<10 THEN PRINT #6;" ";
25 PRINT #6;BRD(P):RETURN
30 DIM BRD(36),NBR(16),X(36),Y(36),R$(10)
40 X(8)=3:X(9)=6:X(10)=9:X(11)=12
41 X(14)=X(8):X(15)=X(9):X(16)=X(10):X(17)=X(11)
42 X(20)=X(8):X(21)=X(9):X(22)=X(10):X(23)=X(11)
43 X(26)=X(8):X(27)=X(9):X(28)=X(10):X(29)=X(11)
44 REM
45 Y(8)=5:Y(14)=8:Y(20)=11:Y(26)=14
46 Y(9)=Y(8):Y(15)=Y(14):Y(21)=Y(20):Y(27)=Y(26)
47 Y(10)=Y(8):Y(16)=Y(14):Y(22)=Y(20):Y(28)=Y(26)
48 Y(11)=Y(8):Y(17)=Y(14):Y(23)=Y(20):Y(29)=Y(26)
49 REM
50 GRAPHICS 2:POSITION 2,4:PRINT #6;"NUMBER SHUFFLE"
60 FOR MUSIC=500 TO 0 STEP -1.5:SOUND 0,0,8,MUSIC:POKE 708,INT(RND(0)*222)
   :NEXT  MUSIC:SOUND 0,0,0,0
70 PRINT "}DO YOU WANT INSTRUCTIONS ";:INPUT R$:IF R$(1,1)<>"Y" THEN 100
80 GRAPHICS 0:POSITION 2,7
81 PRINT "NUMBER SHUFFLE IS A SOLITAIRE GAME IN WHICH 15 NUMBERS ARE
   SCRAMBLED W ITHIN"
82 PRINT "A 4 X 4 SQUARE.  THE NUMBERS ARE THEN MOVED ONE AT A TIME INTO
   AN EMPT Y"
83 PRINT "SQUARE IN AN ATTEMPT TO ARRANGE THE   NUMBERS IN NUMERICAL ORDER.
    THE ONLY"
84 PRINT "VALID MOVE IS ONE OF THE FOUR NUMBERS"
85 PRINT "ADJACENT TO THE EMPTY SPACE IN THE    SQUARE."
86 PRINT :PRINT "WHEN YOU ARE READY, PRESS RETURN";:INPUT R$
99 REM ***** SET UP BOARD *****
100 GRAPHICS 1:POSITION 4,5:PRINT #6;"STANDBY...."
110 POSITION 3,7:PRINT #6;"I'M MIXING UP"
120 POSITION 5,9:PRINT #6;"THE BOARD"
199 REM ***** CHOOSE SET OF RANDOM NUMBERS *****
200 FOR J=0 TO 15:NBR(J+1)=J:NEXT J
209 REM ***** NOW SHUFFLE NUMBERS *****
210 FOR J=1 TO 100
220 A=INT(RND(0)*15)+1
230 B=INT(RND(0)*4)
231 IF B=0 THEN B=A-1:IF B<1 THEN B=A+1
232 IF B=1 THEN B=A-4:IF B<1 THEN B=A+4
233 IF B=2 THEN B=A+1:IF B>16 THEN B=A-1
234 IF B=3 THEN B=A+4:IF B>16 THEN B=A-4
240 TEMP=NBR(A):NBR(A)=NBR(B):NBR(B)=TEMP
250 NEXT J
299 REM ***** NOW SET UP BOARD *****
300 FOR J=1 TO 36:BRD(J)=-1:NEXT J
310 C=0
320 FOR J=8 TO 29
330 IF J=12 OR J=13 OR J=18 OR J=19 OR J=24 OR J=25 THEN 370
340 C=C+1
350 BRD(J)=NBR(C)
360 IF BRD(J)=0 THEN P=J
370 NEXT J
399 REM ***** DISPLAY BOARD *****
400 GRAPHICS 1:POKE 752,1:C=8:POSITION 6,4
410 FOR ROW=1 TO 4
420 PRINT #6:PRINT #6;"   ";
430 FOR J=C TO C+3
435 IF BRD(J)=0 THEN PRINT #6;"   ";:GOTO 460
440 IF BRD(J)<10 THEN PRINT #6;" ";
450 PRINT #6;BRD(J);" ";
460 NEXT J
470 C=C+6
475 PRINT #6:PRINT #6
480 NEXT ROW
```

(continued)

Number Shuffle *(continued)*

```
499 REM ***** GET MOVE NUMBER AND PROCESS *****
500 POP :TRY=TRY+1
505 PRINT "                              ":POKE 656,PEEK(656)-1
510 PRINT "ENTER NUMBER TO MOVE: ";
520 INPUT MOVE
530 IF BRD(P-1)=MOVE THEN BRD(P-1)=0:GOSUB 20:P=P-1:POSITION X(P),Y(P)
    :PRINT #6; "  ":GOTO 600
540 IF BRD(P-6)=MOVE THEN BRD(P-6)=0:GOSUB 20:P=P-6:POSITION X(P),Y(P)
    :PRINT #6; "  ":GOTO 600
550 IF BRD(P+1)=MOVE THEN BRD(P+1)=0:GOSUB 20:P=P+1:POSITION X(P),Y(P)
    :PRINT #6; "  ":GOTO 600
560 IF BRD(P+6)=MOVE THEN BRD(P+6)=0:GOSUB 20:P=P+6:POSITION X(P),Y(P)
    :PRINT #6: "  ":GOTO 600
570 PRINT ">ILLEGAL MOVE":GOTO 510
599 REM ***** CHECK BORAD FOR WIN *****
600 C=0:IF BRD(29)()0 THEN POKE 656,PEEK(656)-1:GOTO 500
610 FOR J=8 TO 28
620 IF J=12 OR J=13 OR J=18 OR J=19 OR J=24 OR J=25 THEN 650
630 C=C+1
631 PRINT C,BRD(J)
640 IF BRD(J)=C THEN 650
642 POKE 656,PEEK(656)-1
644 GOTO 500
650 NEXT J
699 REM ***** CONGRATULATE WITH FANFARE *****
700 GRAPHICS 1:POSITION 2,6:PRINT #6;"CONGRATULATIONS"
710 POSITION 3,8:PRINT #6;"YOU DID IT IN"
720 POSITION 5,10:PRINT #6;TRY;" MOVES"
729 REM ***** PLAY SONG *****
800 FOR J=1 TO 6
810 READ S,S2,I
820 SOUND 0,S,10,8:SOUND 1,S-1,10,4:SOUND 2,S2,10,4:SOUND 3,S2-1,10,2
830 FOR K=1 TO I:NEXT K
840 NEXT J
850 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0:SOUND 3,0,0,0
900 DATA 162,81,15,121,60,20,96,47,25,81,40,70,96,47,20,81,40,100
```

MICRO™

Letterbox

Two Many Lines

Kerry Lourash, author of "Surchange for OSI" in the August issue, noticed an error in his listing. On page 77, there are two line numbers for 1100. Type in only one.

Move the Decimal

An error in the September Software Catalog last month could have bankrupted MicroSoftware International, Inc. *Computer Business Software* was wrongly listed as $64.70. It should have been $6470.00. Pardon our slip.

OSI Delete Modifications

Claude Barron of Quebec, Canada, sent in this update.

In the August issue of MICRO, Morris and Morishita wrote a program for "Delete on the OSI." Here is a modification that will save you seven bytes in your program.

In their program to get the code for the "OK", you go through the output to CRT routine, and then get back to BASIC in address $A319. Another way would simply get you back in BASIC through the warm start located at $A274. This routine will print the "OK". Here are the necessary changes in your BASIC program:

```
18 N=57: FOR X=M TO M+N-1:
   READ J: POKE X, J: NEXT
22 A=INT (M/256): B=M-256*A:
   POKE M-6, A: POKE M-7, B
42 DATA 144, 5, 32, 110, 2, 240, 219,
   76, 116, 162
```

There is no need for a line 44.

I would also like to point out a little mistake in the text. When you write that the code is relocatable with the exception of the JSR at $026E, it should be $0266.

MICRO™

Sensible Use of Apple Game Paddles

by Harry L. Pruetz

Some programming suggestions to improve the validity of game-paddle inputs.

Paddle Use
requires:

Apple computer with either Integer or Applesoft BASIC and the Apple game paddles

When using Apple II game paddles, there are both hardware and software problems involved. With some knowledge of inherent limitations, the paddles can add to the enjoyment of computer games, both while designing a game and playing it.

My Apple II came with two paddles hooked up to the game I/O connector. The paddles were actually rotating potentiometers, rather than the joystick-type paddles available. Unlike two-player games, games designed for one player using both paddles for x-y control would be better played with a real joystick controlling both x and y coordinate movements.

The most obvious limitation on both game paddles is the 300-degree physical limit on control knob movement. Numerical values possible from a paddle range from 0 at the complete counter-clockwise position, up to 255 at the complete clockwise position. However, these values are available for only about a 150-degree rotation of the knob.

The values read from BASIC using the PDL function may be expressed as a function of the angle of the paddle knob as follows:

Angle	Value
0-29	—
30-65	0
66-101	0-63
102-137	64-127
138-173	128-191
174-209	192-255
210-329	255
330-359	—

The value read from a paddle may change by 1 when the knob is moved slightly more than half of a degree! Labeling the paddles and marking the actual ranges are quite helpful for many games.

The paddle cable that plugs into the game I/O connector is a source of problems. Each time the paddles are disconnected, cable pins may get bent when reconnecting the paddles. The pins may be bent back into shape and the electrical connection will still be sound. However, a simple program to sample and display paddle values should be run occasionally to make sure the paddles are functioning correctly.

A monitor routine measures a paddle value by delaying 12 microseconds for each unit value measured after the paddle is triggered. Thus a value of 255 takes three milliseconds to measure. Times of one to three milliseconds for each PDL call are not excessive for Integer or Applesoft BASIC. There are usually many other statements executed between PDL calls. Compiled BASIC or faster computer languages may still use the paddles for game input without noticeable effect from the slow timing speed. It is usually the machine-language implementations of sophisticated, fast-action games that avoid using paddles.

The program listed is a RAM Applesoft BASIC program to test and demonstrate some software techniques useful with game paddles.

Lines 100 through 195 merely sample paddle 0 and print a value if it is different from the previous sample. Line 110 exits the sampling loop when any key is pushed. The paddle may be set to any value greater than 0 and less than 255, and left untouched for a while. Because of the sensitivity of the pot and the monitor routine which determines the value PDL(0), the value often skips back and forth between two consecutive numbers. This causes errors in some games that are not the player's

Listing 1

```
10  REM ************************
15  REM *   PADDLE USE DEMOS
20  REM *
25  REM *  BY HARRY L. PRUETZ
30  REM ************************
35  DIM XC%(128),YC%(128)
40  DEF  FN X(D) = 4 *  INT (1 +  INT ( PDL (0) * 68) / 255)
45  DEF  FN Y(D) = 2 *  INT (2 +  INT ( PDL (1) * 32) / 255)
47  GOTO 95
50  REM *** PDL(0) CALC ***
55 D = ( PDL (0) - 4 * XP) / 4:XP = XP +  SGN (D) *  INT ( ABS (D)):X = 1
    1 + 4 * XP: RETURN
60  REM *** PDL(1) CALC ***
65 D = (255 -  PDL (1) - 3 * YP) / 3:YP = YP +  SGN (D) *  INT ( ABS (D))
    :Y = 10 + 2 * YP: RETURN
70  REM  *** FRAC PDL(0) CALC ***
75 F = 255 / 45:D = ( PDL (0) - F * XP) / F:XP = XP +  SGN (D) *  INT ( ABS
    (D)):X = 6 * XP: RETURN
80  REM  *** FRAC PDL(1) CALC ***
85 F = 255 / 45:D = (255 -  PDL (1) - F * YP) / F:YP = YP +  SGN (D) *  INT
    ( ABS (D)):Y = 4 * YP: RETURN
95  POKE  - 16368,0: HOME
100  REM
101  REM ************************
102  REM * PADDLE 0 SAMPLING
103  REM ************************
104  REM
105 VP =  PDL (0)
110  IF  PEEK ( - 16384) > 127 THEN 195
115 VC =  PDL (0)
120  IF VC <  > VP THEN  PRINT VC;" ";
125 VP = VC
130  GOTO 110
```

(continued)

Listing 1 *(continued)*

```
195  POKE  - 16368,0: HOME
200  REM
201  REM ***********************
202  REM * PADDLE 1 AVERAGING
203  REM ***********************
204  REM
205 VP =  PDL (1)
210  IF  PEEK ( - 16384) > 127 THEN 295
215 VC =  INT ( INT ( PDL (1) + VP) / 2)
220  IF VC <  > VP THEN  PRINT VC;" ";
225 VP = VC
230  GOTO 210
295  POKE  - 16368,0: HOME
300  REM
301  REM ***********************
302  REM * SMOOTH FADING TRAIL
303  REM ***********************
304  REM
305 PC = 1:NP = 128
310 X =  FN X(0)
315 Y =  FN Y(1)
320  FOR I = 1 TO 128:XC%(I) = X:YC%(I) = Y: NEXT I
325  HGR2
330  IF  PEEK ( - 16384) > 127 THEN 395
335  HCOLOR= 0: HPLOT X - 3,Y - 3 TO X + 3,Y + 3: HPLOT X - 3,Y + 3 TO X +
     3,Y - 3
340  HPLOT XC%(PC),YC%(PC)
345  HCOLOR= 3: HPLOT X,Y
350 XC%(PC) = X:YC%(PC) = Y
355 PC = PC + 1: IF PC > NP THEN PC = 1
360 X =  INT ( INT (X +  FN X(0)) / 2)
365 Y =  INT ( INT (Y +  FN Y(1)) / 2)
370  HPLOT X - 3,Y - 3 TO X + 3,Y + 3: HPLOT X - 3,Y + 3 TO X + 3,Y - 3
375  GOTO 330
395  POKE  - 16368,0: HOME
400  REM
401  REM ***********************
402  REM * DISCRETE FADING TRAIL
403  REM ***********************
404  REM
405 PC = 1:NP = 128:XP = 0:YP = 0
410  GOSUB 50
415  GOSUB 60
420  FOR I = 1 TO 128:XC%(I) = X:YC%(I) = Y: NEXT I
425  HGR2
430  IF  PEEK ( - 16384) > 127 THEN 495
435  HCOLOR= 0: HPLOT X - 3,Y - 3 TO X + 3,Y + 3: HPLOT X - 3,Y + 3 TO X +
     3,Y - 3
440  HPLOT XC%(PC),YC%(PC)
445  HCOLOR= 3: HPLOT X,Y
450 XC%(PC) = X:YC%(PC) = Y
455 PC = PC + 1: IF PC > NP THEN PC = 1
460  GOSUB 50
465  GOSUB 60
470  HPLOT X - 3,Y - 3 TO X + 3,Y + 3: HPLOT X - 3,Y + 3 TO X + 3,Y - 3
475  GOTO 430
495  POKE  - 16368,0: TEXT : HOME
500  REM
501  REM ***********************
502  REM * FRACTIONAL PADDLE INC
503  REM ***********************
504  REM
505 XP = 0:YP = 0: GOSUB 70: GOSUB 80:XC = X:YC = Y
510  HGR2 : HCOLOR= 2:SW = 0
515  FOR I = 0 TO 275 STEP 2
520  HPLOT I,0 TO I,183
525  NEXT I
530  HCOLOR= 1: FOR I = X + 1 TO X + 5 STEP 2: HPLOT I,Y TO I,Y + 3: NEXT

535  IF  PEEK ( - 16384) > 127 THEN 595
540  GOSUB 70: GOSUB 80: IF X = XC AND Y = YC THEN 535
545  HCOLOR= 0: FOR I = XC + 1 TO XC + 5 STEP 2: HPLOT I,YC TO I,YC + 3: NEXT

550  HCOLOR= 0: HPLOT XC + 2,YC + 1 TO X + 2,Y + 1
555  HCOLOR= 1: FOR I = X + 1 TO X + 5 STEP 2: HPLOT I,Y TO I,Y + 3: NEXT

560 XC = X:YC = Y: GOTO 535
595  POKE  - 16368,0: TEXT : HOME
999  END
```

fault. When the paddle cable is not properly plugged into the game I/O connector, values may vary even more, sometimes changing when the other paddle is moved.

Lines 200 through 295 provide a more stable readout from paddle 1. A different paddle is used here to allow the program to be used for paddle checkout. When paddle 1 is not being moved, the averaged value VC changes only in very rare cases. A disadvantage of this averaging method is that the range is now only 0 through 254. Averaging also slows down the rapidity with which VC may change. Paddle 0 in the first case can be moved from one extreme to the other during only three or four samplings. Paddle 1 in this case takes about five more samplings for VC to catch up to the actual PDL(1) value. Line 215 could also be changed to give the previous value more weight with the penalty of slowing down the sampling even more. For example:

```
215 VC = INT(INT(PDL(1) + 2*VP)/3)
```

The INT functions required in Applesoft BASIC make the above statement more inefficient than when using Integer BASIC.

Lines 300 through 395 use high-resolution graphics to demonstrate paddle sampling with averaging. XC% and YC% are arrays used as circular queues to allow display of the last 128 locations of the 'X' shape. The functions at lines 40 and 45 convert a paddle value into x and y coordinates. Again, INT functions are required in Applesoft BASIC to insure integer arithmetic. The numbers in the defined functions are fairly easy to obtain by using the "slope-intercept form" for straight lines. From Analytic Geometry:

$f = a + mg$,
where g is the independent variable,
m is the slope of the line,
a is the value of f at $g = 0$,
and f is the dependent variable.

We want the paddles to determine screen location. Only multiples of 4 are used as x-coordinate locations, so the figure 'X' may be plotted at 4, 8, ..., and 276 without $x - 3$ or $x + 3$ causing "illegal quantity" errors. Ignoring the multiplier of 4 for now, f should be 1 when PDL(0) = 0, and 69 when PDL(0) = 255. For $f = a + m*PDL(0)$, $a = 1$ and $m = (69 - 1)/(255 - 0)$, so that $f = 1 + (PDL(0)*68)/255$ and $x = 4*f$. Thus we get line 40, where D is an unused dummy variable:

```
40 DEF FN X(D) = 4*INT(1 + INT
  (PDL(0)*68)/255)
```

Multiples of 2 are used as y-coordinate locations, so the figure 'X' may be plotted at 4, 6, ..., and 188 without y−3 or y+3 causing "illegal quantity" errors. Ignoring the multiplier of 2 for now, f should be 2 when PDL(1)=0, and 94 when PDL(1)=255. For f = a+m*PDL(1), a=2, m=(94−2)/(255−0), and y = 2*f. Thus we get line 45, where D is an unused dummy variable:

```
45 DEF FN Y(D) = 2*INT(2+INT
   (PDL(1)*92)/255)
```

Line 320 initializes the circular queue. The actual loop is lines 330 through 375. The loop performs the following steps:

1. Erase current 'X'.
2. Erase last point in trail.
3. Plot current point in trail.
4. Save point location in queue.
5. Adjust queue pointer.
6. Plot next 'X'.

Lines 360 and 365 average the x and y values obtained from the paddle functions. The averaging has a smoothing effect on movements of the figure 'X' as it is moved around on the high-resolution grid.

Lines 400 through 495 demonstrate a way of obtaining reliable paddle input without averaging. The routine at line 50 is essentially the same as the one at line 60. However, the PDL(0) routine will be explained in detail. Let us consider the case of PDL(0)=0 and XP=0. If PDL(0) is increased to 4, then XP becomes 1. If PDL(0) changes back to 3 by itself, there will be no new XP because the difference is only 1. Now, when PDL(0) is increased to 7, there will still be no change in XP. If PDL(0) changes to 8 by itself and then back to 7 again, the value XP=2 will be calculated and will remain in effect until PDL(0) is changed to 4 or 12. Thus the output of this routine is stable without averaging. I have chosen to call the method differencing. Note that merely dividing PDL(0) by 4 would not give stable results. The use of SGN(D)*INT (ABS(D)) is used since INT(−1.1) gives −2 instead of the −1 desired.

Note also that 255-PDL(1) is used in line 65 rather than just PDL(1). This gives the same "intuitive" coordinate change for the y-coordinate as paddle 0 does for x-coordinate values. Thus, clockwise rotation of paddle 1 causes movement from the bottom to the top of the CRT display.

The rest of this case is essentially the same as lines 300 through 395.

Finally, we have a demonstration in lines 500 through 595 of fractional ranges of the paddles. The routines that sample the paddles are at lines 70 and 80. The fraction involved is 255/45 or 5.66667. This gives a high-resolution display of 46 by 46 cells six points wide and four points deep. Only 276 of 280 horizontal, and 184 of 192 vertical high-resolution points are used. The white cursor is plotted and erased using the complimentary color between the background lines. Straight black lines are drawn between each position of the cursor as it is moved using the game paddles.

Fractions much smaller than 5.66667 may be used in the difference calculations for game paddles. I have used a fraction as small as 1.275 to get 200 stable values from a game paddle. The values are not exactly "one-to-one" for a game paddle. For example, PDL(0)=7 may give a calculated value of 5 or 6 depending on the previous value. However, the calculated values are stable and the game player cannot see the difference.

If you have Integer or Applesoft BASIC programs that use game paddles and do not give satisfactory results, you may consider making a few simple changes similar to the given examples. Programs that use only a few keys may also be changed to use the game paddles. Above all, sit back and relax. An aching back caused by leaning over a keyboard is not really necessary.

Contact the author at 2929 Clydedale, #376, Dallas, TX 75220.

MICRO

Space Invasion for OSI C1P/Superboard

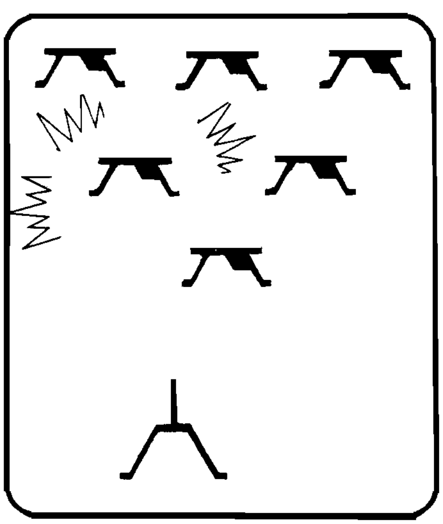
by John S. Seybold

Space Invasion
requires:
OSI C1P or Superboard

In this real-time arcade game you must fight waves of incoming aliens. Each time you clear the screen of aliens it refills with more aliens, faster and closer than the last bunch. The more times you clear the screen, the more points each alien is worth.

The aliens are moved by the user routine; their addresses are updated in line 2510. Z tells the routine which way to move the aliens. To adjust the pace of the game, you may adjust line 2082 or line 2520. If you remove all of the REM statements, the program should run in 4K of memory.

Mr. Seybold is employed by General Dynamics in Pomona, CA. He has owned an OSI Superboard for a year and a half. Address correspondence to 3210 Quartz Lane, #A-10, Fullerton, CA 92631.



Space Invasion

```
2 DIMA(49):W=53366:WW=W-17:AZ=750:X=54214:Y=54235
5 REM-ADJUST LINES 2520 AND 2082 FOR SPEED,ADD SOUND TO 650,7900
10 GOSUB200:POKE11,34:POKE12,2:SC=0
20 HS$="HIGH SCORE":PRINT"      SPACE INVASION"
40 PRINT"      ----- --------":GOSUB200:V=32:KB=57088:M=251
50 PRINT"C > RIGHT":PRINT"X > LEFT":PRINT"M > FIRE":PRINT
80 PRINT:INPUT"DIFFICULTY LEVEL (1-5)";D:IF(D<1)OR(D>5)THEN80
100 PRINT"HIT 'M' TO START":POKE530,1:POKEKB,M
110 IFPEEK(KB)=MTHEN910
120 GOTO110
200 FORK=1TO12:PRINT:NEXT:RETURN
400 REM-TURRET FIRE ROUTINE**********
410 TM=P-V:IFPEEK(TM)<>AATHENPOKETM,39:RETURN
420 POKETM,V:RM=RM-1:TM=0:SC=SC+10:RETURN
600 REM-TURRET FIRE UPDATE ROUTINE***
610 J=PEEK(TM):IFJ=AATHENTM=0:RETURN
620 POKETM,V:TM=TM-V:IFTM<A(B)THENTM=0:RETURN
625 IFTM=AFTHENTM=0:RETURN
630 J=PEEK(TM):IFJ=VTHENPOKETM,39:RETURN
640 POKETM,V:TM=0:IFJ<>AATHENRETURN
650 RM=RM-1:SC=SC+10:SC$=STR$(SC):FORK=1TOLEN(SC$)
660 POKEW+K,ASC(MID$(SC$,K,1)):NEXTK:IFSC<HSTHENRETURN
670 HS=SC:REM-HIGH SCORE PRINT ROUTINE*****
680 FORK=1TOLEN(HS$+STR$(HS))
690 POKEWW+K,ASC(MID$(HS$+STR$(HS),K,1)):NEXTK:RETURN
700 REM-ALIEN FIRE UPDATE ROUTINE****
710 POKEAF,V:AF=AF+V:J=PEEK(AF):IF(J=AA)OR(AF>P)THENAF=0:RETURN
720 IFJ=CVTHENPOKEAF,V:AF=O:RETURN
730 POKEAF,39:RETURN
900 REM-SET UP SCREEN**************************
910 FORK=1TO25:PRINT:NEXT:POKE54149,V:Z=54087:CV=161
920 FORK=1TO4:POKEZ,CV:POKEZ+1,CV:POKEZ+V,CV
930 POKEZ+33,CV:POKEZ-V,CV:POKEZ-31,CV:Z=Z+5:NEXTK:P=54224:BB=49
1000 B=1:AF=0:TM=0:RM=BB:C=10:Z=1:AA=231:TURRET=236
1010 CL=7:FORK=1TOCL:READA(K):A(K)=A(K)+V*D:POKEA(K),AA
1020 FORI=KTOBBSTEPCL:IFI=KTHENNEXTI
1030 A(I)=A(I-CL)+2*V:POKEA(I),AA:NEXTI:NEXTK:POKEP,TU:GOSUB680
1040 FORI=546TO623:READJ:POKEI,J:NEXTI
2000 REM-MOVE ALIENS? PROGRAM LOOP**********************************
2010 C=C-1:IFPEEK(A(B))=VTHENB=B+1
2080 IFPEEK(A(BB))=VTHENBB=BB-1
2082 IFC=>0THENFORK=1TO(60-5*D):NEXTK:GOTO4020
2083 IFRTHEN2500
2085 IFZ<0THEN2100
2090 FORK=YTOA(B)STEP-V:IFPEEK(K)=AATHENZ=V-Z:R=1:GOTO4020
2095 NEXTK:GOTO2500
2100 FORK=XTOA(B)STEP-V:IFPEEK(K)=AATHENZ=V-Z:R=1:GOTO4020
2110 NEXTK
2500 IFZ>0THENPOKEAZ,Z:GOTO2510
2505 POKEAZ,256+Z
2510 X=USR(X):FORK=BTOBB:A(K)=A(K)+Z:NEXTK:IFA(BB)=>XTHEN7900
```

Space Invasion *(continued)*

```
2520 R=0:C=.5+(RM/3):IFABS(Z)<>1THENZ=Z-V
4000 REM-CHECK FOR TURRET FIRE******
4020 J=PEEK(KB):IF((JORM)=M)ANDTM=0THENGOSUB410
4490 REM-CHECK FOR TURRET MOVEMENT**
4500 IF(JOR191)<>191THEN4600
4520 IFP<>YTHEN:POKEP,V:P=P+1:POKEP,TU:GOTO5000
4600 IF(JOR127)<>127THEN5000
4610 IFP<>XTHEN:POKEP,V:P=P-1:POKEP,TU
4990 REM-ALIEN FIRE*****************
5000 IFAF<>0THENGOSUB710:GOTO5100
5010 AL=BB-INT(5*RND(3)):IFAL<BTHENAL=BB
5040 IFPEEK(A(AL))=AATHENAF=A(AL)+V:POKEAF,39
5100 IFAF=PTHEN7900
5200 IFTMTHENGOSUB610:IFRM<=0THEN9000
5210 GOTO2010
7890 REM-TURRET DESTROYED*********************************************
7900 FORI=1TO200:POKEP,2:POKEP,3
7920 POKEP,232:POKEP,233:POKEP,32:NEXT:GOSUB200
7950 PRINT"THE ALIENS HAVE":PRINT:PRINT"OVERCOME YOUR DEFENSES"
7960 PRINT:PRINT"AND LANDED!!":GOSUB200
7965 PRINT"YOUR SCORE WAS";SC:PRINT
7980 FORK=1TO8:POKEW+K,V:NEXTK:FORI=1TO3000:NEXTI
8000 FORK=XTOY:POKEK,V:NEXT:POKE530,0
8020 GOSUB200:PRINT"     SPACE INVADERS":GOSUB200
8025 INPUT"PLAY AGAIN";K$:IFLEFT$(K$,1)="Y"THENRESTORE:GOTO10
8030 FORK=WWTOY:POKEK,V:NEXTK:END
8990 REM-NEXT SCREEN*****************
9000 RESTORE:IFD<7THEND=D+1
9010 FORK=XTOY:POKEK,V:NEXT:IFTM<>0THENPOKETM,V
9045 IFAF<>0THENPOKEAF,V
9050 POKEP,V:GOTO910
10000 DATA 53354,53356,53358,53360,53362,53364,53366
10010 DATA 216,169,211,133,217,169,0,133,216,160,255,177,216,201,231
10020 DATA 208,3,32,66,2,136,208,244,198,217,169,207,197,217,208,234
10030 DATA 96,169,32,145,216,152,72,174,238,2,48,27,24,109,238,2,168
10040 DATA 176,8,169,231,145,216,104,168,136,96,230,217,169,231,145
10050 DATA 216,198,217,104,168,136,96,24,109,238,2,168,24,144,228
```

MICRO™

Apple Shootdown

by Eric Grammer

A type and RUN low-resolution graphics game.

Shoot Down requires:

Apple II with Applesoft and game paddles

Here is a simple game with a short machine-language sound program. The oject of the game is to shoot at all the columns with dots in them. The dots will fire back at you. To move your ship, use paddle (0). To fire, press the button.

The machine-language section should be BSAVEd with A$300, L$30. The lo-res graphics listing is straightforward and is fairly self-explanatory. Good shooting!

Assembler Listing Laser Fire

```
                ;***********************
                ;*
                ;*        LASER FIRE
                ;*
                ;*       ERIC GRAMMER
                ;*
                ;***********************
                          ORG $300
                ;***********************
                REPEAT    EPZ $06
                USRRPT    EPZ $07
                YLOOP     EQU $330
                XLOOP     EQU $331
                SPKR      EQU $C030
                ;***********************
                START:
0300 A9 00                LDA #$00
0302 85 06                STA REPEAT
0304 A5 06      LOOP      LDA REPEAT
0306 C5 07                CMP USRRPT
0308 F0 08                BEQ END
030A E6 06                INC REPEAT
030C 20 13 03             JSR FIRE
030F 4C 04 03             JMP LOOP
0312 60         END       RTS
0313 A2 00      FIRE      LDX #$00
0315 A0 00      YCLEAR    LDY #$00
0317 AD 30 C0             LDA SPKR
031A C8         DELAY     INY
031B 98                   TYA
031C 8E 30 03             STX YLOOP
031F CD 30 03             CMP YLOOP
0322 30 F6                BMI DELAY
0324 E8                   INX
0325 8E 31 03             STX XLOOP
0328 A9 FF                LDA #$FF
032A CD 31 03             CMP XLOOP
032D D0 E6                BNE YCLEAR
032F 60                   RTS
0330                      END
```

The author may be contacted at 95 Old Street Road, Peterborough, NH 03458.

***Listing 1:* Shoot Down Applesoft Listing**

```
10  PRINT : PRINT  CHR$ (4)"BLOAD LASER FIRE.OBJ"
20  TEXT : HOME : PRINT "SHOOT DOWN"
25  DIM V(40)
30  PRINT : PRINT "WHAT'S YOUR SKILL LEVEL?"
40  PRINT : PRINT "1. SIMPLETON"
50  PRINT "2. FAIR"
60  PRINT "3. GOOD"
70  PRINT "4. EXCELLANT"
80  PRINT "5. MR. PERFECT"
90  PRINT : PRINT "PLEASE PRESS THE NUMBER OF YOUR CHOICE:";
100  GET SK$: IF  VAL (SK$) < 1 OR  VAL (SK$) > 5 THEN 100
110  SKILL =  VAL (SK$)
120  VTAB SKILL + 4: FLASH : HTAB 1: PRINT SKILL;: NORMAL
130  VTAB 20: PRINT "PRESS ANY KEY TO BEGIN... ";: GET A$
140  POKE 7,1
160  TEXT : HOME
170  GR : COLOR= 1
180 X =  INT ( PDL (0) / 7.3): GOSUB 380
190  FOR X = 1 TO 21 STEP 4: FOR Y = 3 TO 35 STEP 4: PLOT Y,X: NEXT Y,X
200  COLOR= 5
210  FOR X = 3 TO 19 STEP 4: FOR Y = 5 TO 33 STEP 4: PLOT Y,X: NEXT Y,X
220 SHIPS = 6 - SKILL
225  HOME : VTAB 24: PRINT SHIPS" SHIP";: IF SHIPS = 1 THEN  PRINT " LEFT
     1": GOTO 230
227  PRINT "S LEFT"
230  IF  INT ( RND (1) * (7 - SKILL)) = 1 THEN  GOSUB 440
240 X =  INT ( PDL (0) / 7.3): IF B <  > X THEN  GOSUB 380
250  IF  PEEK ( - 16287) > 127 THEN  CALL 768: GOSUB 280: GOTO 230
260  IF  INT ( RND (1) * (7 - SKILL)) = 1 THEN  GOSUB 440
270  GOTO 230
280  COLOR= 15:C = X
290  FOR I = 34 TO 0 STEP  - 2
300 X =  INT ( PDL (0) / 7.3): IF X <  > B THEN  GOSUB 380
310  FOR L = 1 TO 20: NEXT
320  COLOR= 15: VLIN I,I + 2 AT C + 2
330  COLOR= 0: PLOT C + 2,I
340  PLOT C + 2,I + 1: PLOT C + 2,I + 2
350  NEXT I
360 V(C + 2) = 1
366  COLOR= 14: VLIN 35,37 AT B + 2
370  RETURN
380  COLOR= 0: HLIN B,B + 4 AT 39: HLIN B,B + 4 AT 38: HLIN B + 1,B + 3 AT
     37
390  VLIN 35,37 AT B + 2
400 B = X
410  COLOR= 14: HLIN B,B + 4 AT 39: HLIN B,B + 4 AT 38: HLIN B + 1,B + 3 AT
     37
420  VLIN 35,37 AT B + 2
430  RETURN
440  IF  INT ( RND (1) * 2) = 1 THEN 460
450 Y1 =  INT ( RND (1) * 6) * 4 + 1:X1 =  INT ( RND (1) * 9) * 4 + 3: GOTO
     470
460 Y1 =  INT ( RND (1) * 5) * 4 + 3:X1 =  INT ( RND (1) * 8 + 1) * 4 + 1

470 Y1 = Y1 + 1: IF V(X1) = 0 THEN 490
480  GOTO 860
490  COLOR= 15
500  CALL 768: CALL 768
510  FOR L = Y1 TO 38 - SKILL STEP SKILL: VLIN L,L + SKILL AT X1
520 X =  INT ( PDL (0) / 7.3): IF B <  > X THEN  GOSUB 380
530  NEXT
540  COLOR= 0
550  FOR L = Y1 TO 34 STEP 3: VLIN L,L + 4 AT X1: NEXT
560  IF  SCRN( X1,39) <  > 0 THEN 590
570  COLOR= 0: VLIN Y1,39 AT X1
580 CC = 0: RETURN
```

(contin

Listing 1 *(continued)*

```
590  COLOR= 15
595  HOME
600  FOR A = 35 TO 39: HLIN X,X + 4 AT A: NEXT
610  COLOR= 0: HLIN X,X + 4 AT 35: CALL 768
620  HLIN X,X + 4 AT 36: CALL 768
630  HLIN X,X + 4 AT 37: CALL 768
640  HLIN X,X + 4 AT 38: CALL 768
650  HLIN X,X + 4 AT 39: CALL 768
655  IF SHIPS > 1 THEN  PRINT "PRESS ANY KEY FOR NEXT SHIP ";: GET A$:SHI
     PS = SHIPS - 1:X =  INT ( PDL (0) / 7.3): GOSUB 380: GOTO 225
660  IF X < 5 THEN X = 5
670  IF X > 32 THEN X = 32
680  FOR V = 20 TO 23: VTAB V: HTAB X: PRINT "GAME";: CALL 768: HTAB X: PRINT
     "    ";: NEXT V
690  FOR L = 1 TO 4: VTAB 24: HTAB X - L: PRINT "GAME ";: CALL 768: NEXT

700  FOR V = 20 TO 23: VTAB V: HTAB X: PRINT "OVER";: CALL 768: HTAB X: PRINT
     "    ";: NEXT V
710  FOR L = 0 TO 3: VTAB 24: HTAB X + L: PRINT " OVER";: CALL 768: NEXT

720  PRINT : PRINT
730  PRINT "CARE FOR ANOTHER GAME? (Y/N): ";
740  GET G$: IF G$ = "N" THEN 770
750  IF G$ = "Y" THEN  CLEAR : GOTO 20
760  GOTO 740
770  HOME : TEXT
780 I = 0
790 I = I + 1: VTAB I
800  FOR A = 39 TO 1 STEP  - 1: HTAB A: PRINT ">";: HTAB A: PRINT " ";: NEXT
     A
810 I = I + 1: VTAB I
820  FOR A = 1 TO 39: HTAB A: PRINT "<";: HTAB A: PRINT " ";: NEXT A
830  IF I < 20 THEN 790
840  VTAB 1: FOR I = 1 TO 20: VTAB I: HTAB 40: PRINT "^";: VTAB I: HTAB 4
     0: FOR A = 1 TO 5: NEXT A: PRINT " ";: NEXT I
850  HOME : END
860  FOR CO = 3 TO 35 STEP 2: IF V(CO) = 0 THEN CC = 1: GOTO 440
870 X =  INT ( PDL (0) / 7.3): IF B <  > X THEN  GOSUB 380
880  NEXT CO
890  CALL 768: CALL 768: CALL 768: PRINT "YOU WIN!"
900  GOTO 720
```

MICRO

MICRO™

PET Vet

By Loren Wright

What to Do with 96K — Two Word Processor Approaches

For the designers of Wordcraft Ultra and WordPro 5 Plus, it must have been a pleasant dilemma. Steve Punter and Pro-Micro Software had their popular WordPro 4 Plus to upgrade, while P.L. Dowson and DataView had the sleek Wordcraft 80 to work with. It is only logical that the extra memory provide more room for text, but it is what else has been added to the programs that makes them so interesting.

This column is devoted to a discussion of these two word-processing programs for the CBM 8096. Because these versions retain all the features of their predecessors, my discussion should be of use to people with 8032s, as well as to those with 8096s. A future PET Vet column will cover word processors for the 40-column PETs.

Wordcraft Ultra will run on an 8096, an 8032, or an 8032 expanded with Madison Computer's ZRAM board. WordPro 5 Plus is designed specifically for the 8096; it will not work with the 8032. Wordcraft versions require a special "data key" (formerly known as a "dongle"), while WordPro versions require a functional ROM in the $A000 socket.

I will start with a general discussion of the two word processor families. Except where otherwise noted, Wordcraft refers to all the Ultra configurations and Wordcraft 80, while WordPro refers to WordPro 4 Plus and WordPro 5 Plus. I will also discuss the extra enhancements in the new versions.

The "Flavor" of WordPro

WordPro essentially presents you with a continuous scroll of 80-character lines on which to type. You start in the upper left-hand corner and keep typing until you're done. Words get split, but (not to worry) they will be kept together at output time. This, coupled with there being only two status lines, allows a lot of text to be viewed at once. Margins, centering, justification, paging, line spacing, and several other features are handled at output time. These are specified in the text on special, non-printing format lines.

If you're working with elaborate tables or a particularly fancy text, the continuous nature of the text and the distracting format lines make it difficult to visualize what the result will look like. There is a special output-to-video command that will show you the results on the screen, without wasting paper, but editing must still be done on the original, continuous version, and this is time consuming.

WordPro has a special feature called "Extra Text," which may be used to hold other text files, disk directories, commonly used phrases and paragraphs, or files of names and addresses for filling form letters. In WordPro 5 Plus there are four such extra text areas. WordPros 3 and 4 have only one, but the relative sizes of the extra and main text areas can be apportioned differently for different needs. I used WordPro 5 Plus to write this column, primarily because of the "append characters" function. I stashed the names WordPro, WordPro 5 Plus, Wordcraft, and Wordcraft Ultra in extra memory, so that with only a few keystrokes I could copy those characters at the current cursor position. Whole paragraphs are handled in a similar manner. The extra text also makes it possible to have a whole file of names and addresses in memory at the same time as the target form letter.

The "Flavor" of Wordcraft

If you've used a big, dedicated word processor, you will be more comfortable with Wordcraft. With a few exceptions, what you see on the screen is what you get on paper. You can tell how lines will look before they are printed. Even documents wider than 80 columns can be handled. The screen is a window that can be panned across the text. If a really long word has been moved to the next line, leaving the previous one too short, you can easily tell where to insert a soft hyphen to even up the line lengths. Most format commands are indicated by the presence of reverse field on a text character, but they don't take up space on the screen. Instead of giving you a continuous scroll of lines, Wordcraft gives you a certain number of characters to work with. It forces new pages as needed according to your page-length specifications.

One powerful feature of Wordcraft is the ruler, which indicates the positions of your margins and tab stops. Its current contents are shown on the bottom status line (of five). The ruler may be changed at any time. If you're trying to get a table to look just right, you can try out one arrangement of tab stops, type in your text with tab characters at the right points, and then go back and readjust your ruler without reentering the text or the tab characters. The text will automatically line up at the new tab stops as soon as you finish changing the ruler.

A Comparison of WordPro and Wordcraft

Now that I have discussed the special features of Wordcraft and WordPro, I can compare them on what they have in common. Most word processors perform the following functions in some way. Some do them better than others.

Entry of Text: Both programs do a good job here. You can continue typing without worrying about the way words carry over to the next line. WordPro uses a simple carriage return to force a new line and the tab key to advance to the next tab stop. With Wordcraft, these must be preceded by the control key. The Commodore business keyboard, with its full cursor control, is well-suited to word processing.

Editing of Text: Both WordPro and Wordcraft have commands for deleting, inserting, transferring, and duplicating text. In general, Wordcraft's commands are more powerful, making cut-and-paste operations very easy. WordPro's delete, transfer, and duplicate commands have restrictions, which can be circumvented by inserting and deleting spaces. Insert mode with WordPro is more convenient.

Search and Search-and-Replace: Neither program excels at these functions. WordPro's command structure is complicated. The search-and-replace command is an all-or-nothing proposition. The search command finds the next occurrence of the search string, but requires a different command sequence to continue searching for the same string. Both commands can act globally or just on the current file, and there is an option to ignore case. Wordcraft's commands make more sense, but there is no global option and delimiters (an artifact from line editors) are required around the search string. Both programs use the '?' as a "wild" character.

File Maintenance: WordPro makes this easy. It is possible to recall files using a few characters and an asterisk (the same as with other PET files). Also, file names can be read directly from the text area of the screen, such as from a comment line or directory listing. It is also possible to get a selective directory listing. When you have specified a file name that already exists, WordPro asks you if you want to replace it. Wordcraft, with its chapter organization and optional descriptive names and dates, makes documentation of the contents of a disk much more complete. However, saving and loading files is more cumbersome. Full file names must be typed in.

Support of Printers: WordPro fully supports a limited number of printers. It was designed specifically with the NEC Spinwriters in mind, and the combination works very well. The Diablo 630, Qume Sprint 5, CBM 8027, TEC 1500, and dot matrix types, are supported as well. However, other printers may not be fully supported. Wordcraft comes with a long list of printer data files, each matched to a particular printer/print wheel combination, on the master disk. The Wordcraft dealer can obtain a PDF from the distributor for just about any printer that runs off the IEEE. Wordcraft Ultra has special features for supporting printers with proportional spacing print wheels.

Generation of Form Letters: Wordcraft's handling of these is more powerful. The fields in the letter have unique identifiers, so that the same piece of information from the fill file may be reused. This avoids a lot of extra typing when you construct the fill file. However, the fill file must be used from the disk, and while entries may be used selectively, it is difficult to remember which page numbers go with which entries in the file. This makes editing difficult. WordPro, with its extra memory feature, allows the fill file and the form letter to be in memory at the same time, making editing much easier. Blanks in the form letter are filled sequentially, so if the same item has to be reused in the letter it must appear twice in the fill file. Selective use of entries from the fill file is also more difficult.

Handling of Long Files: WordPro has a chaining feature that allows a document to extend beyond the capacity of the computer. Print, search, search-and-replace, copy, and output-to-video commands can all be specified with global options. A global command will operate not only on the file in memory, but also on all the files linked to it and stored on disk.

Wordcraft, instead of chaining files, uses a "chapter" organization. To continue a file, save the first as chapter 1, then assign the continuation to chapter 2 of the same file. Files may be printed out globally, or by specifying a certain range of chapter numbers. Also, individual pages may be selected. However, commands such as search, search-and-replace, and copy act only on the chapter in memory.

What's Been Added?

Wordcraft's text size has been approximately doubled, from about 11,000 characters in Wordcraft 80 to over 20,000 in Wordcraft Ultra. Wordcraft 80 requires the master disk to be present in one drive, since the editor and print module could not reside in memory at the same time. With Wordcraft Ultra the entire program resides in memory at one time, leaving both drives available for file operations. (The 8032 configuration requires a ROM, which contains program modules.) Wordcraft Ultra adds a set of escape codes that provide such capabilities as selective double-spacing and centering or right alignment of a series of lines. Other escape codes handle

(continued)

pitch, vertical spacing, horizontal spacing, and proportional spacing. Unfortunately, these non-printing characters occupy space on the screen, making it a less accurate representation of what will be printed. Also added is support of Canadian Micro Distributing's MUPET multiple-user system. Wordcraft Ultra also has its own built-in multiple user system. Another big improvement is the provision to handle proportional spacing print wheels. The price of Wordcraft 80 is $395, while that of Wordcraft Ultra is $545.

WordPro 5 Plus includes the main and four extra text areas, each containing 169 80-character lines. Therefore, the maximum length of a single file in memory has not increased from WordPro 4 Plus to WordPro 5 Plus, but then neither has the price (both $450). WordPro 5 Plus has added support of the MUPET system, as well. I have just barely begun to explore the possible uses of the extra text areas.

Conclusions

With the possible exception of the IBM PC, the CBM 8032/8096 has the best keyboard for microcomputer business applications. Easy cursor control, built-in screen editing, and the separate numeric keypad all contribute. Dedicated word processors have many special keys on the keyboard for all the things a word processor has to do. A microcomputer keyboard has to be used for applications besides word processing, so it can't afford the luxury of all the extra keys. Both programs take advantage of what the computer has to offer.

Overall, I rate Wordcraft as a more powerful word processor. However, WordPro is much easier to learn. If you don't use Wordcraft every day, you will find yourself frequently looking things up, even if you learned it thoroughly the first time around. Beyond that, it really depends on what you want to do. If you plan to do a lot of "boiler-plating" (putting a document together using standard phrases and paragraphs) then you will find WordPro easier to use. If you're working on a really long document, such as a book, then the chapter organization of Wordcraft will be handy. For full control of a variety of letter-quality printers, for tabular material, and for material wider than 80 columns, Wordcraft has the edge. A big point in WordPro's favor is that it is (in its many versions) already the most popular word processor on the market for Commodore systems. This means that it is easier to exchange files with other people and that commercial programs, such as spelling checkers and data-base managers, will be written for WordPro first.

When you consider word processors for the 8096 you shouldn't forget Silicon Office — a combination word processor, data-base manager, and communications package. Contributing Editor Jim Strasma covered it in a full-length review in MICRO's June issue. He was particularly impressed with the word processor portion. It can handle long files much better (the whole disk!), can handle multiple-column documents, and can perform calculations using information from the data base. The $995 price is out of the range of the word processors covered here, but if you need a data-base manager, too, be sure to give Silicon Office a lot of consideration.

WordPro is distributed to dealers by Professional Software, Inc., 51 Fremont Street, Needham, MA 02194. Wordcraft 80 and Wordcraft Ultra are distributed to dealers by Computer Marketing Services, Inc., 300 W. Marlton Pike, Suite 26, Cherry Hill, NJ 08002.

Special thanks to Jim Lucivero of NEECO in Needham, MA, for the use of letter-quality printers used in this review.

MICRO

Apple Hi-Res Graphics and Memory Use

by Dan Weston

This article examines the conflicts of programs and hi-res graphics on the Apple II, plus several techniques to avoid these conflicts.

Hi-Res
requires:
Apple II with 32K

When you begin to write long programs that use the hi-res pages of the Apple II, you will overwrite the graphic display area if your program and variables are longer than 6K. There are many ways to get around this problem, which involve manipulating the pointers Applesoft uses to control memory usage. Here I discuss several methods for making the most of your computer's memory.

Normally a BASIC program is loaded beginning at memory location $800 (2048). The program fills memory upwards from $800. LOMEM is set to the end of the program and will change as the program changes.

Simple variables are stored from LOMEM upward as they are defined by the program. Arrays are stored from the end of simple variables upward. An addition to the simple variable space will push the array variables upward with no loss of integrity. Finally, string variables are stored from the top of available memory, HIMEM, downward in memory, with new strings being placed in successively lower memory locations. The pointers that guide the placement of variables are summarized in figure 1, and will be discussed later in this article.

The problem with this storage scheme is that the hi-res pages are located between the program and the end of memory. Hi-res page 1 display area sits between $2000 and $4000 (16384-24575), and hi-res page 2 sits from $4000 to $6000 (24576-32758). There is a 6K block of memory between $800 and the beginning of hi-res page 1. If your program is longer than 6K and invokes HGR, you will find that the last portion of your program has been wiped from memory. Even with programs shorter than 6K, an HGR call can wipe out arrays and variables that are stored above the program.

To avoid this problem, set LOMEM at the upper end of the hi-res page (1 or 2), so that simple variables and arrays will be stored above the hi-res area rather than across it. This method requires that the program be less than 6K, and that the LOMEM statement come before any variables are defined. LOMEM: 16384 will store variables above hi-res page 1. LOMEM: 24576 will store variables above hi-res page 2.

Another method is to put graphics on hi-res page 2 instead of page 1. This frees the 8K bytes of the first hi-res page for program and variable storage, giving

***Figure 1:* Applesoft Memory and Variable Pointers**

Pointer Name	Hex	Dec	Normal Setting	Special Effects	
Beginning of Applesoft program	$67 $68	103 104	program loads at $800	POKE 103,1 POKE 104,64 POKE 16384,0	: program loads : above hi-res : page 1
				POKE 103,1 POKE 104,96 POKE 24576,0	: program loads : above hi-res : page 2
LOMEM beginning of simple variables	$69 $6A	105 106	end of current program	POKE 105,1 POKE 106,64	: put variables : above hi-res : page 1
				POKE 105,0 POKE 106,8	: LOMEM at $800 : below hi-res
End of simple variables	$6B $6C	107 108	adjusts with size of variable table	POKE 107,0 POKE 108,8	: simple : variables : below hi-res
End of array variables	$6D $6E	109 110	adjusts with size of variable table	POKE 109,0 POKE 110,8	: array : variables : below hi-res
End of string variables	$6F $70	111 112	adjusts with size of string table	POKE 111,0 POKE 112,32	: strings put : below hi-res : page 1
HIMEM beginning of string data	$73 $74	115 116	top of usable memory, low end of DOS buffers	POKE 115,0 POKE 116,32	: HIMEM at : $2000, below : hi-res page 1

Listing 1

```
100  HOME : PRINT "VARIABLE POINTER CHANGE TEST"
110  PRINT : PRINT : INVERSE : PRINT "BEFORE MOVE": NORMAL : PRINT : PRINT

120  GOSUB 1000: REM  PRINT POINTER LOCATIONS
130  GOSUB 1300: REM  CHANGE POINTERS
140  GOSUB 1200: REM  DECLARE VARIABLES
150  GOSUB 1400: REM  RETRIEVE VARIABLES
200  END
1000  REM  PRINT POINTER LOCATIONS
1005  PRINT  PEEK (103) + 256 *  PEEK (104);"  PROGRAM BEGINNING"
1010  PRINT  PEEK (105) + 256 *  PEEK (106);"  LOMEM,VARIABLES START"
1020  PRINT  PEEK (107) + 256 *  PEEK (108);"  END OF SIMPLE VARIABLES"
1030  PRINT  PEEK (109) + 256 *  PEEK (110);"  END OF ARRAYS"
1040  PRINT  PEEK (111) + 256 *  PEEK (112);"  END OF STRINGS"
1050  PRINT  PEEK (115) + 256 *  PEEK (116);"  HIMEM,START OF STRINGS"
1090  RETURN
1200  REM  DECLARE SOME VARIABLES
1210 A = 3:B = 6:C% = 7
1220 A$ = "LLLLLLLLLLL":B$ = "UUUUUUUUUU"
1225  PRINT : PRINT
1230  INPUT "NEW STRING?";C$
1232  PRINT : PRINT
1235  DIM X(1,1,1)
1240  PRINT : PRINT : INVERSE : PRINT "AFTER VARIABLES DEFINED": NORMAL :
      PRINT : PRINT
1250  GOSUB 1000
1290  RETURN
1300  REM  CHANGE POINTERS
1310  POKE 105,1: POKE 106,8: REM  LOMEM
1320  POKE 107,1: POKE 108,8: REM  SIMPLE VAR END
1330  POKE 109,1: POKE 110,8: REM  ARRAYS
1340  POKE 111,255: POKE 112,31: REM  STRINGS
1350  POKE 115,255: POKE 116,31: REM  HIMEM
1360  PRINT : PRINT : INVERSE : PRINT "AFTER POINTER CHANGE": NORMAL : PRINT
     : PRINT
1370  GOSUB 1000
1390  RETURN
1400  REM  TEST FOR VARIABLE RETRIEVAL
1410  PRINT : PRINT : INVERSE : PRINT "VARIABLE RETRIEVAL": NORMAL : PRINT
     : PRINT
1420  PRINT "A=";A
1430  PRINT "B=";B
1440  PRINT "C%=";C%
1450  PRINT "A$=";A$
1460  PRINT "B$=";B$
1465  PRINT "C$=";C$
1470  RETURN
```

you an effective program space of 14K, instead of the 6K you would have if you used page 1. To use page 2, just use HGR2 instead of HGR.

But there are some problems with this method. First of all, you cannot use the four lines of text below the graphics screen with HGR2. This will not be a problem if you are using some sort of hi-res character generator like the DOS TOOL KIT's HRCG. The other problem is that you may want to use both of the hi-res pages, say for page-flipping. In this case you would be back to the original 6K limitation.

The next method that might be useful is to relocate the program above the hi-res page (or pages). This is done by POKEing values into the memory locations that Applesoft looks at to see where to LOAD or RUN a new program. These POKEs must be done before the program is loaded or run, say from a "hello" program. Here are the POKEs:

to load above page 1

```
POKE 103,1
POKE 104,64
POKE 16384,0 : REM A 0 MUST BE
               PLACED IN MEMORY
               JUST AHEAD OF WHERE
               THE PROGRAM WILL
               LOAD
```

to load above page 2

```
POKE 103,1
POKE 104,96
POKE 24576,0
```

If the program is loaded above page 1 you will have about 22K (in a 48K system) for the program and variables. If you load above page 2, you will have about 14K. The 6K of memory below page 1 ($800-$2000) will remain unused by Applesoft.

Once you have loaded your program above a hi-res page you will probably want to figure out some way to use the memory that is just sitting empty below page 1. Again, there are several options. Here are just a few:

1. You can locate shape tables, especially long tables that will not fit at location $300, below the hi-res page, beginning at $800. The table can either be POKEd into that memory range or BLOADed at $800. Then the pointers at 232 and 233, which tell Applesoft the location of the current shape table, should be POKEd to point to this location. You could put several tables below page 1 and change the pointers as the program used one or another of the tables.
2. The DOS TOOL KIT's high-resolution character generator (HRCG) can be forced to load below page 1 by modifying to LOADHRCG program from the TOOL KIT. Insert the statement ' HIMEM: 8190 ' in the LOADHRCG program just before the step that says ' PRINT CHR$(4); "BLOAD RBOOT" '. RBOOT uses HIMEM to determine where to load the character generator. By giving a value of 8190 for HIMEM, the program is fooled into putting the character generator below page 1, rather than at the top of memory. You must insert one more step in the LOADHRCG program: ' HIMEM: 38400 ' is needed to reset HIMEM to the top of memory, just below the DOS buffers. This step should come after the step that reads ' CALL ADRS: REM INITIALIZE HRCG '. The value you use to reset HIMEM will depend on the size of your system; the value given is for a 48K Apple. Check the DOS manual for the figures for other size systems.
3. If you are writing a very large program that uses lots of variables, especially arrays, you may find that the variables will overwrite the strings, or *vice versa*, and then you have problems. Applesoft uses a set of pointers that tell the program where to store variables as they are encountered. If you change the pointers you can fool the program into using the memory below page 1 for variable storage, thus freeing room at the top of memory for your program.

See the chart in figure 1 for a more complete description of the pointers. Here are the POKEs that will cause a program loaded above the hi-res area to place its variables in the memory between $800 and $2000. Note: *These POKEs must be done before any variables are used by the program.*

```
POKE 105,0
POKE 106,8 : REM LOMEM AT $800

POKE 107,0
POKE 108,8 : REM SIMPLE VARIABLES
                 ENTERED AT $800
```

Listing 2

```
]RUN LISTING 1
VARIABLE POINTER CHANGE TEST

BEFORE MOVE

16385  PROGRAM BEGINNING
17481  LOMEM,VARIABLES START
17481  END OF SIMPLE VARIABLES
17481  END OF ARRAYS
38400  END OF STRINGS
38400  HIMEM,START OF STRINGS

AFTER POINTER CHANGE

16385  PROGRAM BEGINNING
2049   LOMEM,VARIABLES START
2049   END OF SIMPLE VARIABLES
2049   END OF ARRAYS
8191   END OF STRINGS
8191   HIMEM,START OF STRINGS

NEW STRING?ANYSTRING

AFTER VARIABLES DEFINED

16385  PROGRAM BEGINNING
2049   LOMEM,VARIABLES START
2091   END OF SIMPLE VARIABLES
2142   END OF ARRAYS
8182   END OF STRINGS
8191   HIMEM,START OF STRINGS

VARIABLE RETRIEVAL

A=3
B=6
C%=7
A$=LLLLLLLLLLL
B$=UUUUUUUUUU
C$=ANYSTRING

]
```

```
POKE 109,0
POKE 110,8 : REM ARRAY VARIABLES
             ENTERED AT $800
POKE 111,0
POKE 112,32 : REM STRINGS ENTERED
              AT $2000
POKE 115,0
POKE 116,32 : REM HIMEM AT $2000
```

These values are adjusted dynamically by Applesoft as variables are encountered. This is why many of the pointers are set to the same value initially. You may want to set the pointers differently to allow room below hi-res page 1 for some of the shape tables or routines that were discussed in the first part of the article. Play around with the values until you can use as much of the Apple's memory as possible.

Listing 1 tests the use of these pointers. The program must be loaded above one of the hi-res pages, as discussed earlier. The program displays the pointer values before any manipulation, after they are lowered, and finally after a selection of variables is defined by the program. Listing 2 is the output of this program. It shows how the variable pointers are high, above the program, then shift low, below hi-res page 1, then adjust with the definition of variables.

Once these values are set this way, attempts to add lines to the program will result in an "OUT OF MEMORY" error, because HIMEM is lower than the end of the program. Otherwise, the program should run normally and the memory manipulations will be transparent to the user.

I cannot hope to have covered all the tricks that can be used to get the most out of your Apple, but I hope that the ideas I have put forward will allow you to do some exploring and manipulating on your own.

References:

1. *Applesoft Basic Programming Reference Manual*, Apple Computer Co., 1978, pp 127, 137, 140.
2. Lechner and Worth, *Beneath Apple DOS*, Quality Software, 1981, pg. 8-42.
3. Wagner, "Assembly Lines, Part 17," *Softalk*, February, 1982.

Dan Weston is currently teaching a self-contained eighth grade in Brooks, Oregon. Contact Mr. Weston at 195 23rd NE, Salem, OR 97301.

MICRO

Atari Character Graphics from BASIC, Part 2

by Paul Swanson

The author adds fine scrolling to his character animation program, and introduces programming ANTIC's display list.

Character Graphics II
requires:
Atari 400/800

Last month (53:84) I showed you a very simple method for using a custom character set in BASIC. The amount of memory saved by using character graphics instead of map-mode graphics is substantial. A mode 7 screen would normally use almost 4K of memory. Replacing it with character graphics requires ½K for the character set plus about 240 bytes for the 20 × 12 character screen, for a total of about 770 bytes.

Of course, character graphics does have its drawbacks. For example, how do you move an image across the screen without having it "jump" from one character position to the next? The normal ways to put characters on the screen don't offer many alternatives. You could invent additional characters to mimic the movements. All that involves a lot of programming and many characters for each figure on the screen.

Are character-graphics screens dedicated to only those applications where nothing on the screen moves unless you introduce a player or missile? Are they just to display a pretty background for your program?

ANTIC

Fortunately, you have an Atari computer. The Atari has not one, but two microprocessors you can program. The Atari has one processor, a 6502, which functions as the "brain" of the system. It has another, called ANTIC, that controls just the screen display.

ANTIC's language is a machine language, which means it is all numbers, but the language is easy to learn because there are only a few instructions. Its program is called a display list. Each instruction does something on the screen taken in order from the top of the screen to the bottom.

ANTIC's program usually starts with "blank 8 lines" instructions, using three of them so that the displayed images are all visible on the screen. Televisions are set up with "overscan," which means that part of the actual picture is off the screen in all four directions so that the movies and commercials won't have borders around them. On the computer, you usually want to see the whole screen, so the borders are not so annoying.

Once you have the blank lines out of the way, you need an instruction called a Load Memory Scan (LMS) instruction, which tells ANTIC where the next line is in memory that you want displayed. This command is three bytes long. The first byte is the LMS instruction and the next two define the memory location.

If you continue using LMS instructions for each line on the screen, you can put every line on the screen in a different part of memory. This is not usually done. Instead, you can follow an LMS instruction with a Mode Line instruction, which tells ANTIC to keep incrementing the memory "pointer" for each consecutive line.

The last instruction in the display list is the "Jump on Vertical Blank" (JVB) instruction. A JVB tells ANTIC to wait until the television picture is completed, which is the end of a sixtieth-of-a-second cycle, then "jump" to the location it has in the two bytes that follow the JVB. The LMS and the JVB are both three-byte instructions. The Blank Line and Mode Line instructions are both one byte long.

All of this may mean very little to you without an example. To put an example together with the numbers in decimal would also mean very little because ANTIC interprets them in binary. Hexadecimal is a good compromise. In hexadecimal and decimal, the following is the program for ANTIC that BASIC sets up in response to a GRAPHICS 18 statement (mode 2 without a text window):

Hex	Decimal	Instruction
70	112	Blank 8 lines
70	112	
70	112	
47	71	LMS ANTIC mode 7,
xx	xx	= BASIC mode 3,
xx	xx	plus 2-byte memory location.
07	7	Display ANTIC mode 7
07	7	
07	7	
07	7	
07	7	
07	7	
07	7	
07	7	
07	7	
07	7	
07	7	
41	65	JVB
xx	xx	
xx	xx	

As you can see, the ANTIC numbers for the modes are not the same as the BASIC numbers for the modes. As you become more familiar with display lists, you will find that there are more modes available than the few that BASIC allows. For this example, you need to know that BASIC's mode 2 is ANTIC's mode 7.

So, what has all this to do with moving characters around on a character-graphics screen? If there is nothing more to gain than being able to see what ANTIC's program looks like, then this looks like a mildly educational exercise, right? Read on.

Fine Scrolling

ANTIC has a few more little twists to it than just displaying normal characters on the screen, or even modified characters. It has fine scrolling capabilities, both horizontally and vertically. To use them, you must set the "mode-line" instruction of each line in the display list that corresponds to the line you want to be able to scroll. That's why you need to know what the display list looks like.

Fine scrolling allows you to move the entire character row (or all of the rows if you like, which is what the program at the end of this article does) one dot at a time horizontally and/or vertically. Using a combination of fine scrolling and moving the entire character a whole character position will allow you to move a character display smoothly.

To see how to enable the fine scrolling function, you must first take apart the two instructions that result in a line on the display. In hexadecimal, for an ANTIC mode 7 display, these instructions are the LMS instruction, 47, and the mode-line instruction, 07. In binary, these are 0100 0111 and 0000 0111. Using the numbering that makes the leftmost binary digit number 7 and the rightmost one number zero, the binary digits we want are numbers 4 and 5. Number 4 enables horizontal scrolling: 0101 0111 and 0001 0111. Number 5 enables vertical scrolling: 0110 0111 and 0010 0111. Setting both to one enables both: 0111 0111 and 0011 0111.

To use these from BASIC, we need the decimal equivalent. To enable horizontal and vertical scrolling, add decimal 48, which is binary 0011 0000. That changes the LMS from 71 to 119 and the mode-line instruction from 7 to 55. In decimal, the final display list for a GRAPHICS 18 screen would be: 112 112 112 119 xx xx 55 55 55 55 55 55 55 55 55 55 55 65 xx xx, where xx stands for a memory location not yet determined.

Special Memory Locations

Now that we know what to do with the display list, where do we find it? There are special memory locations in the Atari that can give us, or accept from us, all kinds of information. For example, locations 560 and 561 contain the location of the start of the display list. If we declare a GRAPHICS 18 screen in a program, then set a variable to PEEK(560)+PEEK(561)*256, that variable will have the location of the display list ANTIC is using. You can write a new location to 560 and 561 if you form one on your own, but it is usually easier to just modify the one BASIC has already set up.

In the program, we can find the display list, then modify all the instructions that display a line on the screen by adding 48 to it. That will enable the scrolling we want. We also need a few other special locations. Two locations are required to put the amount of

```
1 REM *** Custom Character Set ***
2 REM *** Program for Part II  ***
3 REM
4 REM
5 REM *** Program by...        ***
6 REM ***        Paul S. Swanson ***
7 REM
8 REM
9 REM --- Calc. position in mem. ---
10 DIM S$(1024)
20 A=ADR(S$)
30 B=INT(A/512+1)*2
40 CBASE=B*256-A+1
47 REM
48 REM
49 REM --- Clear S string ---
50 S$(1)=CHR$(0)
60 S$(1024)=CHR$(0)
70 S$(2)=S$(1)
77 REM
78 REM
79 REM --- Move standard set down ---
80 FOR I=0 TO 511
90 S$(CBASE+I,CBASE+I)=CHR$(PEEK(I+57344))
100 NEXT I
107 REM
108 REM
109 REM --- Set # to character ---
110 FOR I=24 TO 31
120 READ N
130 S$(I+CBASE,I+CBASE)=CHR$(N)
140 NEXT I
147 REM
148 REM
149 REM --- GR.2 - No text window ---
150 GRAPHICS 18
157 REM
158 REM
159 REM --- Find Display List ---
160 DLIST=PEEK(560)+PEEK(561)*256
162 SLOC=PEEK(DLIST+4)+PEEK(DLIST+5)*256
167 REM
168 REM
169 REM --- Set scroll enables ---
170 POKE DLIST+3,PEEK(DLIST+3)+48
180 FOR I=6 TO 16
190 POKE DLIST+I,PEEK(DLIST+I)+48
200 NEXT I
207 REM
208 REM
209 REM --- Initialize position ---
210 VPOS=96
220 HPOS=80
222 POKE 756,B
224 WING=1
226 S=14
227 REM
228 REM
229 REM --- Draw character in position ---
230 V=INT(VPOS/16)
232 IF WING=1 THEN SOUND 0,10,0,6
240 VSCROL=VPOS-V*16
250 H=INT(HPOS/8)
260 HSCROL=HPOS-H*8
262 IF WING=1 THEN WING=2:S$(CBASE+25,CBASE+25)=CHR$(0):S$(CBASE+26,
    CBASE+26)=CH R$(231):GOTO 266
264 WING=1:S$(CBASE+25,CBASE+25)=CHR$(195):S$(CBASE+26,CBASE+26)=CHR$(36)
266 POKE 559,0
270 POKE SLOC+P,0:P=V*24+H:POKE SLOC+P,3
280 POKE 54276,HSCROL
290 POKE 54277,15-VSCROL
292 POKE 559,34
294 SOUND 0,10,0,2
297 REM
298 REM
299 REM --- Read Joystick ---
300 OLDS=S:S=STICK(0)
310 IF S=15 THEN S=OLDS
320 VMOVE=0
330 HMOVE=0
340 IF S=9 OR S=13 OR S=5 THEN VMOVE=2
350 IF S=10 OR S=14 OR S=6 THEN VMOVE=-2
360 IF S>4 AND S<8 THEN HMOVE=1
370 IF S>8 AND S<12 THEN HMOVE=-1
380 IF VMOVE+VPOS>=0 AND VMOVE+VPOS<191 THEN VPOS=VPOS+VMOVE
390 IF HMOVE+HPOS>=0 AND HMOVE+HPOS<192 THEN HPOS=HPOS+HMOVE
400 IF VMOVE=2 THEN WING=2
410 GOTO 230
1000 DATA 0,195,36,24,24,36,0,0
```

scrolling we want and one is required to turn ANTIC off when we change the scrolling values. If we don't turn ANTIC off, we get some very annoying "snow" on the screen. Other, less predictable things have been reported happening when ANTIC was on when scrolling values were changed.

The Program

It is always easier to see what is happening when you have a real example in front of you. Enter the program into your Atari so that your reading keeps up with the amount you have entered. That will make it easer to see what's going on.

Lines 1 through 150 are actually all explained in Part 1.

Lines 10 through 40 find a ½K boundary in the S$ string. Lines 50 through 70 clear the S$ string to ASCII code zeroes. Lines 80 through 100 move the "built-in" character set down (not required for this program). Lines 110 through 140 insert the special character into the set, and line 150 declares graphics mode 2 without a text window.

After you have lines 1 through 150 typed into your Atari, enter through line 162. Line 160 sets the variable DLIST equal to the location where the display list created by BASIC starts. Line 162 looks at the memory location in that display list where the screen starts in memory. This is required later on.

From there through line 200, the program sets the enable bits on all of the ANTIC instructions that display a line on the screen. Since you know that one will be changed from 71 to 119 and the other from 7 to 55, you could use the number 119 in line 170 instead of PEEK(DLIST+3)+48 and 55 in line 190 instead of PEEK(DLIST+I)+48. The more generalized form in the program will make it more easily adaptable to other display list applications.

Moving up to line 224 presents a few unusual statements. These statements initialize everything that didn't fit into the above categories. VPOS and HPOS are initialized to somewhere in the middle of the screen. VPOS and HPOS are the vertical and horizontal positions — not character positions, but the positions in dots — of the figure we will be moving. Each character is 16 dots high and 8 dots wide. Line 222 sets the character set base address. This statement was also used in Part 1. WING (line 224) keeps track of which position the "wings" are in. S is initialized so that the bird will be flying up when the main part of the program begins.

Lines 230 through 260 calculate the character position on the screen with the remainder from the division determining the fine-scrolling amount. Since the characters are 16 dots high and 8 dots wide in mode 2, these are the two values by which we divide. The SOUND statement doesn't have anything to do with the position, exactly. It is placed there as a matter of timing so that the "wingflapping" noise is initiated properly.

Lines 262 and 264 take care of putting the wings in the correct position, alternating by setting WING to 2 if it is 1 and to 1 if it is 2. Again, the subscripts for the S$ string and the values for the CHR$ function are explained in Part 1.

Finally, we get to where the real action takes place in the program. Line 266 puts a zero in location 559, which turns ANTIC off as soon as it finishes the current sweep of the screen. The next statement will not need ANTIC off, but will function as enough of a delay to guarantee that the current sweep of the screen is completed before the fine scrolling values are POKEd into the special memory locations.

Line 270 erases the old position of the figure, then calculates the new position and puts the figure there.

POSITION statements do not work when you enable the scroll because ANTIC automatically assumes a "wide playfield" instead of a normal one, which has 24-character lines instead of 20-character lines. That is why the formula P=V*24+H is required. The variable P will be the "old" position for the next loop, so that you don't have to calculate anything to erase the old figure before you POKE the new one into place.

Lines 280 and 290 set the horizontal and vertical values into the special memory locations reserved for them. Note that the vertical scroll value runs in the opposite direction and must be subtracted from its maximum value, 15, to get the correct one. When these values are POKEd, ANTIC is turned back on by POKEing 34 into location 559, as is done in line 292. Line 294 stops the "wing-flapping" sound and, like line 232, is positioned here for the sake of timing the sound.

Now that we have all of the statements in place for moving the figure around, we need some way to control where it moves. I chose the joystick for input. It is read at line 300. Line 310 causes all readings where the stick is in the "neutral" position to be ignored. These two statements will use the last non-neutral position for the direction, if the joystick is centered.

When you move the joystick, lines 320 through 390 interpret the movement into the new position by updating VPOS and HPOS. The vertical movements are all 2 dots at a time to compensate for the difference in the two dimensions so that the bird will fly up at about the same rate as it will fly horizontally. Lines 380 and 390 make sure the figure stays on the screen, then line 400 goes back to put the figure where you just moved it. Line 1000 is the character shape for the READ at line 120.

Now you can RUN the program and move the figure around the screen with your joystick. If you don't have a joystick, you should be able to figure out how to move it with the four arrow keys.

Notice that the figure does flash a little when you move it. This happens when you turn ANTIC off. This can be limited by decreasing the delay (line 270) after you turn ANTIC off. Remember that line 270 does not require that ANTIC be off, so it functions as the delay. You can shorten the delay as much as you like until you start getting snow on the screen when you move the figure horizontally. You can do this by breaking the line up so that the first statement executes before the POKE 559 statement. This snow is the problem you avoid by turning ANTIC off.

Note that I did include a few statements in the program that weren't described. These statements are set up to cause the bird to flap its wings when it is moving horizontally or up (i.e., VMOVE does not indicate "down"). When the bird moves downward, the wings do not flap and the flapping sound stops.

The program is not the most efficient way to scroll character-graphics screens, but it does show the general idea of how it is done. You can make the program more efficient by doing things like replacing the part that reads and interprets the joystick with a faster routine. You may develop a different way of handling the positions from the VPOS and HPOS approach. If you keep the character position and the scroll values separately, you will not need the two divisions. If you do it right, you can gain some speed there. Remember that the program was written to be instructional rather than efficient, so you should find many areas where you can speed up.

Play around with the program until you get ideas on how to use character graphics for your own programming project. Your own project will familiarize you with the advantages of character graphics. Remember, too, that you are saving lots of memory.

MICRO

MICRO™

Apple Slices

By Tim Osborn

This month's column discusses one of the fundamental elements of any computer system, the block move. I also present a subroutine to perform block moves. If you find yourself saying "but the monitor already includes a block-move routine," read on; I'll show you why you may want to use my block move instead. I'll also show you why it is sometimes advantageous to use the monitor's routine.

What Is a Block Move?

A block move is a byte-by-byte movement of data from one range of memory to another. The area from where the data originates is called the source and the target area is called the destination. The length of the move is the difference between the source end and the source beginning plus one. The distance of the move is the difference between the source beginning and the destination beginning.

Let's use the monitor's routine to illustrate some examples of block moves. Type in the following sequence:

```
]CALL -151
  (enter the monitor)
*3000:01 02 03 04 05 06 07 08
  (initialize memory)
*3000 < 3002.3003M
  (move 3002 - 3003 to 3000 - 3001)
```

Now type:

```
*3000.3007
  (dump range of memory)
```

and receive the following dump:

```
3000- 03 04 03 04 05 06 07 08
```

Observe that the block move was successful. Now try this:

```
*3001 < 3000.3006M
  (move 3000 - 3006 to 3001 - 3007)
*3000.3007
  (dump range of memory)
```

and receive the following dump:

```
3000- 03 03 03 03 03 03 03 03
```

The computer first moved 3000 to 3001, then 3001 to 3002, and so on, so that the full destination range is filled with the same value.

This situation is called an overlap. When the destination is higher than the source combined with an overlap, it is necessary to move the data starting at the source end working toward the source beginning, which is called a right-move, as opposed to a left-move. By doing a right-move in the above example and dumping the range of memory, the following results would be obtained:

```
3000- 03 03 04 03 04 05 06 07
```

These results are correct.

A similar problem exists where the destination start address is lower than the source start address and the two ranges overlap. In this case the right-move (source-end first) will cause the same sort of problems and it is necessary to use the standard left-move (source start first) to avoid these problems. When the ranges do not overlap, it makes no difference which type of move is used, either the right-move or the left-move.

```
                ;*****************************
                ;*                           *
                ;*         BLOCK-MOVE        *
                ;*         TIM OSBORN        *
                ;*                           *
                ;* A P P L E   S L I C E S   *
                ;*                           *
                ;*****************************
                ;
                ;PAGE ZERO EQUATES
                ;
                ;A1 THRU A4 ARE PASSED FROM THE MONITOR
                ;A5 IS COMPUTED INTERNALLY
                ;
003C            A1L      EPZ $3C            ;A1=THE START OF SOURCE
003D            A1H      EPZ $3D
003E            A2L      EPZ $3E            ;A2=THE END OF SOURCE
003F            A2H      EPZ $3F
0042            A4L      EPZ $42            ;A4=THE START OF DESTINATION
0043            A4H      EPZ $43
0044            A5L      EPZ $44            ;A5=THE END OF DESTINATION
0045            A5H      EPZ $45
0800            ;
0800            ;OTHER EQUATES
0800            ;
0800            ;CTRL-Y VECTOR LOCATION
03F8            USRADR   EQU $3F8
0800            ;
0300                     ORG $300
0300                     OBJ $800
0300            ;  '
0300            ;START WILL ESTABLISH THE CONTROL-Y
0300            ;VECTOR. WHEN THE MONITOR ENCOUNTERS
0300            ;A CONTROL-Y IT WILL JUMP TO ENTRY
0300            ;
0300 A9 4C      START    LDA #$4C           ;JUMP INSTRUCTION
0302 8D F8 03            STA USRADR
0305 A9 10               LDA #ENTRY         ;LOW BYTE OF ENTRY ADDRESS
0307 8D F9 03            STA USRADR+1
030A A9 03               LDA /ENTRY         ;HIGH BYTE OF ENTRY ADDRESS
030C 8D FA 03            STA USRADR+2
030F 60                  RTS                ;INITIALIZATION COMPLETE
0310            ;
0310            ;DISTL+DISTH RECYCLE START'S STORAGE
0310            ;SINCE START IS ONLY NEEDED AT BRUN
0310            ;
0300            DISTL    EQU START          ;INTERNAL STORAGE FOR
0301            DISTH    EQU START+1        ;THE DISTANCE OF THE MOVE
0310            ;
0310            ;ENTRY IS THE MAIN ENTRY POINT
0310            ;WHICH IS REACHED WHEN THE CTRL-Y
0310            ;IS ENCOUNTERED
0310            ;
0310 38         ENTRY    SEC                ;COMPUTE DISTANCE
0311 A5 42               LDA A4L            ;BY SUBTRACTING
0313 E5 3C               SBC A1L            ;THE SOURCE START (A1) FROM
0315 8D 00 03            STA DISTL          ; THE DESTINATION START (A4)
0318 A5 43               LDA A4H
031A E5 3D               SBC A1H
031C 8D 01 03            STA DISTH
031F 18                  CLC
```

(Continued)

The Block-Move Routine

BLOCK-MOVE (listing 1) incorporates the ideas expressed above by performing all block moves where the destination address is higher than the source address with a right-move and all moves where the destination address is lower than the source address with a left-move.

To install BLOCK-MOVE you must BRUN the object code. The routine START will enable the CTRL-Y vector by initializing a JMP $310 (the address of BLOCK-MOVE's ENTRY routine) into $3F8 through $3FA. When the monitor encounters a CTRL-Y (control key pressed simultaneously with the Y key) it will effectively JSR to $3F8. After START initializes the CTRL-Y vector the monitor will effectively JSR to ENTRY upon encountering a CTRL-Y.

The syntax to use BLOCK-MOVE is exactly the same as that to use the monitor M command:

(dest) < (start).(end)

The (end) statement is then followed by a CTRL-Y instead of the usual M. When execution reaches ENTRY the monitor subroutines have already converted:

1. (dest) to a sixteen-bit integer stored at $42 through $43 (A4L through A4H or A4).
2. (start) to a sixteen-bit integer stored at $3C through $3D (A1L through A2H or A1).
3. (end) to a sixteen-bit integer stored at $3E through $3F (A21 through A2H or A2).

All of the above values are in modulo-256 form in low-byte, high-byte order. A user-written program can make use of BLOCK-MOVE by setting up the above values and JSRing to ENTRY.

BLOCK-MOVE must first compute the distance (DISTL through DISTH, $300 through $301) by subtracting A1 from A4 and storing the results at DIST (see lines 54 through 60). After this, DIST is added to A2 to obtain the destination end address — A5 (lines 61 through 67). Next destination start is compared to the source start in lines 68 through 76. If the source start is greater than the destination start, the MOVELT routine is used to perform the block move. If the destination start is greater than the source start then MOVERT routine is used. If the destination start is equal to the source start then no move is performed and an RTS is done at line 74.

MOVERT takes bytes starting at A2 (source end) and moves them to A5 (destination end). It decrements both values moving another byte after each decrement until A5 = A4, which means the destination pointer is equal to the destination beginning. (See lines 84 through 101.)

MOVELT takes bytes starting at A1 (source start) and moves them to A4 (destination start). It increments both values moving another byte after each increment until A4 = A5, which means that the destination pointer is equal to the destination end. (See lines 107 through 122.)

Both routines return to the monitor (or user-written program) when they have completed their task. Experiment with BLOCK-MOVE and the monitor move command to get a feel for the differences between the two routines.

The advantage of using the monitor subroutine is that it allows you to initialize memory to desired patterns of byte values. It can be very handy, for instance, to initialize ranges of memory to binary zeros, but it is not limited to that. All sorts of patterns can be created depending upon the nature of the overlap. For those times when you are moving data forward with overlapping ranges of memory, use BLOCK-MOVE.

MICRO

```
0320 AD 00 03   62            LDA DISTL       ;ADD DISTANCE
0323 65 3E      63            ADC A2L         ;TO SOURCE END
0325 85 44      64            STA A5L         ;TO OBTAIN DESTINATION END
0327 AD 01 03   65            LDA DISTH
032A 65 3F      66            ADC A2H
032C 85 45      67            STA A5H
032E A5 42      68            LDA A4L         ;SEE IF DESTINATION IS
0330 C5 3C      69            CMP A1L         ;GREATER OR LESS THAN SOURCE
0332 D0 07      70            BNE ENTRY1
0334 A5 43      71            LDA A4H         ;ALSO CHECK TO MAKE SURE
0336 E5 3D      72            SBC A1H         ;THAT A1 DOES'NT = A4
0338 D0 05      73            BNE ENTRY2
033A 60         74            RTS             ;A1=A4 SO RETURN
033B A5 43      75   ENTRY1   LDA A4H
033D E5 3D      76            SBC A1H         ;SOURCE
033F B0 03      77   ENTRY2   BCS MOVERT      ;IT'S GREATER -MOVE RIGHT
0341 4C 67 03   78            JMP MOVELT      ;IT'S LESS    -MOVE LEFT
0344            79   ;
0344            80   ;MOVERT MOVES THE DATA STARTING AT THE
0344            81   ;SOURCE END WORKING TOWARDS THE
0344            82   ;SOURCE BEGINNING
0344            83   ;
0344 A0 00      84   MOVERT   LDY #$00
0346 B1 3E      85   MOVERT1  LDA (A2L),Y     ;GO FROM OLD END
0348 91 44      86            STA (A5L),Y     ;TO NEW END
034A C6 44      87            DEC A5L         ;DECREMENT DESTINATION POINTER

034C A9 FF      88            LDA #$FF
034E C5 44      89            CMP A5L         ;IF A5L=$FF
0350 D0 02      90            BNE MOVERT2
0352 C6 45      91            DEC A5H         ;THEN DECREMENT A5H
0354 C6 3E      92   MOVERT2  DEC A2L         ;DECREMENT SOURCE POINTER
0356 C5 3E      93            CMP A2L         ;IF A2L=$FF
0358 D0 02      94            BNE MOVERT3
035A C6 3F      95            DEC A2H         ;THEN DECREMENT A2H
035C A5 44      96   MOVERT3  LDA A5L         ;SEE IF WE HAVE MOVED
035E C5 42      97            CMP A4L         ;ALL BYTES (A5=A4)
0360 A5 45      98            LDA A5H
0362 E5 43      99            SBC A4H
0364 B0 E0     100            BCS MOVERT1     ;NO MUST CONTINUE
0366 60        101            RTS             ;ALL DONE
0367           102   ;
0367           103   ;MOVELT MOVES THE DATA STARTING AT
0367           104   ;THE BEGINNING OF THE SOURCE WORKING
0367           105   ;TOWARDS THE END OF THE SOURCE
0367           106   ;
0367 A0 00     107   MOVELT   LDY #$00        ;INITIALIZE Y-REG
0369 B1 3C     108   MOVELT1  LDA (A1L),Y     ;MOVE FROM OLD SOURCE
036B 91 42     109            STA (A4L),Y     ;TO NEW DESTINATION
036D E6 3C     110            INC A1L         ;INCREMENT SOURCE POINTER
036F D0 02     111            BNE MOVELT2     ;IF EQUAL TO ZERO
0371 E6 3D     112            INC A1H         ;THEN INCREMENT HIGH BYTE
0373 E6 42     113   MOVELT2  INC A4L         ;INCREMENT DEST POINTER
0375 D0 02     114            BNE MOVELT3     ;IF EQUAL TO ZERO
0377 E6 43     115            INC A4H         ;THEN INCREMENT HIGH BYTE
0379 A5 44     116   MOVELT3  LDA A5L         ;CHECK TO SEE IF
037B C5 42     117            CMP A4L         ;WE ARE DONE WITH MOVE (A4=A5)

037D A5 45     118            LDA A5H
037F E5 43     119            SBC A4H
0381 B0 E6     120            BCS MOVELT1     ;NOT DONE, LOOP
0383 60        121            RTS             ;ELSE RETURN TO CALLER
0384           122            END
```

Reviews in Brief

Product Name: **The Programmable Cube**
Equip. req'd: Apple II with 48K or 64K and DOS 3.3
Price: $34.95 includes diskette and extensive documentation
Manufacturer: Metacomet Software
P.O. Box 31337
Hartford, CT 06103

Description: This program will solve a Rubik's cube. "Cube" will scramble a cube and solve it for you. It also has an option to make designs and patterns. It is easy to use and relies on standard cube notation for entering moves.

Pluses: *The Programmable Cube* comes with an extensive user's guide to teach how to write programs to make designs or even solve cubes based on your own cube-solving strategy. It has a "mirror" behind the graphics cube to show the obverse sides of the cube. The program also includes simulated rotation of the cube as moves are made. You can enter a cube of your choice or choose the order of the colors on the cube.

Minuses: The language, while not difficult to learn, is similar to many of the graphics-control programs and takes a fair amount of time to become familiar with it.

Skill level required: Ability to follow directions is needed to run the solving portion. Previous programming experience in any language would be a help to program in cube language.

Reviewer: Phil Daley

Product Name: **Ghost Gobbler**
Equip. req'd: TRS-80 Color Computer with 16K
Price: $21.95/cassette; $24.95/disk
Manufacturer: Spectral Associates
141 Harvard Ave.
Tacoma, WA 98466

Description: *Ghost Gobbler* is Spectral Associates' version of the popular arcade game, PAC-MAN. Using a joystick, you control a gobbler that travels around a maze eating dots. You must be wary of the four ghosts that also frequent the maze. If they catch you, you will be eaten. Your only protection is to eat an "energizer" dot, whereupon you can score points by gobbling the ghosts. There are 17+ screens, and extra points can be gained by gobbling bonus shapes. Extra men can be gained by scoring high.

Pluses: The game is in 6809 machine language, and comes on cassette. The program will load onto disk and execute properly from disk once loaded. The game is fast action, and operates smoothly in all skill levels. A teleportation spot allows the player to quickly escape to another section of the board. Sixteen skill levels are available, so novices and experts can compete on a more even scale.

Minuses: The game supports only one player at a time, though it does keep records of the ten highest scorers. It operates as a linear device rather than a switch type. Many users complain of the joystick action at first, but practice does improve its action.

Documentation: A single sheet instruction.

Skill level required: Anyone who can handle a joystick can play the game, but expert skills are required to get past the first two or three screens.

Reviewer: John Steiner

Product Name: **Speed Reader**
Equip. req'd: Apple II with Applesoft in ROM and DOS 3.3
Price: $
Manufacturer: Special Delivery Software
10260 Bandley Drive
Cupertino, CA 95014

Description: *Speed Reader* is a five-part reading program designed to improve comprehension and increase reading speed through the development of concentration, attention span, and more effective eye movement. Two copies of the *Speed Reader* Master Program diskettes, one copy of the *Speed Reader* Data diskette, and an easy-to-read manual are included in the software package. The main menu has five lessons: warm-up excercise (letters), warm-up exercise (words), eye movement exercise, column reading lesson, and reading passage lesson.

Pluses: The manual contains charts for each lesson on which progress can be recorded; user's scores are provided after each activity is completed. Several reading selections are offered in Lessons 3, 4, and 5, most of which are informative and interesting. You can increase reading speed, change column justification, and decrease window size in Lessons 3, 4, and 5.

Minuses: Vocabulary level and topics of several selections are beyond that of an average fifth-grade student. It is not clear to the user whether the RETURN key should be used to continue the program; no editing is permitted and selections cannot be added to the disk. Only one user can use *Speed Reader*; booting the diskette will not provide several users with the program.

Skill Level Required: Grades five through adult; private instruction. Spelling proficiency is a must as well as typing skills.

Reviewer: Cathy LaSalle

Product Name: **GR2716 ROM/EPROM Emulator**
Equip. req'd: Not applicable
Price: $78.00
Manufacturer: Greenwich Instruments Limited
U.S. Distributor: LMS Electronics
3401 Monroe Road
Charlotte, NC 28205

Description: The *GR2716 EPROM Emulator* is a pin-for-pin replacement of the 2716 EPROM for use during system

Reviews in Brief *(continued)*

development. The device consists of RAM memory and a lithium power cell housed in a 24-pin package about .6 inches high. When used in the read mode, the device is plugged into a normal system EPROM socket, and functions exactly like an EPROM. Connectors with leads are supplied also. The package is provided with three additional wirewrap connections on the end between pins 1 and 24. Two of these pins are connected to the system reset to prevent inadvertent writing to the memory during system restart. When the third pin, labeled WE, is connected to a normal system static-write enable signal, it causes the circuit to function as a static RAM chip. The battery is guaranteed to retain memory for three years, with ten years quoted as typical. Other versions are available to replace other EPROMS.

Pluses: This processor performs exactly as specified. The documentation is terse, but adequate. New literature has come out since this review. Sample WE circuits are given for several CPUs.

Minuses: The legs appear fragile and could be broken easily. The problem can be circumvented by installing the GR2716 into a 24-pin soldertail socket and then plugging the entire assembly into the system. One other minor problem is the placement of the WE and Reset pins on the end of the device. There have been fit problems on crowded boards, but the chip-in-a-socket approach also solved this problem by raising the chip above board level.

Skill level required: Reasonably serious hardware and machine-language system software developer.

Reviewer: Wayne D. Smith

Product Name:	**Telewriter (disk version)**
Equip. req'd:	TRS-80C, 16K, RS disk system, printer
Price:	$49.95 cassette, $59.95 disk
Manufacturer:	Cognitec 704 Nob Ave. Del Mar, CA 92014 (714) 755-1258

Description: *Telewriter* is a word processor for the TRS-80C. The editor features a cursor-oriented, 51- × 24-character display with real lower-case characters. The graphics screen is used for text display, and provides a much greater area of visible text than most color computer word processors.

Pluses: *Telewriter* is one of the better editors I have seen. Many features, including embedded printer commands, not found on more expensive processors, are available in *Telewriter*.

Minuses: Requires a disk I/O program on disks. The binary file format adds extra steps when using *Telewriter* with an ASCII file. Neither horizontal scrolling nor right justification is supported.

Documentation: Seventy pages of well-written reference material are included.

Skill level required: No previous experience with word processors is required.

Reviewer: John Steiner

MICRO

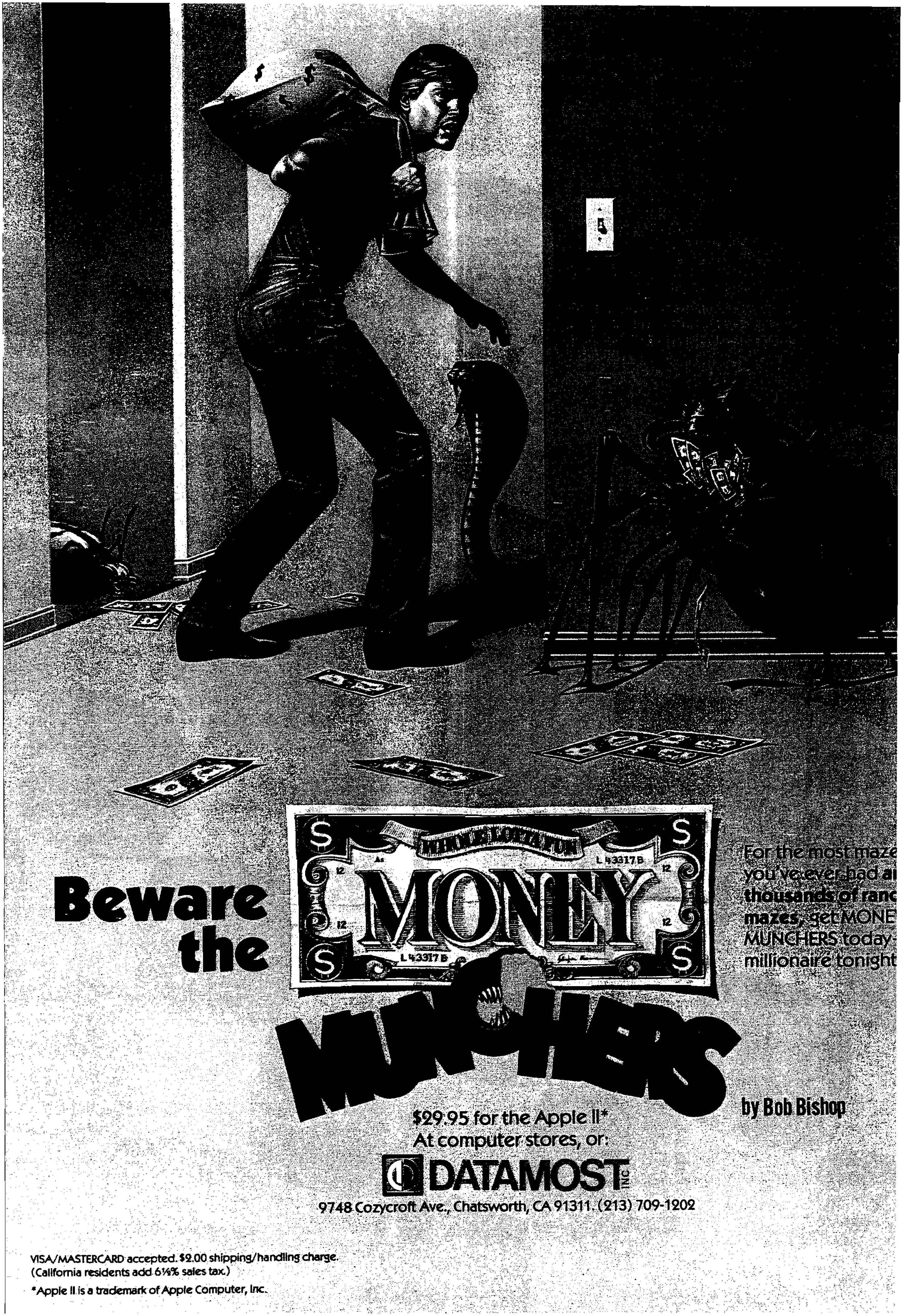
MONSTER MASH

It is late at night in a monster infested graveyard and you have been given the job of keeping the monsters in. All you have between you and complete chaos is a new MonsterMasher System and quick reflexes.

Monster Mash is an original and unique arcade action game written in assembly language for the Apple II and Apple /// (in emulation mode).

$29.95

The Software Farm
3901 So. Elkhart
Aurora, CO. 80014
PH: (303) 690-7559


WHOLE LOTTA FUN
MONEY
Beware the
MUNCHERS
For the most maze
you've ever had a
thousands of ranc
mazes, get MONE
MUNCHERS today
millionaire tonight
by Bob Bishop
$29.95 for the Apple II*
At computer stores, or:
DATAMOST INC.
9748 Cozycroft Ave., Chatsworth, CA 91311. (213) 709-1202
VISA/MASTERCARD accepted. $2.00 shipping/handling charge.
(California residents add 6½% sales tax.)
*Apple II is a trademark of Apple Computer, Inc.

Getting Around the Apple Hi-Res Graphics Page

by Eagle I. Berns

This article describes a method to split an Applesoft BASIC program in order to make available the core both above and below the hi-res pages. This allows the program to utilize the graphics area without having to sacrifice portions of memory.

GETTING requires:

Apple II with 32K
RENUMBER

If your Apple II has 48K of memory, and you write a program that starts at $0800 (hex) and goes to, say, $4100, you may notice that though there's still a lot of core available for your program, you have lost the ability to use either of the hi-res graphics pages (the first runs from 2000 to 3FFF and the second from $4000 to $5FFF). When this problem came up for me, I wondered if there might be some way to have my program bridge the gap over either or both the graphics areas and continue on the other side.

In reading current literature I found various references to ways in which this could be done. However, they all required multiple steps and patching source statements. What I wanted was an automatic process. There are several EXEC files used in the process I eventually developed, but the user of the program need only issue the one command "RUN SPLIT", and the rest of the process is automatic.

The EXEC files and utility program needed to do the job are listed at the end of this article. First, however, I will give a general description of how the task is accomplished, and then a detailed description of the program itself.

Basically, the statements of the Applesoft program to be split are scanned *via* their internal pointer links from statement to statement, until the last statement before location $2000 is reached. At this point a number of dummy statements are inserted. (This version uses the &Renumber utility from the Apple II tool kit to renumber the program in a way that allows the insertion of a number of dummy statements. A simple modification is necessary if some other renumber utility is used.) The utility then proceeds to find the link that exists between the location before $2000, and the location directly after $4000. It then finds the places to POKE, relinking the program across the dummy statements (where the graphics area is located). Since a LOAD will reset all links, the two POKEs must be a part of the original program. Also, since the POKEs must be there when the split is complete, they must be inserted before we begin the split, so as not to destroy the relative positioning of statements in the program.

Listing 1: **SPLIT**

```
100  TEXT : HOME
110  PRINT "ENTER NAME OF PROGRAM TO SPLIT": PRINT : PRINT : HTAB (7): INPUT
     A$
120  PRINT
130  PRINT "ENTER LOMEM ADDRESS FOR LOADING PROGRAM": PRINT : PRINT : HTAB
     (7): INPUT L
140  PRINT : PRINT "ENTER  1 IF SPLITTING OVER HGR1,"
150  PRINT "        2 IF SPLITTING OVER HGR2,"
160  PRINT "        0 IF SPLITTING OVER BOTH."
170  PRINT : PRINT "WHICH ";: INPUT S
180 A =  INT (L / 256)
190 B = L - A * 256
200  REM
210  POKE 210,B: REM SAVE LOMEM
220  POKE 211,A
230  REM
240  REM SET SPLIT BOUNDS.
250  REM 212 HOLDS START OF PAGE
251  REM 213 HOLDS LENGTH
260  REM 771 HOLDS LENGTH
270  REM
280  IF S = 1 THEN 320
290  IF S = 2 THEN 330
300  IF S = 0 THEN 340
310  GOTO 140
320  POKE 212,32: POKE 213,32: GOTO 360
330  POKE 212,64: POKE 213,32: GOTO 360
340  POKE 212,32: POKE 213,64
350  REM
360 D$ =  CHR$ (4)
370  PRINT D$;"OPEN SPL.SETUP"
380  PRINT D$;"DELETE SPL.SETUP"
390  PRINT D$;"OPEN SPL.SETUP"
400  PRINT D$;"WRITE SPL.SETUP"
410  PRINT "FP"
420  PRINT "RUN LOADAPA"
422  PRINT "POKE 103,";B + 1
424  PRINT "POKE 104,";A
426  PRINT "POKE ";B;",0"
430  PRINT "LOAD ";A$
440  PRINT "&R 2,1"
450  PRINT "1 POKE 0000,000:POKE 0000,00"
460  PRINT "SAVE SPLITPROG"
470  PRINT "EXEC SPL.FINDER"
480  PRINT D$;"CLOSE SPL.SETUP"
490  PRINT D$;"EXEC SPL.SETUP"
```

When the RUN SPLIT command is issued, you see the following:

ENTER THE NAME OF THE PROGRAM TO SPLIT
<type in the name>

ENTER LOMEM ADDRESS FOR LOADING PROGRAM
<Normally this is 2048 (or $0800), but you may change this, if you like>

ENTER 1 IF SPLITTING OVER HGR1
2 IF SPLITTING OVER HGR2
0 IF SPLITTING OVER BOTH

WHICH: <your choice depending on your needs>

Then wait as your screen goes through some contortions, writing messages, etc. Finally it will write out: END OF JOB. A program called SPLITPROG will have been created which, when run, will not be interfered with if hi-res graphics are used. Be sure to hold on to the original program for making modifications since SPLITPROG does not accept modifications gracefully.

Now I will present the programs and support utilities, with a description of their execution sequence.

SPLIT Description

Lines 180-350 are POKEd away for the rest of the utilities to use. Loc's 210 and 211 are for program start-up location; 212 and 213 are the addresses for the boundaries of the split.

Lines 370-400 and 580-590 create and execute SPL.SETUP, which does the following:

FP	<cleans things up a bit>
RUN LOADAPA	<from Apple II Tool Kit>
POKE 103,<addr1> POKE 104,<addr> POKE <loc>,0	<set up LOMEM for loading the user program to be split>
LOAD <program>	<loads the program>
&R 2,1	<renumbers to allow a dummy line 1 to be inserted that will be modified>
1 POKE 0000,000:POKE 0000,00	
SAVE SPLITPROG	<this will eventually become the split program>
EXEC SPL.FINDER	<goes off to find where to split>

SPL.FINDER Description

The EXEC is copied to the end of the user program and then that part is executed.

Lines 63980-63984 scan the program to be split to find the statement just preceding the hi-res page beginning.

Lines 63984-63986: if the program isn't large enough to require splitting, we stop here.

Lines 63987-63988 POKE away the statement number detected for later use.

Line 63989 invokes the next phase, which does the actual splitting process.

SPL.EXEC MAKER Description

Lines 5-6 compute the number of dummy REM SPLIT statements to be inserted to cover the appropriate graphics area.

Line 7 retrieves the line number where the split starts.

Lines 8-80 create and execute TEMPEXEC, which performs the following functions upon invocation:

Lines 31-34 set up the appropriate LOMEM value and then load SPLITPROG.

Line 35 renumbers the program to allow insertion of the REM split statements.

Line 36 adds a GOTO statement around the REMs so they can't be executed.

Lines 37-60 add the dummy REM statements.

Line 65 EXECs the final phase of the splitting process, which creates the appropriate POKE for line 1 to relink the final program execution.

***Listing 2:* SPLIT FINDER**

```
63980 I = 1 + PEEK (210) + PEEK (211) * 256
63981 J = PEEK (I + 1)
63982 IF J = PEEK (212) THEN 63987
63983 M = I:I = J * 256 + PEEK (I)
63984 IF I <> 0 THEN 63981
63985 PRINT "PROGRAM DOES NOT REACH HIGH-RES PAGE"
63986 END
63987 POKE 208, PEEK (M + 2): POKE 209, PEEK (M + 3)
63988 POKE 103, 1: POKE 104, 8: POKE 2048, 0
63989 PRINT CHR$(4);"RUN SPL.EXEC MAKER"
RUN 63980
```

***Listing 3:* SPLIT EXEC MAKER**

```
5 Y = 36
6  IF  PEEK (213) = 64 THEN Y = 72
7 Z =  PEEK (209) * 256 +  PEEK (208)
8 D$ =  CHR$ (4)
9  PRINT D$;"OPEN TEMPEXEC"
10  PRINT D$;"DELETE TEMPEXEC"
20  PRINT D$;"OPEN TEMPEXEC"
30  PRINT D$;"WRITE TEMPEXEC"
31  PRINT "POKE 103,PEEK(210)+1"
32  PRINT "POKE 104,PEEK(211)"
33  PRINT "POKE (PEEK(210)+256*PEEK(211)),0"
34  PRINT "LOAD SPLITPROG"
35  PRINT "&R ";Z + 3 + Y;",1,";Z
36  PRINT Z;" GOTO ";Z + 3 + Y
37  PRINT Z + 1;" REM SPLIT"
40  FOR I = Z + 2 TO Z + 2 + Y
50  PRINT I;" REM SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT S
     PLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPL
     IT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT SPLIT
      SPLIT SPLIT SPLIT SPLIT SPLI"
60  NEXT I
65  PRINT "EXEC SPL.PATCHER"
70  PRINT D$;"CLOSE TEMPEXEC"
80  PRINT D$;"EXEC TEMPEXEC"
```

Listing 4: SPLIT PATCHER

```
63970 I = l +  PEEK (210) + 256 *  PEEK (211)
63971 Jl =  PEEK (208):J2 =  PEEK (209)
63972  IF ( PEEK (I + 2) = Jl) AND ( PEEK (I + 3) = J2) THEN 63976
63973 I =  PEEK (I) +  PEEK (I + 1) * 256:M = I
63975  GOTO 63972
63976 K = M
63977 J =  PEEK (M + l)
63978  IF J = ( PEEK (212) +  PEEK (213)) THEN 63981
63979 M = J * 256 +  PEEK (M)
63980  GOTO 63977
63981  PRINT  CHR$ (4);"OPEN TEMPEXEC"
63982  PRINT  CHR$ (4);"DELETE TEMPEXEC"
63983  PRINT  CHR$ (4);"OPEN TEMPEXEC"
63984  PRINT  CHR$ (4);"WRITE TEMPEXEC"
63985  PRINT "1 POKE ";K;",";
63986 L =  PEEK (M):J = 2: IF L > 9 THEN J = 1: IF L > 99 THEN 63988
63987  PRINT  LEFT$ ("000",J);
63988  PRINT L;": POKE ";K + 1;",";  PEEK (M + 1)
63989  PRINT "PRINT CHR$(4);"; CHR$ (34);"DELETE SPL.SETUP"; CHR$ (34)
63990  PRINT "DEL 63970,63995"
63991  PRINT "SAVE SPLITPROG"
63992  PRINT "PRINT "; CHR$ (34);"END OF JOB"; CHR$ (34)
63993  PRINT "PRINT CHR$(4);"; CHR$ (34);"DELETE TEMPEXEC"; CHR$ (34)
63994  PRINT  CHR$ (4);"CLOSE TEMPEXEC"
63995  PRINT  CHR$ (4);"EXEC TEMPEXEC"
RUN 63970
```

SPL.PATCHER Description

The EXEC is copied to the end of the user program and then that part is executed.

Lines 63970-63980 compute the actual machine addresses before and after the split, which must be modified to relink the program.

Lines 63981-63995 create and EXEC TEMPEXEC, which does the following:

> *Lines 63985-63988* replace the initial dummy POKE with the appropriate POKE for relinking the program.

> *Lines 63989-63993* clean up some of the garbage left by the process, save the final SPLITPROG, and print the "END OF JOB" message.

As you can see, a lot goes on, and a number of EXEC files are executed. This is mainly for the purpose of making the entire process automatic. I welcome any changes or modifications that would streamline the process.

Mr. Berns has been involved in computing since 1959, working on large-scale computer systems as a systems analyst/programmer. He has written both the BASIC compiler and interpreter and the LISP system for Stanford University where he has been employed for the past 14 years. He may be contacted at 735 La Para Ave., Palo Alto, CA 94306.

MICRO

Extra Colors for the Atari

by Richard I. and Donna M. Marmon

Two techniques are presented to achieve extra colors on the Atari screen. One uses alternating adjacent dots of different colors, while the other uses alternating displays of different colors.

Extra Colors
requires:
Atari 400/800 (8K)

Atari graphics modes are limited in the number of colors that can be displayed at any one time. In Mode 7, for example, only four colors can be used. There are many techniques available that will expand your choices. In this article we describe two of them and provide sample program illustrations.

Color Dot Mixing

With this technique you place pixels of different colors next to one another in an alternating pattern. For instance, a red pixel followed by a blue pixel followed by a red pixel, and so on. In a relatively large bounded area, such as a square, the overall perceived shade is distinct from the individual colors comprising the pattern. Using this approach, the number of distinct colors in Mode 7 expands to 10: primary colors 1, 2, 3, and 4; and the "mixed" colors 1-2, 1-3, 1-4, 2-3, 2-4, and 3-4. Of course, you must make a judicious choice of the four primary colors so that the mixed colors will look good and appear distinct.

Program 1 will help in your exploration of Color Dot Mixing. It places two sets of two squares on a Mode 7 screen, one against a white background and one against a black background. The program then asks you to type in a pair of color-luminance combinations. It colors a square in each set with each color you chose, and then shows you the result of the mix in a square directly below. Why are there two sets of squares? Since colors look different

Listing 1

```
1 REM **PROGRAM 1**
2 REM
3 REM ILLUSTRATES COLOR DOT MIXING
4 REM YOU INPUT THE COLORS YOU WANT
5 REM THE PROGRAM THEN DISPLAYS THEM
6 REM AND MIXES THEM FOR YOU-
7 REM AGAINST A WHITE AND BLACK BACKGROUND
8 REM
10 GRAPHICS 7
20 SETCOLOR 0,0,14:COLOR 1
30 FOR I=80 TO 159 STEP 1
40 PLOT I,0:DRAWTO I,79:NEXT I
50 PLOT 10,10
60 DRAWTO 20,10:DRAWTO 20,20:DRAWTO 10,20:DRAWTO 10,10
70 PLOT 30,10
80 DRAWTO 40,10:DRAWTO 40,20:DRAWTO 30,20:DRAWTO 30,10
90 PLOT 20,30
100 DRAWTO 30,30:DRAWTO 30,40:DRAWTO 20,40:DRAWTO 20,30
110 COLOR 0
120 PLOT 90,10
130 DRAWTO 100,10:DRAWTO 100,20:DRAWTO 90,20:DRAWTO 90,10
140 PLOT 110,10
150 DRAWTO 120,10:DRAWTO 120,20:DRAWTO 110,20:DRAWTO 110,10
160 PLOT 100,30
170 DRAWTO 110,30:DRAWTO 110,40:DRAWTO 100,40:DRAWTO 100,30
180 COLOR 2
190 FOR I=11 TO 19 STEP 1
200 PLOT I,11:DRAWTO I,19
210 PLOT I+80,11:DRAWTO I+80,19
220 NEXT I
230 FOR I=21 TO 29 STEP 2
240 FOR J=31 TO 39 STEP 2
250 PLOT I,J:PLOT I+80,J
260 NEXT J:NEXT I
262 FOR I=22 TO 29 STEP 2
264 FOR J=32 TO 39 STEP 2
266 PLOT I,J:PLOT I+80,J
268 NEXT J:NEXT I
270 COLOR 3
280 FOR I=31 TO 39 STEP 1
290 PLOT I,11:DRAWTO I,19
300 PLOT I+80,11:DRAWTO I+80,19
310 NEXT I
320 FOR I=22 TO 29 STEP 2
330 FOR J=31 TO 39 STEP 2
340 PLOT I,J:PLOT I+80,J
350 NEXT J:NEXT I
352 FOR I=21 TO 29 STEP 2
354 FOR J=32 TO 39 STEP 2
356 PLOT I,J:PLOT I+80,J
358 NEXT J:NEXT I
360 PRINT "First color(0-15), lum(0-14 even)"
370 INPUT CF,LF
380 PRINT "Second color(0-15), lum(0-14 even)"
390 INPUT CS,LS
400 SETCOLOR 1,CF,LF
410 SETCOLOR 2,CS,LS
420 PRINT :PRINT :PRINT
430 PRINT "First: Color=";CF;" Lum=";LF
440 PRINT "Second: Color=";CS;" Lum=";LS
450 GOTO 360
```

against different background colors, we decided to experiment with different backgrounds.

You can easily modify this program to further enlarge the technique. Why not mix more than two colors? Or how about different cycles; one red pixel followed by two blue pixels, for example? Try background colors other than black and white. You'll really see the difference!

The possibilities from color dot mixing are great and they are suitable for many applications. The resulting colors, however, appear rather coarse to the eye. The next technique uses display list interrupts to create new colors that are much purer and far more pleasing to the eye.

Color Dot Alternation

To understand this technique, we must first review some Atari display theory. When you look at a display on your television or monitor, it appears almost as if the display is painted on the screen. Actually, the display is being regenerated by ANTIC, 60 times per second. But this is so fast that you don't see the resulting flicker.

An explanation of the display list and display list interrupts appears in many places, so we won't go into that in great detail: just enough to give a frame of reference. Every 60th of a second (or frame), ANTIC goes through its display list and associated display memory, retrieving the color register number for a given pixel from display memory and displaying the pixel at the correct point on the screen in the color specified in the appropriate color register. What would happen if, on alternate frames, different colors were displayed for the same pixel? During frame 1, for instance, the color of a pixel might be red. During frame 2, it might be blue. During frame 3, it would go back to red, and so on. The result, according to color wheel theory, is that the pixel should appear purple. And indeed it does. If you kept track which pixels of your display should be pure red, which should be pure blue, and which should be alternating red-blue, you would obtrain three pure and distinct colors for the price of two! In a Mode 7 display, this makes 10 colors a possibility.

This technique may be implemented in several ways. Program 2 demonstrates one way. As in program 1, program 2 asks you to type in two color-luminance combinations. These colors are displayed in separate squares, and then below them a rectangle is displayed with the colors alternating on different display frames.

Listing 2

```
1 REM **PROGRAM 2**
2 REM
3 REM ILLUSTRATES COLOR DOT ALTERNATION
4 REM YOU INPUT THE COLORS YOU WANT
5 REM THE PROGRAM THEN DISPLAYS THEM
6 REM AND MIXES THEM FOR YOU
7 REM
10 GRAPHICS 7
20 SETCOLOR 4,0,14
30 PRINT "FIRST COLOR(0-15), LUM(0-14 EVEN)"
40 INPUT CF,LF
50 PRINT "SECOND COLOR(0-15), LUM(0-14 EVEN)"
60 INPUT CS,LS
70 SETCOLOR 0,CF,LF
80 SETCOLOR 1,CS,LS
90 COLOR 1
100 FOR I=21 TO 39
110 PLOT I,21:DRAWTO I,39:NEXT I
120 COLOR 2
130 FOR I=51 TO 69
140 PLOT I,21:DRAWTO I,39:NEXT I
170 SETCOLOR 2,0,0
180 COLOR 3
190 FOR I=26 TO 54
200 PLOT I,51:DRAWTO I,69:NEXT I
210 POKE 36770,240
215 RESTORE
220 FOR I=0 TO 39
230 READ A:POKE 1536+I,A:NEXT I
240 DATA 72,138,72,169,0,141,10,212
250 DATA 141,24,208,169,20,141,0,2
260 DATA 104,170,104,64,72,138,72
270 DATA 169,0,141,10,212,141,24,208
280 DATA 169,0,141,0,2,104,170,104,64
290 POKE 1540,CF*16+LF:POKE 1560,CS*16+LS
300 POKE 512,0:POKE 513,6
310 POKE 54286,192
320 END
```

The alternation is produced by the display list interrupt routines shown in listing 3. In listing 2, the routines are POKEd into memory at lines 220-280. When the routines are executed in their appropriate frames, the color register used for the third square is flip-flopped between the two colors you chose. During the odd frames, the color register is set to the first color. During the even frames, the color register is set to the second color. Each interrupt routine causes the other one to execute during the next frame by modifying the display list interrupt vector. The input colors are set into the interrupt routines by line 290. The result on the screen is that the third square has a different color than the other two.

To fully utilize Color Dot Alternation, some additional programming is needed. You must keep track of which pixels are to be mixed and which are not. This can be accomplished by using multiple display list interrupts in conjunction with tables giving the mode lines to be mixed.

The implementation given here uses two display list interrupts for alternate frames. Another method is to use a single display list interrupt that flip-flops a color register. You would change the contents of the color register during alternate frames. A final method, although costly in memory, is to have two separate display memories. One would contain the color register numbers used during the odd frames, and the other would contain the color register numbers used during the even frames. If a pixel is to be a pure color, its associated color register number would be the same for odd and even frames. If a pixel is to be mixed, its color register number would alternate in the two display memories between the two registers to be mixed. A display list interrupt would be used to change the display list itself to point to the two

Listing 3

```
                DISPLAY LIST INTERRUPT ROUTINES FOR PROGRAM 2

PHA        ODD FRAME DLI ROUTINE
TXA
PHA
LDA #0          AT RUN TIME CONTAINS FIRST INPUT COLOR
STA WSYNC
STA COLPF2      PUTS FIRST COLOR IN COLOR REGISTER 2
LDA #20         SETS EVEN FRAME DLI ROUTINE
STA $512        TO EXECUTE DURING NEXT FRAME
PLA
TAX
PLA
RTI


PHA             EVEN FRAME DLI ROUTINE
TXA
PHA
LDA #0          AT RUN TIME CONTAINS SECOND INPUT COLOR
STA WSYNC
STA COLPF2      PUTS SECOND COLOR IN COLOR REGISTER 2
LDA #0          SETS ODD FRAME DLI ROUTINE
STA $512        TO EXECUTE DURING NEXT FRAME
PLA
TAX
PLA
RTI
```

display memory areas for different frames.

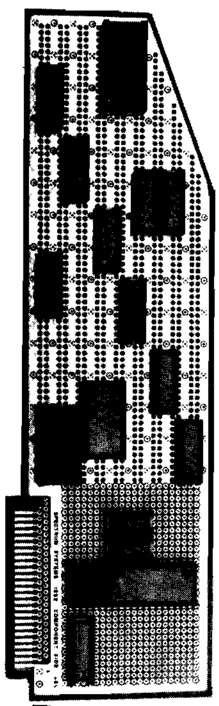
One warning about this technique. When you alternate between colors of different luminances, flickering will occur. The flickering will worsen as the luminances get farther apart and will be almost non-existent when the luminances are the same, especially with the higher color numbers. This flicker effect can be very useful for special effects in your programming.

The expansion possibilities for Color Dot Alternation seem endless. You can experiment with cycles of three or more. With a cycle of two, your color palette is 256. With higher cycle numbers, the choices are greater, of course. We have used the technique effectively with cycles of up to four.

Both Color Dot Mixing and Color Dot Alternation are simple techniques that expand the color possibilities of your Atari. By using these techniques, or variations, you can begin to realize the full graphics potential of your computer.

Contact the authors at 901 Green Forest Drive, Montgomery, AL 36109.

MICRO™



Introduction to 3-D Rotations on the Apple

by Chris Williams

The techniques of 3-D rotation are discussed. An Applesoft demonstration program is provided, which includes general-purpose routines for yaw, pitch, and roll.

ROTATE
requires:
Apple with Applesoft

I am fascinated by the computer-generated special effects recently proliferated through the film industry. The primary building block for these special effects is the 3-D rotation. I've discovered that these effects are remarkably easy to produce; this article and program pass on the techniques required.

The program is written entirely in Applesoft and isn't offensively slow, but don't expect fluid motion. The program documentation is thorough, but you should make two copies (one with REMs, one without). Executing the REMed program will try your patience.

I won't provide an in-depth discussion on the math; you don't need to understand it to do rotations. Just strip out my subroutines and use them where you need them.

The program draws a hi-res 3-D rectangular box and then rotates it. The rotation occurs in discrete steps of 15 degrees. It takes about 90 seconds to rotate through 360 degrees.

This box is a real-world object — it has height, width, and depth. The task in doing 3-D display is to project real-world objects onto a two-dimensional surface.

Figure 1 is the Apple hi-res screen for X and Y. Z is supposed to be an axis drawn out of the screen and perpendicular to it. Since that can't be done, represent Z as a line drawn at a 45-degree angle, upper left to lower right. This line allows you to put depth on the screen.

Figure 2 introduces the box and shows how the 3-D to 2-D projection works. Notice that the corners of the object are numbered points. They correspond to the CR array in the program. This array is dimensioned (3,8). The object has eight corners and each corner has three elements representing X, Y, and Z coordinates. Lines 170 through 400 of the program simply define the box in 3-D space.

Now you need to project it onto the 2-D plane. X and Y coordinates correspond precisely so they pose no problem. The Z coordinate is another matter.

Since Z is defined to be at 45 degrees to both X and Y, then the Z coordinate is related to X and Y as X = sin(45)*Z, Y = sin(45)*Z. Sine and cosine of 45 degrees are both .707. Specify a corner's position in 3-D space as CR(1,I), CR(2,I), and CR(3,I) where I selects the corner and 1,2,3 is XYZ. A corner's 2-D projection onto the X-Y screen is computed in lines 490 and 500 using X = X + .707*Z, Y = Y + .707*Z. Lines 510 and 520 are there merely for scaling and putting the box near the center of the screen.

The edges of the box are defined even more simply. They are in a table look-up contained within arrays I1 and I2. Notice that both these arrays are dimensioned as 12 as there are 12 edges on the box.

The number in each element of these arrays is a corner (see figure 2). That means an edge exists from point I1(I) to point I2(I). If you look at lines 550 through 590, you'll see HPLOT draws an edge from X(I1(I)),Y(I1(I)) to X(I2(I)),Y(I2(I)).

That's all it takes to draw a 3-D box. You can convince yourself of this by turning off the rotation in the program (insert line 455 GOTO 480). This will display the box in X,Y.

Now take out the GOTO and try rotating. Here, the problem is again one

Figure 1

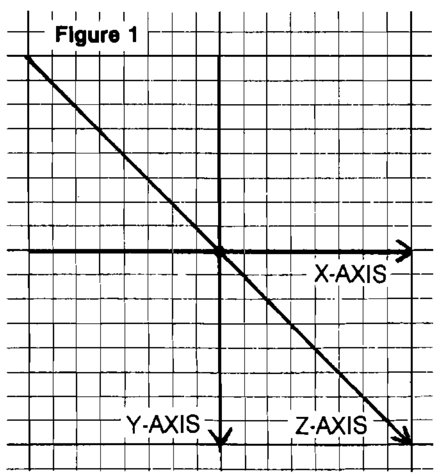
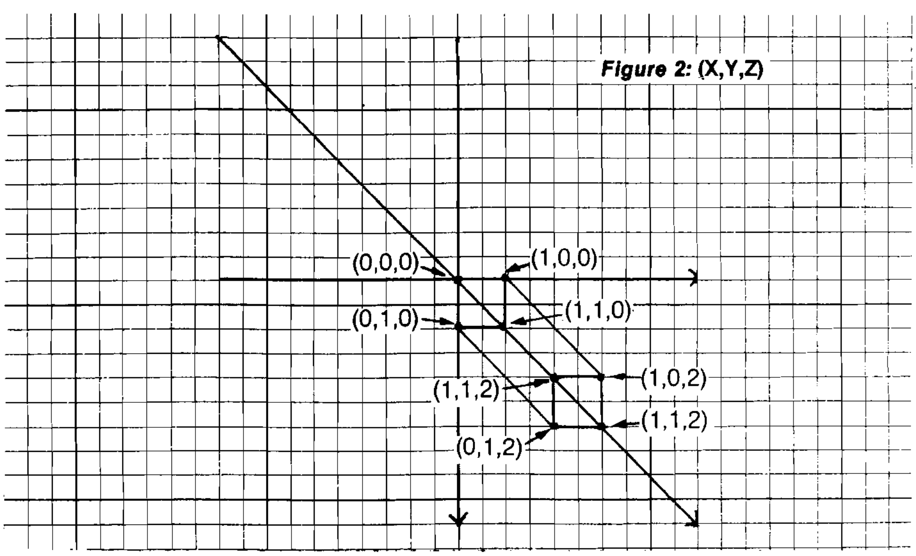


Figure 2: (X,Y,Z)



of axis definition. Refer back to figure 1. Call rotation about the Z axis — Roll, about the X axis — Pitch, and about the Y axis — Yaw. The program does a Yaw rotation only.

The trick is in the matrix multiply. If you let a given corner's coordinates be X1,Y1,Z1, then

$$\begin{pmatrix} X2 \\ Y2 \\ Z2 \end{pmatrix} = \begin{pmatrix} \text{ROTATION MATRIX} \end{pmatrix} \times \begin{pmatrix} X1 \\ Y1 \\ Z1 \end{pmatrix}$$

where X2,Y2,Z2 is that point's new, rotated coordinates.

If you don't understand that, don't worry. You'll be able to just plug numbers; you won't need to understand it.

The rotation matrix above is defined in lines 620 through 720. The angle for the sine and cosine call comes from line 450 where degrees are incremented by 15 each time through the loop and then converted to radians.

The choice of where the sine and cosine terms go in the rotation matrix determines whether the rotation is Roll, Pitch, or Yaw. Looking at the program, see that for Yaw

$$\text{DIM RT(3,3)} = \begin{pmatrix} c(\text{Yaw}) & 0.0 & s(\text{Yaw}) \\ 0.0 & 1.0 & 0.0 \\ -s(\text{Yaw}) & 0.0 & c(\text{Yaw}) \end{pmatrix}$$

How to run a listing in
MICRO's Software/Hardware

The Software and Hardware Catalogs are provided as a service both to our readers and to the manufacturers. These entries are not MICRO reviews, but descriptions provided by the manufacturer.

To run a free listing in either catalog, a company fills out the appropriate form or merely mails in their material in the same format that appears in the magazine.

We try to limit entries to one company per month, on a first-come-first-serve basis.

If you sell products our readers should know about, write to Software/Hardware Catalog, MICRO, P.O. Box 6502, Chelmsford, MA 01824.

Listing 1

```
    REM  ***************
    REM  ROTATE TUTORIAL
    REM  BY C. WILLIAMS
    REM  COPYRIGHT 1982
    REM  BY C. WILLIAMS
    REM  ***************
10  DIM CO(3),C(3)
20  DIM I1(12),I2(12)
30  DIM RT(3,3)
40  DIM CRNR(3,8)
50  DIM XP(8),YP(8)
60  DIM D(3)
65  REM  READ EDGE-START POINTS INTO I1
70  FOR I = 1 TO 12
80  READ A
90 I1(I) = A
100  NEXT
110  DATA  1,2,3,4,5,6,7,8,1,2,3,4
115  REM  READ EDGE-END POINTS INTO I2
120  FOR I = 1 TO 12
130  READ B
140 I2(I) = B
150  DATA  2,3,4,1,6,7,8,5,5,6,7,8
160  NEXT
165  REM  DEFINE 3-D POSITION OF CORNERS
170 CRNR(1,1) = 0.
180 CRNR(2,1) = 0.
190 CRNR(3,1) = 0.
200 CRNR(1,2) = 0.
210 CRNR(2,2) = 1.
220 CRNR(3,2) = 0.
230 CRNR(1,3) = 1.
240 CRNR(2,3) = 1.
250 CRNR(3,3) = 0.
260 CRNR(1,4) = 1.
270 CRNR(2,4) = 0.
280 CRNR(3,4) = 0.
290 CRNR(1,5) = 0.
300 CRNR(2,5) = 0.
310 CRNR(3,5) = 2.
320 CRNR(1,6) = 0.
330 CRNR(2,6) = 1.
340 CRNR(3,6) = 2.
350 CRNR(1,7) = 1.
360 CRNR(2,7) = 1.
370 CRNR(3,7) = 2.
380 CRNR(1,8) = 1.
390 CRNR(2,8) = 0.
400 CRNR(3,8) = 2.
405  REM  LINE 410 SETS VARIABLES FOR SPEED
410 P7 = .707:OE = 1:TW = 2:TR = 3:TT = 30:SF = 75:FV = 15:EI = 8:HU = 150
420  HCOLOR= 3
430  HOME : VTAB 5: INPUT "ENTER INITIAL ROTATION ANGLE ";AG
440 RAD = 3.14159 / 180.
445  REM  THE LOOP BEGINS HERE WITH ANGLE INCREMENT
450 AG = AG + FV:AG = AG * RAD
460  GOSUB 620: REM  RT DEFINED
470  GOSUB 740: REM  MATRIX MULTIPLY
475  REM  THIS LOOP DOES THE 3-D--->2-D PROJECTION, PLUS SCREEN SCALING
480  FOR I = OE TO EI
490 XP(I) = CRNR(OE,I) + P7 * CRNR(TR,I)
500 YP(I) = CRNR(TW,I) + P7 * CRNR(TR,I)
510 XP(I) = (XP(I) * TT) + HU
520 YP(I) = (YP(I) * TT) + SF
530  NEXT
535  REM  ERASE OLD BOX, FULL SCREEN
540  HGR : POKE - 16302,0
545  REM  DRAW IT, PLUS THE EDGES
550  FOR I = 1 TO 12
560 P1 = I1(I)
570 P2 = I2(I)
580  HPLOT XP(P1),YP(P1) TO XP(P2),YP(P2)
590  NEXT
600  GOTO 450
610  END
615  REM  SUBROUTINE AT 620 DEFINES RT
620 CANGLE =  COS (AG)
630 SANGLE =  SIN (AG)
640  FOR I = 1 TO 3
650  FOR J = 1 TO 3
660 RT(I,J) = 0.
670  NEXT J: NEXT I
680 RT(1,1) = CANGLE
690 RT(2,2) = 1.
700 RT(3,3) = CANGLE
710 RT(1,3) = SANGLE
```

(contin

Listing 1 *(continued)*

```
720 RT(3,1) = ( - SANGLE)
730  RETURN
735  REM  SUBROUTINE AT 740 DOES X,Y,Z GET
740  FOR I = OE TO EI
750  FOR J = OE TO TR
760 C(J) = CRNR(J,I)
770  NEXT J
780  GOSUB 840: REM  MM3
790 CRNR(OE,I) = CO(OE)
800 CRNR(TW,I) = CO(TW)
810 CRNR(TR,I) = CO(TR)
820  NEXT : RETURN
830  END
835  REM  LINE 840 STARTS THE MATRIX MULT.
840  FOR K = 1 TO 3
850 D(K) = 0.
860  NEXT K
870  FOR II = OE TO TR
880  FOR J = OE TO TR
890 D(II) = D(II) + RT(II,J) * C(J): NEXT J
900  NEXT II
910  FOR II = OE TO TR
920 CO(II) = D(II)
930  NEXT II
940  RETURN
```

If you'd like to do Pitch rotations, rearrange the array to look like this:

$$\text{DIM RT(3,3)} = \begin{pmatrix} 1. & 0.0 & 0.0 \\ 0.0 & c(Pitch) & -s(Pitch) \\ 0.0 & s(Pitch) & c(Pitch) \end{pmatrix}$$

And for Roll rotations, like this:

$$\text{DIM RT(3,3)} = \begin{pmatrix} c(Roll) & s(Roll) & 0.0 \\ -s(Roll) & c(Roll) & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

If you don't know matrix algebra, it doesn't matter. Just rearrange lines 680 through 720 to produce arrays as shown above and you can rotate any way you wish.

That's all there is to 3-D rotation. I cringe as I say that because there are all sorts of things to be reckoned with. Refraction, shading, and hidden lines and objects are not topics to be discussed in an article with a title like this one. But the program here is a good foundation.

There is room for improvement, of course. The more adventuresome readers might try defining other 3-D shapes and then rotating them. The subroutines are all there. Or you might work for speed; a more fluid update would certainly be a plus. Let me know what you come up with.

Mr. Williams is an electrical engineer/writer. He may be contacted at 5676 S. Meadow Lane #101, Ogden, UT 84403.

MICRO

From Here to Atari

By Paul Swanson

We are pleased to introduce our new Atari Column. Paul Swanson has published articles in several microcomputer magazines and has authored a book on disk techniques. He runs his own software consulting firm and markets a full-size keyboard for the Atari 400.

The Atari has two microprocessors instead of one, unlike what you would find in most systems. The main processor is the 6502. Atari, Inc., custom-made the second processor, ANTIC, which is used to control just the graphics screen. A few other custom chips in the Atari are not full microprocessors, but you can control them through the hardware registers.

Taking advantage of all the Atari's special hardware can be quite complicated. Fortunately, because of the various graphics modes available, much of this hardware can be accessed through BASIC without knowing very much about the hardware itself.

You can write simple programs using the information in the BASIC Reference Manual supplied with the BASIC cartridge. More complex programs that take advantage of Atari's special chips require more information. I will supply you with details of this special hardware in upcoming columns.

Each month's column will focus on some feature of Atari's hardware. I intend to take these topics from letters I receive, so if you have questions or need more information on a specific function, send me a letter at 97 Jackson St., Cambridge, MA 02140. (If you don't want your name mentioned, be sure to note that in your letter. I will include excerpts from letters in my columns.)

In addition to answering questions concerning the Atari, this column will also contain information on new hardware and software from Atari and other manufacturers. Already available from second sources are memory boards for both the 800 and the 400. The 32K board (for both) has been available from several different companies for over a year now. There is also 48K on a single card that you can plug into your 400. I've heard of a 64K board but haven't seen it yet. The Atari can't address 64K directly because of the 16K reserved for the operating system ROM and the hardware registers. The 64K board is bank-selectable so it doesn't exceed the 48K allowed. Available pseudo disks give you 128K of memory that you can access as if it were a disk.

Many new products have been announced for the Atari, and many are about to be announced. Much of the new software available is in response to Atari Inc.'s interest in home educational applications. *My First Alphabet*, by Fernando Herrera, is still one of the best educational software packages available for younger children. A new program called *Master Type*, from Lightning Software, makes an interesting spaceship game out of learning touch typing. If you want to learn how to type, you may want to get your local computer store to demonstrate this program for you. It is listed at $39.95, requires 32K and a disk drive, and keeps your interest with 17 levels of complexity. These two software packages, as well as the memory boards, are available at most computer stores that carry Atari.

This column, on occasion, will try to clarify conflicting rumors. For example, I have heard three or four people asking about the "special vector" you can use in the Atari to get rid of the key click. I can appreciate the need for eliminating that noise — it seems much louder when everyone else is asleep — but I disassembled part of the operating system looking for the keyboard click and found no place to POKE anything that would eliminate it. The part in question is in ROM and you can't alter ROM with a POKE. It doesn't check with any RAM locations before the JSR (Jump to SubRoutine) that produces the click. The only ways I can see to eliminate the click include physically disconnecting the keyboard speaker or writing your own keyboard handler. Even writing the keyboard handler would not eliminate the click under every possible condition; you have to initiate the handler every time one of a variety of different things happens.

Future columns may also include a listing of a short utility program (BASIC) that is in the public domain. No machine-readable copies will be available. If you really don't want to type it in, check with your local user group to see if anyone already typed it in.

My December column will feature the Atari Regional Software Acquisition Centers; January will include available technical literature. December's column will be of particular interest if you plan to market any of your Atari software. January's topic covers places where you can find lists and explanations of all those special memory locations that you need to develop fancier software.

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

78. Color Computer News, Issue No. 10 (July, 1982)

Jackson, Jesse W., "TTYPELOG," pg. 58-60.
A utility for the Color Computer to log tape information to your printer.

Rouse, Alan, "Mortgage," pg. 62-64.
A "What If" program for the Color Computer.

Eichman, Steve, "Shaplist," pg. 65-67.
An Inventory/File program for the 6809-based Radio Shack Color Computer.

Aker, Jack L., "BASIC Program Line Mover," pg. 69-70.
A utility for a Color Computer with Extended BASIC, and 16K or 32K memory.

Becus, Georges A., "TRACETXT Listing Produced by PRINTXT CTRACE," pg. 71-75.
A trace utility for the 6809-based Color Computer.

Pretty, Richard, "A Pull on Your 'Art' Strings," pg. 77.
A simple string art program for the TRS-80 Color Computer.

Barnes, Mark, "Gold Mine," pg. 78-81.
A graphics maze game for the 6809-based Color Computer.

Wright, Darrel, "Convert That File," pg. 83-84.
A utility program to convert a Telewriter text file to a standard ASCII data file or *vice versa*. For the TRS-80 Color Computer.

Peterson, Russell M., II, "Exploring Graphics Modes," pg. 87-88.
Using the SDS80C Editor/Assembler on the TRS-80 Color Computer.

79. 80-U.S. Journal, 5, No. 7 (July, 1982)

Roberts, R.N., "Togetherness," pg. 88-89.
Tape merge for the 6809-based TRS-80 Color Computer.

Causer, R. Shane, "17K of RAM," pg. 94-95.
Squeeze extra space from memory on your TRS-80 Color Computer. This is done by saving 1K of memory out of a program requiring 16K of memory.

80. The Rainbow, 2, No. 1 (July, 1982)

Parkman, Bob, "Silent Answer," pg. 4.
A driver to interface a TI Silent 700 with the 6809-based TRS-80 Color Computer.

Mir, Jorge, "Let's Go On A Simple Rainbow Adventure," pg. 9-17.
An adventure-type program for the Color Computer.

Blyn, Steve, "Design Programs to Help Children Learn," pg. 18-19.
BEEPEROO is a simple program for the Color Computer which can be used to reinforce the concept of simple addition of three-digit numbers.

Penrose, Paul, "Playing Around With Your 80C," pg. 22.
Playing music with the TRS-80 Color Computer.

Morgan, Alan J., "Synchronizing with Your SAM Chip," pg. 24-25.
Using the 6883 SAM chip with the 6809-based Color Computer.

Nolan, Bill, "Make Magic Rings on Magic 80C," pg. 27-29.
For addicts of fantasy role-playing programs, this Color Computer listing rolls up magic rings for you.

Rutledge, E.P., "Disk File Helps You Keep Track of Everything," pg. 30-33.
A utility for the 6809-based Color Computer.

Lewandowski, Dennis S., "Let's Soak Up Some Assembly," pg. 38-39.
A second tutorial on 6809 assembly-language programming.

Ebbert, Jim, "Is There Any Escape from No Escape?", pg. 42-45.
A space-navigation graphics program for the Radio Shack TRS-80 Color Computer.

Waclo, John, "The NFL Report Can Choose This Fall's Winners," pg. 50-64.
Put the National Football League on your Color Computer.

81. Compute! 4, No. 1 (January, 1982)

Mansfield, Richard, "BRANCH NEVER and QUIF Assembling on SuperPET," pg. 146-149.
An instructional article for SuperPET users.

82. Microcomputing, 6, No. 8 (August, 1982)

Whitman, James A., "Pascal and BASIC Square Off, Continued," pg. 22.
Benchmark tests including data on the 6809-based TRS-80 Color Computer.

83. MICRO, No. 51 (August, 1982)

Steiner, John, "Interfacing the Color Computer," pg. 33-36.
Circuits to interface the TRS-80 Color Computer to an RS-232 port and a motor control relay are presented. A Morse Code send/receive program is included as a demonstration.

Staff, "MICRO Reviews in Brief," pg. 39-41.
Reviews include discussion of a disassembler for the 6809 and the Compuvoice Synthesizer for the 6809-based Color Computer.

Dial, Wm. R., "6809 Bibliography," pg. 118.
Some 26 items relating to 6809 literature are listed.

84. 80 Micro, No. 52 (September, 1982)

Miller, Franklyn D., "The Colorful Computer — Part II," pg. 152-162.
Twenty programs for the 6809-based TRS-80 Color Computer.

Tucker, Richard, "Cheaper Upgrade," pg. 186-188.
Do it yourself and save substantial cost in converting a 4K Color Computer, 8K Color BASIC to a 16K Extended Color BASIC by installing the new ROM yourself.

Norman, Scott L., "Pascal Goes Color," pg. 198-202.
Compiled Pascal for the Color Computer is discussed.

Sprouse, Gerald, "Joystick Paintbrush," pg. 230-232.
Use the Color Computer like a drawing board, employing two programs listed.

Osborne, Frank H., "Conversion," pg. 238-240.
Rewrite Level II BASIC programs to run on the 6809-based Color Computer.

Heusinkveld, John, "PCLEAR 0," pg. 282.
Make hi-res graphics use high memory on the 6809-based Color Computer.

85. Personal Computer, 5, No. 8 (August, 1982)

Curtis, Mike and Whelan, Joe, "The Dragon," pg. 112-116.
Dragon 32 is a new 6809-based British microcomputer with full color, 32K RAM, 16K Microsoft BASIC, etc.

86. Color Computer News, Issue No. 11 (August, 1982)

Morrow, Ken, "Program Relocation," pg. 19-21.
Discussion and listing for program relocation on the 6809-based TRS-80 Color Computer.

Bassen, Howard, "Optimizing High-Resolution Animated Games in Extended BASIC — Part 2," pg. 23-26.
Tutorial on the use of game routines with several demo listings.

Lester, Lane P., "Motion Picture Programming and the Teacher," pg. 27-28.
Animation on the Color Computer.

MICRO™

MICRO™

Software Catalog

Name: **Adventure to Atlantis**
System: Apple II
Memory: 48K
Language: Machine Code
Hardware: One disk drive
Description: This game combines the best features of adventure games, arcade games, and fantasy role-playing games. *Adventure to Atlantis* is a sequel to Odyssey: The Complete Adventure. The struggle between the forces of magic (The High One) *versus* the forces of science (The Atlanteans) continues. The game uses four methods to grab the player's attention: high-resolution color graphics and animation, sound effects to enhance the action, random events at all stages of the adventure, and embedded arcade-like action.
Price: $40.00
Includes one floppy disk.
Author: Robert C. Clardy
Available:
Synergistic Software
830 N. Riverside Drive
Suite 201
Renton, WA 98055

Name: **The Business Bookkeeping System™**
System: Apple II, Apple II Plus, Apple III
Memory: 48K
Language: Applesoft BASIC
Hardware: Two disk drives, video monitor, printer, emulation mode (Apple III)
Description: *The Business Bookkeeping System* is a cash accounting system that includes general ledger, customer activity, vendor activity, and employee activity. It contains eleven general ledger reports, five customer activity reports, four vendor activity reports, and six employeee activity reports.
Price: $395.00
Includes indexed documentation, ten diskettes, toll-free software support.
Available:
Dakin 5 — Authorized Apple dealer

Name: **Astro Blast**
System: TRS-80 Color Computer
Memory: 16K/cassette 32K/disk
Language: Assembly
Hardware: Joysticks
Description: *Astro Blast* is a space "shoot-em-up" featuring the highest resolution graphics, lots of color, and dramatic sound effects. Wave after wave of alien attackers are a challenge to your joystick and fire-button skills. Three selectable skiil levels coupled with automatic game acceleration challenge both novice and professional.
Price: $24.95/cassette
$29.95/disk
Includes full instructions.
Author: Ron Krebs
Available:
Mark Data Products
23802 Barquilla
Mission Viejo, CA 92691

Name: **INTROL-C/6809 Compiler System**
System: FLEX, UniFLEX, OS-9, CP/M
Memory: 56K
Language: C
Description: Powerful C language compiler develops programs in C for systems running under FLEX, UniFLEX, or OS-9; also as cross-software for CP/M-based development systems. Full-blown implementation provides full support of all standard C except initializers and bitfields. Written entirely in C itself, Introl-C includes a C compiler, 6809 assembler, linking loader, standard runtime library and library manager, and unsurpassed code efficiency.
Price: From $400.00
Includes diskette, user manual, one-year maintenance/update.
Author: Richard D. Pennington
Available:
Introl Corp.
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

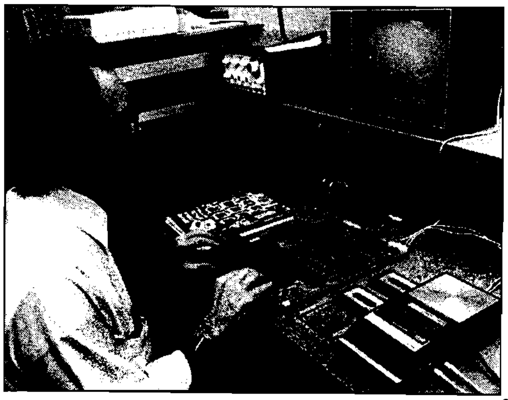
Name: **Frazzle**
System: Apple II or Apple II Plus
Memory: 48K
Language: 6502 Assembly
Hardware: Disk Drive (Dual DOS 3.2/3.3)
Description: Muse announces the release of *Frazzle*, the computer game that puts you in the future. The scene begins in space, with you commanding a Frazzle Force Ship that is suddenly under attack by Beasties. Colorful screen graphics show your position on the ship's radar screen, the force field surrounding you, and odd-shaped Beasties zooming in. Your ammunition is pulsating Energy Probes, which beep and flash as you release them on the screen. You must stop the Beasties while avoiding collisions with them, with the walls of the force field, and with your own ammunition. Sound effects include the electronic hum of a radar monitor and the squish of Energy Probes dissolving the Beasties.
Price: $24.95
Author: J.C. Nolan
Available:
Direct from Muse and computer stores nationwide

Name: **K-Razy Kritters**
System: Atari 400/800
Memory: 8K
Language: Machine Code
Hardware: ROM Cartridge
Description: This challenging celestial adventure with ten levels of play begins with three command ships. The player's active command ship attempts to destroy free-falling alien patrol "Kritters" descending from above. Weapons include standard missiles and super-missiles. If a command ship is destroyed, a sanitation crew will remove the wreckage.
Price: $39.95 suggested retail
Includes ROM cartridge and instruction booklet.
Author: Torre Meeder
Available:
K-Byte
1705 Austin
Troy, MI 48084
or your local computer software dealer

Name: **3D Drawing Board**
System: TRS-80 Color Computer
Memory: 16K
Language: BASIC
Description: *3D Drawing Board* is a tool for education, entertainment, or serious projects. It helps you draw objects in three dimensions, rotates them, and changes elevation, size, and distance. The drawings can be saved to tape or disk for future use.
Price: $24.95/cassette
$29.95/disk
Includes complete instruction manual and program with samples.
Author: Mark Laessig
Available:
Computerware
Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **Computer-Aided Instruction for General Chemistry**
System: Apple, Commodore PET, TRS-80 III
Memory: 48K/Apple II
Language: Applesoft/Apple II
Hardware: Apple — DOS 3.3
PET — 4040 or 2031 disk drive
Description: A comprehensive set of 20 programs designed to supplement a course in general chemistry. Each program contains 50-70 drills, exercises, and problems (approximately 24½ hours of machine time) presented in an interactive format. The questions are randomly generated. No computer experience is necessary. Intended for introductory college level or advanced placement in high school.
Price: $325.00
Includes four disks for the Apple II, two disks for the PET, and four or five disks for the TRS-80, and complete documentation.
Authors: William Butler
Raymond Hough
Available:
John Wiley & Sons, Inc.
Eastern Distribution Center
Order Processing Dept.
1 Wiley Drive
Somerset, NJ 08873

MICRO



Super Sale!

40% Off On Ohio Scientific Superboard II A Complete Computer System On A Board

Includes full-size 53-key keyboard, video and audio cassette interfaces; SWAP, Modem, sampler cassettes; manual; 8K BASIC-in-ROM, with 8K RAM. Requires 5-V/3 amp regulated DC power supply. 30-day limited warranty. Supply is limited. ONLY $200.00

Plus Sensational Limited-Time Savings On Ohio Scientific C1P Series personal computers, Superboard and C1P accessories, spare replacement parts, printers, monitors, integrated circuits, and other computer-related components.

To Order

Call us directly or return order coupon with your check, money order, or Mastercard or Visa Account Number. Orders will normally be shipped within 48 hours after receipt. $100.00 minimum order.

FREE
Sampler Cassettes with each Superboard II and C1P series order!

Taxi (Game), Electronic Equations, Loan Finance, Straight and Constant Depreciation, Uneven Cash Flows

Tiger Tank, Flip Flop, (Logic Game), Hectic, Black Jack, Master Mind

Cleveland Consumer Computers & Components
1333 S. Chillicothe Road, Aurora, OH 44202
TO ORDER: CALL 1-800-321-5805 TOLL FREE
(Ohio Residents Call 216-562-4136)

☐ SUPERBOARD II, $200.00

☐ Send Detailed Catalog/Order Form

Name ____________________

Address ____________________

City ____________ State ________ Zip ________

Payment by enclosed check or money order or charge to:

☐ MasterCard ☐ VISA

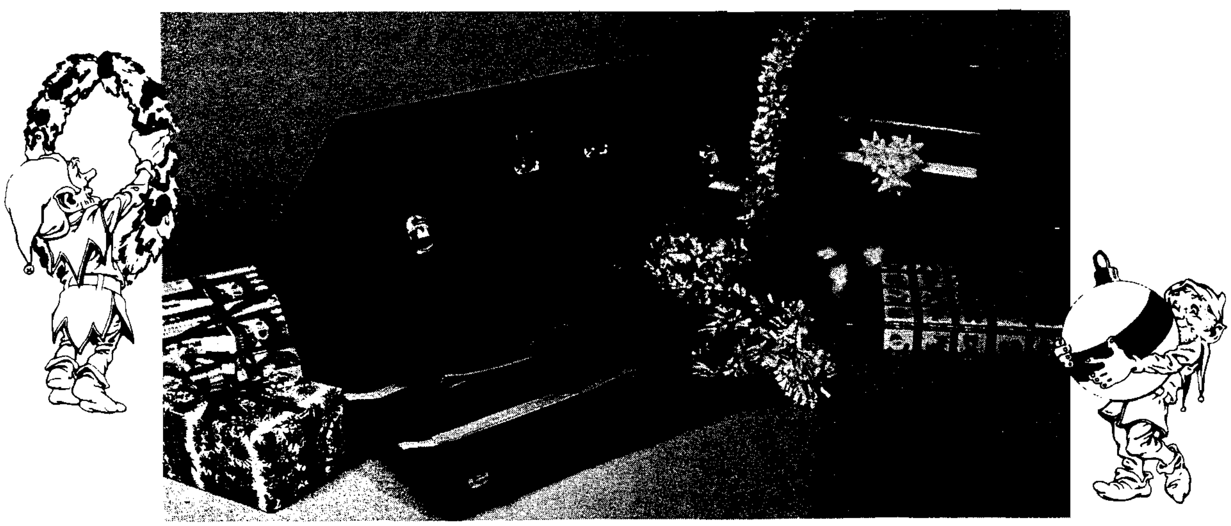
Account # ____________ Expiration Date ____________

Total Amount Charged or Enclosed $________

Ohio Residents Add 5.5% Sales Tax. All Orders Will Be Shipped Insured By UPS Unless Requested Otherwise.

comp case

IS THE PERFECT GIFT FOR THE COMPUTER THAT HAS EVERYTHING



Your investment in a computer is long term. You expect many years of reliable service. Comp Case protects your investment when not in use or while transporting it from office or home. Your computer never leaves the case — you simply remove the lid, make the electrical connection, and operate. Comp Cases are available for most brands of computers and peripherals.

Comp Case is the perfect Christmas gift. Comp Cases are available in better computer stores in our area. Or call 800-848-7548 direct to receive your Comp Case in time for Christmas.

master charge VISA

computer case company

COMPUTER CASE COMPANY • 5650 INDIAN MOUND COURT • COLUMBUS, OHIO 43213

MICRO™

Hardware Catalog

Name: **Joystick**
System: Apple II Plus
Description: This joystick is a joy to use. Its heavier metal case doesn't slip or slide like plastic. And the very sensitive switch is guaranteed for 1,000,000 pushes. Better control and reliability.
Price: $59.95
Available:
Datamost, Inc.
9748 Cozycroft Avenue
Chatsworth, CA 91311
(213) 709-1202
or computer stores

Name: **Apple-Mate**
Hardware: Floppy disk drive
Description: Add this floppy disk drive to your Apple computer. It is 100% compatible with the Apple Disk II drive, and with all Apple software including half-track-protected software.
Price: $335.00
Includes cable.
Available:
Quentin Research, Inc.
19355 Business Center Dr.
Northridge, CA 91324
(213) 701-1006

Name: **Freedom 100**
Description: This is an ASCII CRT terminal. It has a block mode, ten function keys, 15 graphics characters with full attributes, one-page screen memory, and separate attributes buffer.
Price: $595
Available:
Liberty Electronics, USA
100 Clement Street
San Francisco, CA 94118

Name: **Gimix Intelligent Serial I/O Processor Board**
System: Gimix 6809
Memory: 128K minimum
Hardware: S-30 I/O bus board
Description: This board reduces the number of interrupts between user terminals and the host CPU by buffering data transfers between system and users and preprocessing of the data. It has on-board CPU and RAM/EPROM memory, three RS-232C serial ports, a Z8038 F10 I/O interface, and supports up to three users. It requires on-board software and OS drivers.
Price: $438.11
Includes 4K RAM. (Software is optional.)
Available:
Gimix Inc.
1337 W. 37th Pl.
Chicago, IL 60609
(312) 927-5510

Name: **The Spectrum Stick**
System: TRS-80 Color Computer
Memory: 4K and up
Hardware: Joystick
Description: This new joystick has a hair-trigger firebutton and swivel-ball type component stick. The extra-long cable makes it easier to put your joystick where you want it. Red LED indicator reminds you to shut off the Color Computer.
Price: $39.95 plus $2.00 S/H
Includes 10-foot cable, red LED indicator, joystick, firebutton, case, and joystick control
Available:
Spectrum Projects
93-15 86 Drive
Woodhaven, NY 11421
(212) 441-2807 Voice
(212) 441-3755 Computer

Name: **Disk-O-Tier**
System: All disk-based systems 5¼" or 8"
Description: A convenient desktop holder for diskettes that prevents damage by laying them flat, but allows full visibility of all diskettes. It is molded of durable NAS smoked plastic, and holds eleven diskettes.
Price: $9.50 plus $2.00 postage
$19.00 for twin-pack, ppd.
Available:
ETS Center
Dept. 97
Box 651
35026-A Turtle Trail
Willoughby, OH 44094
(216) 946-8479

Name: **Pro-Guard 8" Floppy Controller**
System: Apple III
Memory: 2.2 megabytes
Hardware: 8" Shugart-compatible drives
Description: *Pro-Guard 8" Floppy Controller* adds up to 2.2 megabytes of removable media and provides backup for Apple profile. IBM 3740 format allows 8" disks to be read on other computers, including IBM mainframes.
Price: $695
Includes DOS, SOS, Pascal, CP/M distribution software, cables, manual.
Available:
Apple Dealers
MICRO-D
SUA, Inc.

Name: **Programmable Sound Module**
System: TRS-80 Color Computer
Memory: 4K and up
Language: BASIC
Description: *The Programmable Sound Module* is a plug-in cartridge for the Color Computer. A separate audio-microprocessor and ROM inside the cartridge combine to extend BASIC's vocabulary with a versatile sound-effects system. Complex noises can be created with short BASIC phrases and maintained independently of your program, allowing simultaneous video and audio effects.
Price: $139.95
Includes PSM cartridge, operating system in ROM, and full instructions.
Available:
Maple Leaf Systems
Box 2190
Station "C"
Downsview, Ontario
Canada M2N 2S9

Name: **Voter 30**
System: Apple II
Language: BASIC
Description: *Voter 30* is a peripheral hardware/software package for training, marketing, and educational uses using a group response system for up to 30 participants. Each participant gets a hand-held keypad to respond to multiple-choice questions. *Voter 30* tabulates the responses and produces a color bar chart showing the breakdown, while keeping a record of the individual station responses.
Price: $595 for interface card with programs and manual.
$125.00 each for polling stations with cable and connectors.
Available:
Reactive Systems, Inc.
40 North Van Brunt Street
Englewood, NJ 07631
(201) 568-0446

Name: **Mini-Video #82-140**
System: 6502-based video board
Memory: 4K RAM/ 4K EPROM
Language: Video Display; Monitor & Tom Pittman's Tiny BASIC
Hardware: Assembled circuit board
Description: Add a video display to your AIM or other computer. It will run Tom Pittman's Tiny BASIC with the addition of the parallel keyboard, 5V power supply and video monitor. The 2716-character generator will produce 256 8×8 characters ASCII upper- and lower-case and graphic characters. The 44-pin expansion connector can be used to add up to 6K of memory or extra I/O ports. The cursor is flashing underline type. Power requirements 5 volts, 600 MA, 3 watts.
Price: $149.95
Includes documentation and assembled board without EPROMs.
Available:
John Bell Engineering, Inc.
1014 Center Street
San Carlos, CA 94070
(415) 592-8411

MICRO

ATARI 400/800

Atari 400 and 800 are color-and-sound computers. 6502 is the main processor and ANTIC handles video. Atari 400 has a membrane keyboard and Atari 800 has a full-size typewriter keyboard.

Peripherals may include up to four disk-drive units, a cassette unit, printer, and the 850 interface module. Four programmable controller ports handle joysticks, paddles, light pens, and other accessories.

Sophisticated graphic capabilities include: 256 colors (16 may be displayed on the screen at once), 17 graphic modes (6 character and 11 map), high-resolution graphics (up to 320 × 192), and powerful player/missile graphics.

Some Useful Memory Locations on the Atari

Page 0 Locations:

Hex	Dec	Length	Name	Description
0010	016	1	POKMSK	IRQ mask
0012	018	3	RTCLOK	Real time clock
				018 is hi order
0041	065	1	SOUNDR	Noisy I/O flag
0042	066	1	CRITIC	Critical I/O flag
004D	077	1	ATRACT	Attract mode flag
0052	082	1	LMARGIN	Left margin
0053	083	1	RMARGIN	Right margin
0080	128	2 *	LOMEM	Buffer used to tokenize line of BASIC
0082	130	2 *	VNTP	Variable name table start
0084	132	2 *	VNTD	Variable name table dummy end
0086	134	2 *	VVTP	Variable value table start
0088	136	2 *	STMTAB	Statement table
008A	138	2 *	STMCUR	Current statement pointer
008C	140	2 *	STARP	String and array area
008E	142	2 *	RUNSTK	BASIC's software stack
0090	144	2 *	MEMTOP	Top of memory used by BASIC program
00BA	186	2	STOPLN	Line number of most recent stop or error
00C3	195	1	ERRSAV	Error number causing TRAP branch
00CB	203	7		Available for user machine-language programs
00D0	212	2		Value return from machine language to BASIC

* Indicates location is a vector

ANTIC Commands (Display List)

Blank line commands:
Command is (# lines + 1)*16
Ex.: Blank 8 lines is (8 + 1)*16, or $70 = dec. 144

Jump instruction: $01 = dec. 1 (2-byte instruction)

Jump on vertical blank = $41 = dec. 65 (2-byte instruction)

Display line instruction is equal to the IR mode number ($02-$0F)

To set special features, add these values to the instructions:

Add Hex	Dec.	For
10	16	Horizontal scroll enable
20	32	Vertical scroll enable
40	64	Load memory scan (makes it a 2-byte instruction)
80	128	Display list interrupt enable

Note: Display list interrupt can be enabled on any command, but scrolling and load memory scan can be enabled only on display commands.

ANTIC Commands (Display Modes)

Mode IR	Mode BASIC	Horiz. Pixels	Scan Lines Per Mode Line	Colors	Char/Map Mode
02	0	40	8	1.5	C
03		40	10	1.5	C
04		40	8	5	C
05		40	16	5	C
06	1	20	8	5	C
07	2	20	10	5	C
08	3	40	8	4	M
09	4	80	4	2	M
0A	5	80	4	4	M
0B	6	160	2	2	M
0C		160	1	2	M
0D	7	160	2	4	M
0E		160	1	4	M
0F	8	320	1	1.5	M

GTIA:

The GTIA modes are versions of mode 8, instituted by adding 64, 128, or 192 to PRIOR.

GTIA Mode	Description
9	16 luminances of background hue
10	9 colors - uses $D012 through $D01A
11	16 hues of background luminance

Compiled by Paul Swanson

ATARI 400/800
MICRO Data Sheet #10

ATARI 400/800

COLOR VALUES:

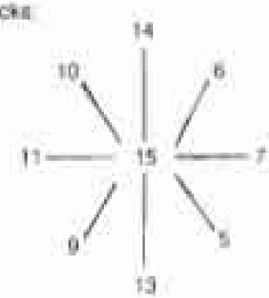
0 Black-gray-white
1 Burnt Orange
2 Orange
3 Red-orange
4 Red
5 Purple
6 Blue-purple
7 Blue
8 Blue
9 Gray-blue
10 Turquoise
11 Aqua
12 Green
13 Yellow-green
14 Orange-green
15 Orange

Default colors (SETCOLOR values):
0,2,8 (Orange)
1,12,10 (Light green)
2,9,4 (Dark blue)
3,4,8 (red)
4,0,0 (Black)

For COLORS:
POKE hue•16 + luminance into COLx

For SOUND:
POKE frequency into AUDFx
POKE distortion•16 + volume into AUDCx

Joysticks



Hardware Registers and Shadows

Address Hex	Address Dec	Shadow Hex	Shadow Dec	Name	Description	(R)ead or (W)rite
COLORS:						
D012	53266	02C0	704	COLPM0	Player-missile 0	W
D013	53267	02C1	705	COLPM1	Player-missile 1	W
D014	53268	02C2	706	COLPM2	Player-missile 2	W
D015	53269	02C3	707	COLPM3	Player-missile 3	W
D016	53270	02C4	708	COLPF0	Playfield 0	W
D017	53271	02C5	709	COLPF1	Playfield 1	W
D018	53272	02C6	710	COLPF2	Playfield 2	W
D019	53273	02C7	711	COLPF3	Playfield 3	W
D01A	53274	02C8	712	COLBK	Playfield background	W
SOUND:						
D200	53760			AUDF1	Channel 1 Frequency	W
D201	53761			AUDC1	Channel 1 Control	W
D202	53762			AUDF2	Channel 2 Frequency	W
D203	53763			AUDC2	Channel 2 Control	W
D204	53764			AUDF3	Channel 3 Frequency	W
D205	53765			AUDC3	Channel 3 Control	W
D206	53766			AUDF4	Channel 4 Frequency	W
D207	53767			AUDC4	Channel 4 Control	W
D208	53768			AUDCTL	Audio Control	W
PLAYERS AND MISSILES:						
D000	53248			HPOSP0	Horiz. pos. Player 0	W
D001	53249			HPOSP1	Horiz. pos. Player 1	W
D002	53250			HPOSP2	Horiz. pos. Player 2	W
D003	53251			HPOSP3	Horiz. pos. Player 3	W
D004	53252			HPOSM0	Horiz. pos. Missile 0	W
D005	53253			HPOSM1	Horiz. pos. Missile 1	W
D006	53254			HPOSM2	Horiz. pos. Missile 2	W
D007	53255			HPOSM3	Horiz. pos. Missile 3	W
D008	53256			SIZEP0	Size of Player 0	W
D009	53257			SIZEP1	Size of Player 1	W
D00A	53258			SIZEP2	Size of Player 2	W
D00B	53259			SIZEP3	Size of Player 3	W
D00C	53260			SIZEM	Size of all Missiles	W
D407	54279			PMBASE	Player-missile base	W
CHARACTERS:						
D401	54273	02F3	755	CHACTL	Character control	W
D404	54276			HSCROL	Horizontal scroll	W
D405	54277			VSCROL	Vertical scroll	W
D409	54281	02F4	756	CHBAS	Character set base	W
CONTROL REGISTERS:						
D01B	53275	026F	623	PRIOR	Priority Select	W
D01D	53277			GRACTL	Graphics control	W
D20E	53774	0010	016	IRQEN	IRQ enable	W
D402	54274	0230	559	DLISTL	Display list ptr., low	W
D403	54275	0231	560	DLISTH	Display list ptr., high	W
D40E	54286			NMIEN	Non-maskable int. enab.	W

Address Hex	Address Dec	Shadow Hex	Shadow Dec	Name	Description	(R)ead or (W)rite
GAME CONTROLLER PORTS:						
D010	53264	0284	644	TRIG0	Joystick trigger 0	R
D011	53265	0285	645	TRIG1	Joystick trigger 1	R
D012	53266	0286	646	TRIG2	Joystick trigger 2	R
D013	53267	0287	647	TRIG3	Joystick trigger 3	R
D200	53760	0270	624	POT0	Paddle 0	R
D201	53761	0271	625	POT1	Paddle 1	R
D202	53762	0272	626	POT2	Paddle 2	R
D203	53763	0273	627	POT3	Paddle 3	R
D204	53764	0274	628	POT4	Paddle 4	R
D205	53765	0275	629	POT5	Paddle 5	R
D206	53766	0276	630	POT6	Paddle 6	R
D207	53767	0277	631	POT7	Paddle 7	R
D208	53768			ALLPOT	Pot port state	R
D20B	53771			POTGO	Start pot scan	W
D300	54016	0278	632	PORTA	STICK0	R
		0279	633	PORTA	STICK1	R
D300	54016			PORTA	Direction register	W
D301	54017	027A	634	PORTB	STICK2	R
		027B	635	PORTB	STICK3	R
D301	54017			PORTB	Direction register	W
D302	54018			PACTL	Port A control	W
D303	54019			PBCTL	Port B control	W
MISCELLANEOUS:						
D01F	53279			CONSOL	Consol switches	R
D01F	53279			CONSOL	Keyboard speaker	W
D20A	53770			RANDOM	random number gen.	R
D20F	53775	0232	562	SKCTL	Serial port control	W
D40B	54283			VCOUNT	Vertical line counter	R
D40C	54284	0234	564	PENH	Light pen horiz. pos.	R
D40D	54285	0235	565	PENV	Light pen vert. pos.	R

Collision Registers for Players and Missiles

D000	53248	M0PF	Missile 0 - playfield	R
D001	53249	M1PF	Missile 1 - playfield	R
D002	53250	M2PF	Missile 2 - playfield	R
D003	53251	M3PF	Missile 3 - playfield	R
D004	53252	P0PF	Player 0 - playfield	R
D005	53253	P1PF	Player 1 - playfield	R
D006	53254	P2PF	Player 2 - playfield	R
D007	53255	P3PF	Player 3 - playfield	R
D008	53256	M0PL	Missile 0 - player	R
D009	53257	M1PL	Missile 1 - player	R
D00A	53258	M2PL	Missile 2 - player	R
D00B	53259	M3PL	Missile 3 - player	R
D00C	53260	P0PL	Player 0 - player	R
D00D	53261	P1PL	Player 1 - player	R
D00E	53262	P2PL	Player 2 - player	R
D00F	53263	P3PL	Player 3 - player	R

Advertiser's Index

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

Next Month in MICRO

Commodore Feature

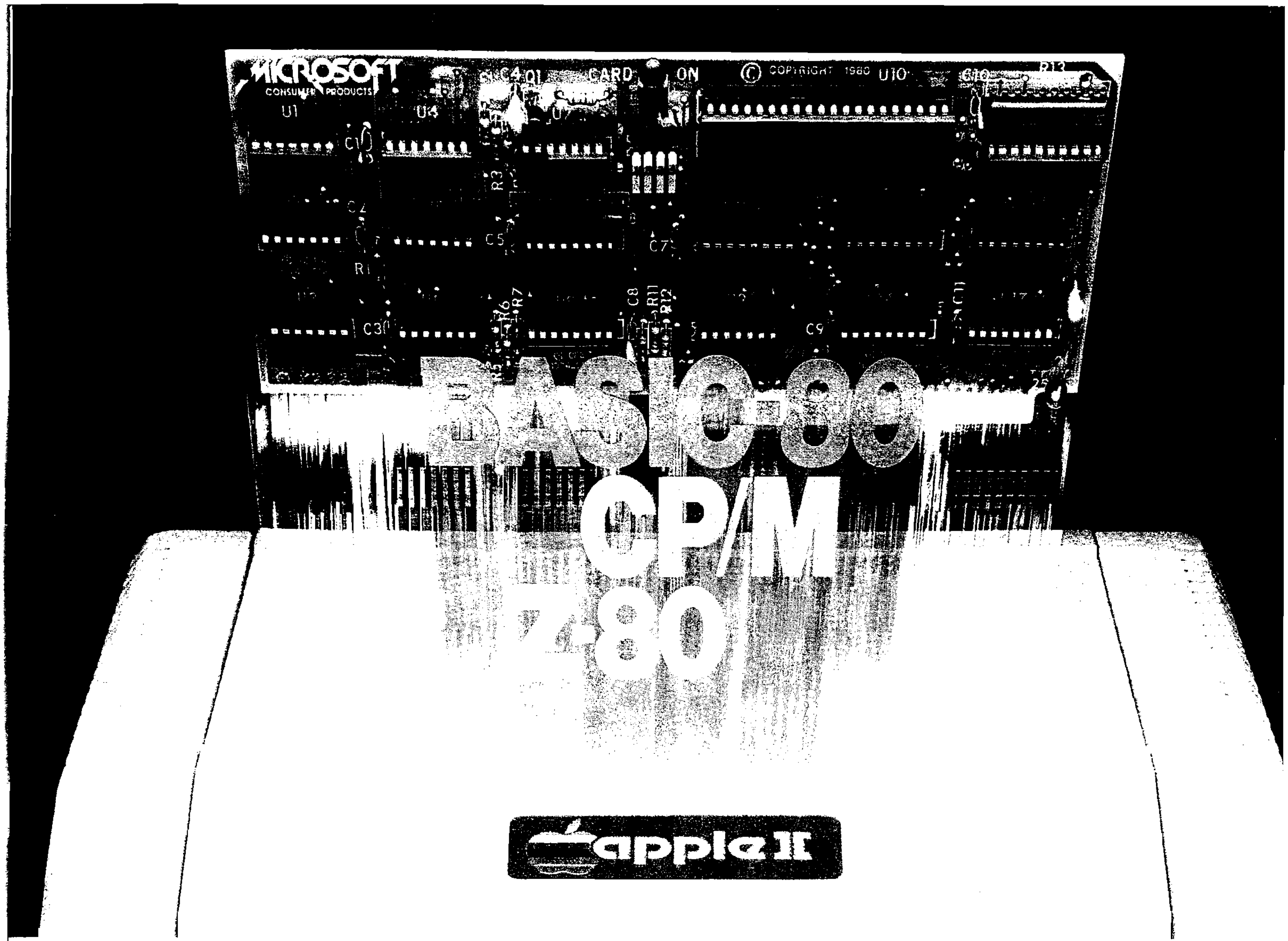
- **SuperPET's Waterloo ASCII Character Set** — A description of the ASCII character set used in the Waterloo interpreters supplied with the SuperPET.
- **BASIC Squeeze** — A routine to squeeze out imbedded blanks, line separators, and comments for a BASIC program.
- **Microcomputers in the Chemical Engineering Curriculum, Part 2** — Analog transducers in a digital world.
- **Add a BASIC Line Delete Command** — A BASIC line delete command allows the user to delete blocks of BASIC program lines at the touch of a single key. The article shows how to implement this command, in machine language, on Commodore computers, including the VIC-20.
- **SOUP** — An efficient compare program for machine-language program files on Commodore disk. Uses BASIC 4.0 disk commands, but is otherwise compatible with other Microsoft BASICs.
- **It's All Relative — CBM Disk Techniques, Part 1** — An explanation of how to get the most from CBM's powerful disk operating system. Examples are drawn from a well-written mailing list package.
- **VIC Jitter Fixer** — Add this routine to your programs to help get reliable readings from your VIC paddle, joystick, and light pen registers.

And more...

BASIC Macro Function for Cursor Control on the OSI
Applesoft GOTO/GOSUB Checking Routine
Logic Instructions of the 68000
Atari Graphics

New Columns!

CoCo Bits — for the Color Computer
From Here to Atari returns



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

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

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

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

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

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

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

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

MICROSOFT
CONSUMER PRODUCTS

MICROSOFT Inc.
10700 Northup Way • Bellevue, WA 98004

THE PROWRITER COMETH.

(And It Cometh On Like Gangbusters.)

Evolution.

It's inevitable. An eternal verity.

Just when you think you've got it knocked, and you're resting on your laurels, somebody comes along and makes a dinosaur out of you.

Witness what happened to the Centronics printer when the Epson MX-80 came along in 1981.

And now, witness what's happening to the MX-80 as the ProWriter cometh to be the foremost printer of the decade.

SPEED

MX-80: 80 cps, for 46 full lines per minute throughput.

PROWRITER: 120 cps, for 63 full lines per minute throughput.

GRAPHICS

MX-80: Block graphics standard, fine for things like bar graphs.

PROWRITER: High-resolution graphics features, fine for bar graphs, smooth curves, thin lines, intricate details, etc.

PRINTING

MX-80: Dot matrix business quality.

PROWRITER: Dot matrix correspondence quality, with incremental printing capability standard.

FEED

MX-80: Tractor feed standard; optional friction-feed kit for about $75 extra.

PROWRITER: Both tractor and friction feed standard.

INTERFACE

MX-80: Parallel interface standard; optional serial interface for about $75 extra.

PROWRITER: Parallel and serial interface standard.

WARRANTY

MX-80: 90 days, from Epson.

PROWRITER: One full year, from Leading Edge.

PRICE

Heh, heh.

Distributed Exclusively by Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021. Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.

LEADING EDGE.™

For a free poster of "Ace" (Prowriter's pilot) doing his thing, please write us.