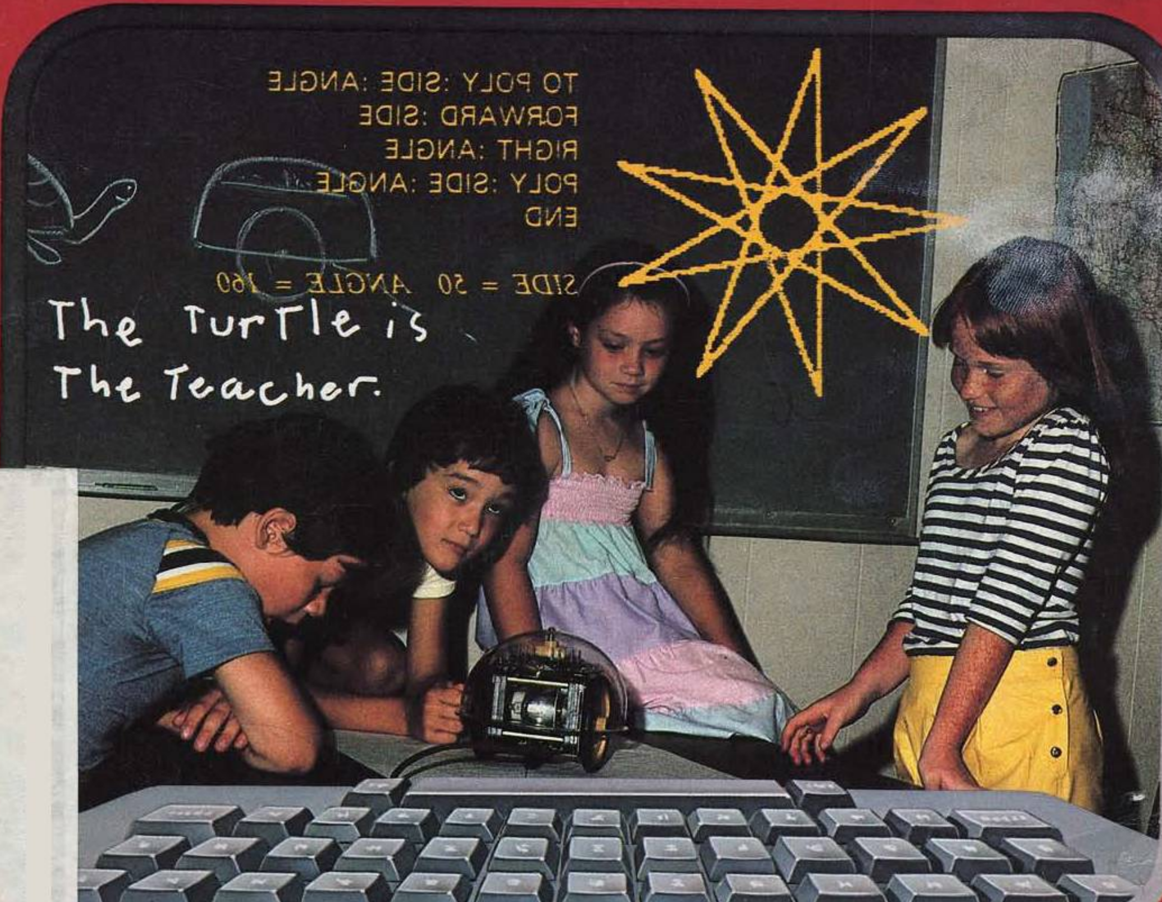
NO. 53 OCTOBER 1982

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £2.00

MICRO

THE 6502/6809 JOURNAL



Education Feature

PET Screen Utilities

Equation Plotting with the Apple

Atari Programming Techniques

Organization a problem?

Look what Sensible Software is doing for you...

DISK ORGANIZER

Here is a fast and friendly utility to organize the files on your diskettes using DISK ORGANIZER. You can order the files on a diskette not only alphabetically, but in any order you want, TITLES can be entered, describing a file or group of files on the diskette. These TITLES may be entered in normal, inverse, flashing, and lower case letters. All work is done on a RAM version of the Directory; so if you decide the changes you made are not correct, you can simply abort the process and keep the old catalog. Since DISK ORGANIZER works on a RAM version of the Directory, disk access is kept to a minimum; hence, DISK ORGANIZER is extremely fast.

The following functions are also included:

- DYNAMIC DISPLAY of all file names in the Directory.
- RENAMING the "HELLO" program.
- A SIMULATED CATALOG to show the modified Directory before it is written to the Diskette.
- ALPHABETIZING the file names.
- UNDELETING deleted files.
- PURGING deleted files.
- RENAMING files (with the same character input options as TITLING.
- LOCKING and UNLOCKING (some or all) files.
- DELETING files.
- DELETING DOS for increased data storage.
- A powerful SMART KEY to automatically locate the next valid file name for any specified operation.

DISK ORGANIZER... Apple II or Apple II Plus with 48K and one or more Disk Drives... **$30.00**

Is inter-disk travel and intra-disk file manipulation causing difficulties?

Then SUPER DISK COPY is for YOU!

Super Disk Copy is much more than just another copy program. It's practically a 'mini-DOS' in itself. Super Disk Copy is completely menu driven and works with DOS 3.1, 3.2, and 3.3. Files may even be transferred from one DOS to another (example: DOS 3.3 to DOS 3.2). Super Disk Copy makes the conversion to DOS 3.3 less painful. After seeing Super Disk Copy, you will agree it is the best copy program on the market! Among the many features are:

- Wildcards may be entered anytime a file is used. They allow both automatic or user verification of the file handling. (Example: to copy file names that start with 'Apple', you would enter 'Apple = ').
- COPY single files (Applesoft, Integer, Text, Binary, Relocatable).
- COPY DOS.
- COPY ENTIRE DISK. Super Disk Copy uses a unique technique for fast copying.
- A BRUTE FORCE COPY made for Pascal, FORTRAN, and CP/M diskettes.
- An optional rearrangement of files so that they occupy contiguous sectors for improved access time.
- UNDELETE deleted files.
- A PLOT of disk usage.
- FIX file sizes. Make sure your valuable files are protected and unused sectors are available.
- DELETE DOS. Frees up extra space on your diskettes.
- ALPHABETIZE file names in directory.
- REPLACE illegal characters in file names.
- Optional 'INIT' of copy diskette.
- LOCK or UNLOCK files and much, MUCH more...

SUPER DISK COPY was one of only two programs to receive a rating of 100 in PRICE/USEFULNESS RATIO in THE BOOK OF APPLE COMPUTER SOFTWARE 1982. They said "This is a definite MUST HAVE utility package."

48K and DISK II required... **$30.00**

For a complete catalog send $1.00, refundable with your first purchase.

If you have a large software collection and have problems locating specific programs or you need to be reminded of what they do...then you need MULTI-DISK CATALOG

Multi-Disk Catalog is designed specifically for keeping track of the contents of the APPLE diskette library. The resulting master catalog can be sorted, searched, and printed. Multi-Disk Catalog is entirely menu driven, easy to use, and very fast. Some of the unique features included are:

- Works with 13 and 16-sector formated diskettes (DOS 3.1, 3.2, and 3.3). Diskettes from either type DOS can be added to the same data-base used.
- You assign the volume identification number (range 1 to 999); there is no need to INIT each of your disks with a unique volume number.
- A 2 letter classification code can be added to each file name. The classifications are retained when the disk volume is updated.
- Multi-key sorting. Up to 3 keys may be selected for sorting the file names. The sort keys include disk number, file name, classification, file type, and file size.
- Multi-Disk Catalog is written entirely in machine language, so it is very fast.
- A powerful search mask can be used to find a file name or groups of file names.
- Multi-Disk Catalog supports special printer interfaces.
- Both sides of a diskette may be entered under the same disk number.
- Titles may be entered for each diskette and much more!

"There is no doubt about it, this is the best and most versatile master catalog program available"—THE BOOK OF APPLE COMPUTER SOFTWARE 1982.

48K and one Disk Drive required... **$25.00**

Sensible Software

6619 Perham Drive • West Bloomfield, Michigan 48033 • Phone (313) 399-8877

Visa and MasterCard welcome, please add $1.25 postage and handling per diskette.

MICRO™

October Highlights

Education Feature

Our educationally-oriented selections cover a broad range of computer use in education — from elementary and secondary schools to a college teaching laboratory to a public health-care facility.

Gloria Stein, a fifth grade teacher, has been using microcomputers in her classroom for three years (p. 49). On the whole, it has been a very positive experience. Florence Taber (p. 47) discusses the use of micros in special education settings and Judith Toop (p. 47), the use of micros in a high school chemistry classroom.

John Ellis and Lynda Raines (p. 59) have been using Apple computers in a public health-care facility to educate clients on a wide variety of health topics. They have overcome a number of problems in getting untrained users to use the computers. Dr. Saltsburg and his colleagues at the University of Rochester have been using PETs in their undergraduate chemical engineering laboratories. A four-part series with an overview of the computers' involvement begins on page 53.

Finally, Edward Carlson (p. 41) reviews three different versions of the educational programming language LOGO. The designers of these implementations have taken different approaches.

Commodore

Two articles in this month's education feature relate to Commodore machines. Saltsburg, Heist, and Olson (p. 53) begin a series describing the implementation of PET computers in the University of Rochester's chemical engineering teaching laboratories. Gloria Stein (p. 49) discusses the ongoing involvement of PETs in her Ann Arbor, MI elementary school.

Thomas Henry (p. 11) discusses machine-language techniques for controlling the screen. He gives special attention to the special features of the CBM 8032. In our "Short Subjects" section (p. 72), Mike Casella presents a cassette fast program locator for the VIC.

This month's PET Vet (p. 69) discusses the structured features of Waterloo microBASIC on the SuperPET.

Apple

This issue offers a great deal of interest to Apple owners. Eugene Rolfe's "Autograph" (p. 15) demonstrates the use of an EXEC file to insert mathematical equations into a hi-res plotting routine.

In the Education section of this issue, two of the articles relate to the Apple. Edward Carlson discusses the different versions of LOGO available for the Apple in "The Three Faces of LOGO" (p. 41). Lynda Ellis and John Raines discuss the design of menus for unsupervised computer use in "A Personal Computer for Untrained Users" (p. 59).

In "Apple Pascal Turtle Graphics" (p. 37), Mr. Raines provides a program to dump Pascal hi-res screens. Tim Osborn's "Apple Slices" column discusses a technique to transfer Pascal machine-language programs from a language disk to DOS 3.3 format disks.

6809

By implementing some Apple Pascal functions in 6809 code instead of 6502, significant speed improvements can be made. Tom Whiteside discusses these improvements in depth (p. 79).

Ralph Tenny shows how to quickly and inexpensively upgrade your Color Computer to 32K in "The Homespun 32K Computer."

Atari

Frank Roberts (p. 75) presents an Atari BASIC subroutine for flexible formatting of numerical data. It is particularly useful for making sure that decimal points line up and for handling trailing zeros in dollar figures.

Contributing editor Paul Swanson begins a three-part tutorial series (p. 87) on using character graphics to mimic some of the capabilities of the memory-consuming high-resolution modes. Paul's Atari column begins next month.

Return Your Questionnaire!

Did you miss our questionnaire near the back of your September MICRO? Please be sure to take a few minutes to complete it. We need this information to help keep us in touch with our readers' needs. Mail it today. The postage is on us.

MICRO™

It Pays to Write for MICRO!

Publishing an article in MICRO means quick distribution of your ideas to thousands of people *and* provides you with a little extra income. MICRO is sold in computer stores and on newsstands all over the world! Our circulation is growing by leaps and bounds. **And we pay as well as *BYTE!*** Send for a copy of our Writer's Guide now.

MICRO is actively seeking articles to present to our audience of sophisticated computerists. We welcome contributions on any aspect of 6502/6809/68000 hardware and software for the Apple, Atari, CBM/PET, TRS-80 Color Computer, VIC, 6809, or 68000. Topics of particular interest for upcoming issues are Programming Languages (besides BASIC), Communications, Operating Systems, and New Computers.

NO. 53 OCTOBER 1982

MICRO™

THE 6502/6809 JOURNAL

EDUCATION FEATURE

RUNTIME UTILITIES

HARDWARE

TUTORIAL

COLUMNS

STAFF

President/Editor-in-Chief
ROBERT M. TRIPP

Publisher
MARY GRACE SMITH

Editorial Staff
PHIL DALEY — Technical editor
JOHN HEDDERMAN — Jr. programmer
MARJORIE MORSE — Editor
JOAN WITHAM — Editorial assistant
LOREN WRIGHT — Technical editor

Graphics Department
HELEN BETZ — Director
PAULA M. KRAMER — Production mgr.
EMMALYN H. BENTLEY — Typesetter

Sales and Marketing
CATHI BLAND — Advertising mgr.
CAROL A. STARK — Circulation mgr.
LINDA HENSDILL — Dealer Sales
MAUREEN DUBE — Promotion

Accounting Department
DONNA M. TRIPP — Comptroller
KAY COLLINS — Bookkeeper
EILEEN ENOS — Bookkeeper

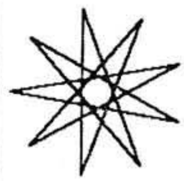
Contributing Editors
DAVE MALMBERG
JIM STRASMA
PAUL SWANSON
RICHARD VILE

Advertising Sales Representatives
See Page 127

Subscription/Dealer inquiries
(617) 256-5515

DEPARTMENTS

About the Cover



```
TO POLY :SIDE :ANGLE
FORWARD :SIDE
RIGHT :ANGLE
POLY :SIDE :ANGLE
END
```

SIDE = 50 ANGLE = 160

The "turtle" shown on the cover is actually a computer-controlled robot. Students, using a new educational language called LOGO, can easily program the turtle to move and draw pictures.

Thanks to Terrapin, Inc. for the use of a turtle. They are one of three companies that sell LOGO for the Apple.

Pictured from left to right are Michael Blute, Dustin A. Bland, Tricia Blute, and Emily Dube Ferrier. The MICRO Kids are the offspring of Dawn Blute, receptionist, Cathi Bland, advertising manager, and Maureen Dube, promotion.

Cover photo by Phil Daley

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

Are We Ready?

Schools today are at the threshold of a revolution in educational technique that rivals the importance of the invention of classroom learning. According to John Herriot (*Creative Computing*, April 1982), "There is a very strong possibility that before the end of the century students will be receiving all of their instruction from computers, with no contact with live teachers whatsoever."

Some of us may think it is more important to ensure that this revolution is beneficial to all aspects of the learning process rather than to emphasize the removal of human contact from education. Nevertheless, the main point is that there are hundreds of thousands of microcomputers in classrooms across the nation and there will be more. Prices will continue to drop due to intense international competition and increased technological breakthroughs. Apple Computer is, according to published reports, attempting to put an Apple II in every school in the country even if they have to give them away. Frost and Sullivan predict that by 1990 there will be 15 million computers in individual homes in the USA.

Some teachers and administrators may insist that the current trend is just a fad like the "new math" was several years ago, but there is no doubt that more and more computers are more and more accessible to more and more children. The full impact of the computerization of education has not yet been realized either by parents or educators or the general public. As computer-knowledgeable students reach the marketplace with an ever-increasing skill level, computer literacy will become a requirement in our information-oriented society. In the future, computer literacy will be as important as the ability to read is today. *Newsweek* predicts that by 1985 75% of all jobs will be computer-related. Now is the time to prepare for the new learning environment.

So where should schools begin? Many teachers may have only superficial training in mathematics and other technical areas. Others, trained in the humanities, may be resistant to computerized learning. It is important, therefore, to offer teachers non-threatening in-service training to help them view the microcomputer as a new teaching tool, as well as to help them become sophisticated users themselves.

One significant feature of Computer-Assisted Instruction is that students learn without realizing they are being educated. Simulation programs teach politics, economics, sociology, and natural science while students think they are only playing. For that ploy to work, programs must run easily and must be highly graphic. A package with screens full of text might as well be printed in a book. High-quality educational software should use lots of graphics, be fast-paced, provide immediate feedback, have clear-cut goals, and offer challenging levels of play. Only in that way will the screen hold the attention of the student the way an arcade game does. The advent of interactive video disk programs will further increase the quality of graphics presentations.

When all educators realize how excited students are by the interactivity and versatility of the computer as a teacher, even greater strides will be made in the use of the new technology to teach creative thinking, problem-solving, and logic. Even very young children are not intimidated by microcomputers. For example, Stanford's Bing Nursery School is using micros to teach preschool children basic skills. Children's Computer Workshop, a subsidiary of Children's Television Workshop, is marketing educational video games that encourage children to exercise logic and problem-solving techniques as they play. Students can progress from using well-written software to programming in Logo and BASIC where they not only have to understand a new language, but also the logic of solving a problem. The important thing for all of us to realize is that for educational purposes the computer teaches much more than subject matter. It may be difficult to imagine the educational system forecast by Mr. Herriot, but with proper planning and creative use microcomputers can be highly beneficial to the educational attainment of all students. The ultimate impact of widespread computer knowledge on society will be that the educational environment will be able to promote the acquisition of advanced problem-solving tools and modes of logical thought. We cannot have too many of those attributes in the modern world.

Phil Daley

Letters/Updates

Tom Hackley of Sunnyvale, CA, sent us this revision:

A minor modification to A.J. Zadiraka's Auto Entry Utility Program (MICRO 50:93) adds a view memory capability. This change removes the ESC key exit to the monitor and substitutes the space key, which displays the contents of the current address and increments to the next address. Holding down the space key invokes the auto repeat function, which rapidly fills the screen with a scrolled display of the memory contents.

The changes are in the figure to the right.

Hackley's Revision

```
03AA: C9 20                CMP #$20          ; SPACE KEY
03AC: F04A                 BEQ EXIT3

03F8: B1FE        EXIT3    LDA (ADRLO),Y     ; GET DATA
03FA: 20 CF 03             JSR UNPACK        ; DISPLAY IT
03FD: 68                   PLA               ; FIX STACK
03FE: 68                   PLA
03FF: 68                   PLA
0400: 68                   PLA
0401: 4C 49 03             JMP LOOP+11       ; NEXT ADDRESS
```

Marvin L. DeJong from Point Lookout, MO, caught an error in his article, "Timing and Counting with the 6522," which appeared in MICRO (50:13).

Line 60 should read:

```
60 POKE 10, 76 : POKE 11, 12 :
   POKE 12, 16
```

Tim Osborn in Manchester, NH, offers a correction to his Apple Slices column (51:108). Line one, listing 2, on page 110 should read as follows:

```
1 BFR = 8192: PRINT CHRS (4)
"BRUN AMPERRWTS.CODE,$8000"
```

MICRO™

FOR YOUR APPLE II

Industry standard products at super saver discount prices

PARALLEL PRINTERS
NEC 8023 or C-ITOH 8510

(Virtually identical) Specifications: • 100 CPS dot matrix printer • 80 column print—136 characters per line • Tractor/friction feed • 7 different print fonts included • 2K printer buffer • Proportional spacing • Bit image graphics and graphic symbols.

NEC 8023 or C-ITOH **$495**
NEC 8023 or C-ITOH 8510 with Parallel Interface and Cable **$550**
EPSON 100 with Parallel Interface and Cable **$749**

Z-80 CARD FOR YOUR APPLE
MICROSOFT SOFTCARD

With CP/M* and MBASIC.

(List: $399) **$289**

NEW! *Best Buy!!!*
ADVANCED LOGIC SYSTEM Z-CARD With C-PM*

Has everything the Softcard has except MBASIC. Works with Microsoft's disks too.

(List $269) ...... **Special at $195**

NEW!
ALS SYNERGIZER

CP/M* operating package with an 80 column video board, CP/M* interface, and 16K memory expansion for Apple II. Permits use of the full range of CP/M* software on Apple II. Includes SuperCALC.

(List: $749) **$549**

NEW!
U-Z-80 PROCESSOR BOARD (From Europe)

Software compatible with Softcard and ALS Software **$149**

MICROSOFT + PREMIUM SYSTEM

Includes Videx Videoterm, Softswitch, Microsoft and Softcard, Microsoft and Z-80 Card, and Osborn CP/M* Manual **$595**

JOYSTICK

Takes the place of two Apple Paddle Controllers.

From BMP Enterprises. Heavy duty industrial construction and cable. Non-self centering. With polarity switches for consistent motion control.

(List: $59) **$39**

MONITORS FOR YOUR APPLE

AMDEK 300G (18MHZ Anti-Glare Screen) **$179**
NEC 12" HIRES GREEN **$179**

SUPER SPECIAL!
SPECIAL 12" GREEN MONITOR **$99**

SPECIAL AND NEW

5 MEGABYTE HARD DISK

For Apple II. Supplied with controller. Use with CP/M, Apple DOS, & Apple Pascal **$1995**

5¼" DISK DRIVE

Use with standard Apple II disk controller. .**$295**

5¼" FLOPPY DISKS

With hub rings. Box of 10.

With other purchase **$19.95**
Without purchase **$23.00**

16K MEMORY EXPANSION MODULE

The preferred 16K RAM Expansion Module from PROMETHEUS. Fully compatible with CP/M* and Apple Pascal*. With full 1-year parts and labor warranty. (List: $169) **$75**

WORD PROCESSING SPECIAL WITH WORDSTAR AND SUPERCALC!

Do professional word processing on your APPLE. All necessary hardware and software included. Complete 80 column video display, enhanced character set, 16K memory board, Z-Card with CP/M* software, Wordstar and word processing software and SuperCALC.

(List: $1,128) .... **Special at $695**

NEW! *from Prometheus!* **ExpandaRAM**

The only 128K RAM card that lets you start with 16K, 32K, or 64K of memory now and expand to the full 128K later. Fully compatible with Apple Pascal, CP/M*, and Visacalc. No Apple modification required. Memory management system included with all ExpandaRAMs. Disk emulators included with 64K and 128K versions.

MEM-32 Two rows of 16K RAMS make a 32K RAM Card **$209**
MEM-64 One row of 64K RAM. With DOS 3.3 disk emulator **$299**
MEM-128 Two rows of 64K RAMS installed make a 128K Card. With DOS 3.3 disk emulator **$399**
MEM-RKT 64K RAM Add-On-Kits— 64K Dynamic RAMS. Each **$125**
VISICALC Expansion Program for MEM-128 .... **$75**
MEM-PSL Pascal disk emulator for MEM-128 .... **$45**

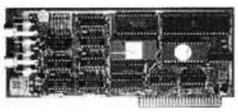
MODEMS FOR YOUR APPLE II

HAYES Smartmodem **$229**
MICROMODEM II **$279**

VERSAcard FROM PROMETHEUS

Four cards on one! With true simultaneous operation. Includes: (1) Serial Input/Output Interface, (2) Parallel Output Interface, (3) Precision Clock/Calendar, and (4) BSR Control. All on one card. Fully compatible with CP/M* and Apple Pascal*.

(List: $249) **$169**



80 COLUMN VIDEO DISPLAYS FOR APPLE II
SMARTERM

(Not to be confused with SUPRTERM)

Software switching from 80 to 40 and 40 to 80 characters. 9 new characters not found on the Apple keyboard. Fully compatible with CP/M* and Apple PASCAL*. With lowest power consumption of only 2.5 watts.

(List: $345) **$225**

SMARTERM EXPANDED CHARACTER SET

7" x 11" matrix with true decenders. Add to above **$40**

Best Buy! **Combination SMARTERM and EXPANDED CHARACTER SET**

Special at **$260**
VIDEX, VIDEOTERM **$249**
VIDEX ENHANCER II **$119**

NEW! **CENTRONICS COMPATIBLE PARALLEL INTERFACE**

From PROMETHEUS. For use with Epson, NEC, C-ITOH, and other printers. Fully compatible with CP/M* and Apple Pascal*.

PRT-1, Only **$69**

GRAPHITTI CARD

Prints HIRES page 1 or 2 from onboard firmware. Features: True 1:1 aspect ratio, prints emphasized mode, reverse mode, rotates 90 degrees ... plus more. Compare all this with the Grappler. We think you'll agree that this is the best graphics card on the market. Specify for use with EPSON, NEC-8023, C-ITOH Prowriter, or Okidata.

(List: $125) **$89**

SOFTWARE

WORDSTAR .... **Special at $195**
SPELLSTAR **$125**
SUPERCALC **$175**
D BASE II **$525**
VISICALC **$149**
DB MASTER **$189**

All equipment shipped factory fresh. Manufacturers' warranties included. Please add $3.00 per product for shipping and handling. California: add 6% tax; BART Counties: 6½%.

All items are normally in stock

Phone for Quick Shipment!

(415) 490-3420

... And we'll be here to help after you receive your order. Feel free to call the SGC Technical Staff for assistance.

SGC

The mail order specialists

342 Quartz Circle, Livermore, CA 94550

Machine-Language Screen Utilities for the CBM-8032

by Thomas Henry

A discussion of the screen-related ROM subroutines of the CBM-8032. These routines, called by machine-language programs, lead to attractive formatting of screen displays. In addition, the author explains the cursor positioning mechanism of Commodore computers.

The Commodore CBM-8032, with its 80-column screen, makes attractive formatting of displays a snap. Included in its BASIC interpreter are special commands that make the most of the large screen. These include commands to create windows, delete lines, insert lines, scroll up or down, and so on. But how are these functions implemented in machine-language programming? These and many other useful subroutines are located in the CBM-8032 ROMs and can be called up to create beautiful displays.

First you have to get a character to the screen. Just load the accumulator with the Commodore ASCII code for the character (not the PEEK/POKE number), and jump to subroutine $FFD2. This procedure will display the character at the current screen location. For example, if you want to display an "&" at the current screen location, use the following code:

```
LDA #$26
JSR $FFD2
```

That's all it takes! You can print any character to the screen with this subroutine, including the cursor controls and clear screen functions. For example, to HOME the cursor, load the accumulator with #$13 and JSR $FFD2. To move the cursor (or current screen location) to the right, load the accumulator with #$1D and call the subroutine.

As users of Commodore equipment know, cursor controls can be either direct or programmed. A direct cursor control performs the function immediately. For instance, if you hit the [CURSOR RIGHT] key, the cursor will immediately move to the right. However, if you type the BASIC line:

```
10 PRINT "[CURSOR RIGHT]
     This is an example."
```

where [CURSOR RIGHT] is the cursor right key, then listing the line shows a reverse Q image indicating that, when the line is executed, a cursor right is to be performed. Now note the difference: CURSOR RIGHT can mean to actually move the cursor right, or to store a special character that will cause the movement to be executed when the program is run.

Machine-language programming does not require this programmed-mode cursor control. When you send a cursor command to $FFD2, you want it performed immediately. To insure that this happens, clear a flag in the zero page. Location $CD tells the CBM-8032 whether a cursor control is direct or programmed. Set this location to zero and the cursor control is in the direct mode. Set it to a one and the cursor is in the programmed mode. A short way to clear $CD to zero is to perform a logic shift right. For example, move the cursor down one row:

```
LSR $CD
LDA #$11
JSR $FFD2
```

The LSR command will shift out bit zero and replace it with bit one. Bit zero may have been a one or a zero, depending on previous conditions, but the higher bits are always zeros. The net effect is that location $CD has been cleared. The next two commands then execute the CURSOR DOWN function.

Before leaving the subject of programmed *vs.* direct cursor, note that whenever an odd number of quotes has been printed to the screen, location $CD is set to one. When an even number of quotes has been printed, $CD is cleared. A carriage return also clears $CD. Another interpretation of this flag is "Are you in quotes or out of quotes?".

As mentioned before, routine $FFD2 will print a character at the current screen location. How do you set this location to a specific row and column? You could load the accumulator with CURSOR UPs, DOWNs, RIGHTs, LEFTs, and HOMEs, and print them to the screen in the proper order to land the screen pointer at the desired location. But there is an easier way. Locations $C4 and $C5 contain the low byte and high byte, respectively, of the current screen line. This pointer contains the screen location of the first space on a line for each of the 25 lines. For example, if the cursor were sitting in the HOME position, then the current screen line pointer would be pointing to $8000. If the cursor were moved down one line, $C4 and $C5 would be pointing to $8050.

Now that you have a pointer to the screen line, what is the position on this line? Location $C6 will contain a number from #$00 (the first column) to #$4F (last column). This, then, is all you need to print a character to any desired screen location. First load $C4 and $C5 with the proper screen line pointer and then load $C6 with the proper column. Next load the accumulator with the desired character and call subroutine $FFD2. The CBM-8032 will automatically add the contents of the screen line pointer to the column pointer, and then display the character at that position.

That's pretty slick! However, you still need to know the screen line pointer for the various rows. For example, line one is $8000, line two is $8050, line three is $80A0, and so on. This series is certainly not our favorite set of numbers to remember. It would be easier to specify just the row and column of the desired screen location and have the computer do the rest. You can do this! If you want the character "&" printed at the tenth row and fifth column, use the following routine (remember to start counting rows and columns from zero):

```
LDA #$0A
STA $D8
LDA #$05
STA $C6
JSR $E067
LDA #$26
JSR $FFD2
```

As you can see, location $D8 contains the row and $C6 the column. Subroutine $E067 then computes the proper screen line pointer, based on the contents of $D8, and deposits the result in $C4 and $C5 automatically! You have reduced your work to specifying simply the row and column where you want something printed.

Creating windows is a simple procedure. Location $E0 contains the number of the top line desired, and location $E1 contains the line number of the bottom of the window. $E2 contains the left window margin, and $D5 contains the right window margin. Simply load these locations with the desired data and a window will be created. Subroutine $FFD2 automatically compensates because it knows the new limits of the new window. For example, create a window that starts at line 3, ends at line 17, and has columns 4 and 63 for the left and right borders:

```
LDA #$03
STA $E0
LDA #$11
STA $E1
LDA #$04
STA $E2
LDA #$3F
STA $D5
```

This short machine-language routine will then create the desired window.

How about writing your own BASIC command to create a window? A command line like this would be nice:

```
WINDOW 3,17,4,63
```

It creates the same window as that given in the machine-language routine above. A programming challenge then, is to create the needed code to implement this new BASIC statement. Hint: you'll have to intercept the normal CHARGET routine and divert it to your own subroutine first.

(Ed note: Implementing such a command would slow down the execution of all BASIC commands. The convenience may not be worth the cost.)

There are two routines built into the CBM-8032 for clearing the screen and homing the cursor. To clear the screen simply JSR $E051. This not only clears the screen but homes the cursor as well. Note that you could load the accumulator with the value of the CLEAR key and then jump to subroutine $FFD2 to clear the screen, but this takes two extra bytes. To home the cursor simply JSR $E05F. This, too, is a much quicker and more efficient method.

I hope this article has given you ideas on how to use the excellent routines built into the CBM-8032. If you have a disassembler you might take a look at locations $E000 through $E788 to get more ideas on how the screen editing functions are implemented. Happy experimenting!

The author may be contacted at Transonic Laboratories, 249 Norton Street, Mankato, Minnesota 56001.

MICRO

Auto Graph for the APPLE

by Eugene J. Rolfe

The following equation-plotting programs illustrate how an EXEC file on an Apple II Plus can be used to allow a BASIC program to modify itself. Several Applesoft and DOS features are utilized: sequential files, EXEC files, ONERR, and HGR.

AUTO GRAPH
requires:

Apple II with 48K
Disk drive and Applesoft

Graphing Equations

Many plotting routines require that the input information be in the form of data. Generating this data can be quite tedious for complicated functions. The programs described here allow equations to be graphed directly.

The equation must be in a form where Y is expressed as a function of X, and must be compatible with the syntax of Applesoft. The program does not check the syntax of the equation. Applesoft will do this itself, and errors will be trapped by using an ONERR GOTO. Any of the numeric functions and mathematical operators available in Applesoft can be used in the equation.

The equation to be plotted is entered into a string variable by using an INPUT statement. A line number is concatenated onto the equation string. The combined string is then stored in a sequential file, named EQ. When this file is EXECed, the equation is inserted into the program stored in memory.

Remember that commands stored in an EXEC file work very much like commands typed at the keyboard. Statements with line numbers will be inserted into the program stored in memory, line numbers alone will delete program lines, and commands without line numbers will be executed immediately.

To demonstrate this plotting technique, the first program to be examined will be a simplified version that only plots in one quadrant (X and Y both positive). This simplified program operates over a fixed range of X and Y; $0 <= X <= 279$ and $0 <= Y <= 191$. This range allows the user to become familiar with the technique without the burden of excessive detail. The line numbers of the first example are arranged to make them as compatible with the second program as possible. In the second example, the program allows any range of values for X (within the limits of Applesoft). This program then calculates the range of Y and automatically scales the screen accordingly.

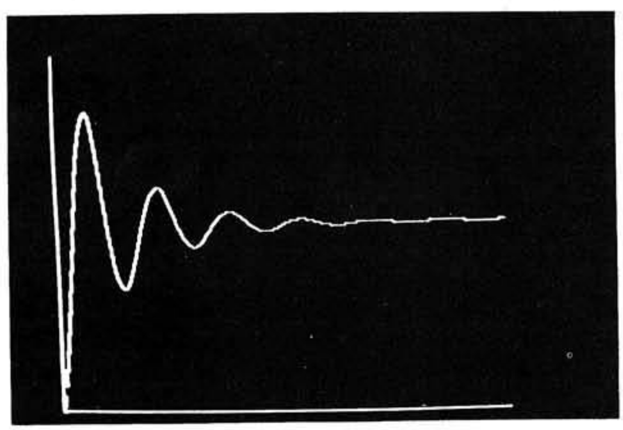
No special instructions are required to operate these programs. However, note that the cursor will not be visible at the conclusion of these programs. Type any character to return to the text mode. Any time you wish to plot the same equation again, type RUN 150.

Warning! If these programs are renumbered, the line numbers used as strings and constants within the BASIC statements must be changed manually.

Expanding the Program

Allowing an EXEC file to modify

Example of Simple Graph Y = 100*(1 − EXP(− X/40)*COS(X/7))



***Listing 1:* Simple Graph**

```
10   TEXT : HOME
20   PRINT "ENTER EQUATION (EXAMPLE: Y=X+4)"
30   PRINT
40   INPUT "";E$
50 F$ = "1010 " + E$
60 D$ =  CHR$ (4)
70   PRINT D$;"OPEN EQ"
80   PRINT D$;"DELETE EQ"
90   PRINT D$;"OPEN EQ"
100   PRINT D$;"WRITE EQ"
110   PRINT F$
120   PRINT "RUN 170"
140   PRINT D$;"CLOSE"
150   PRINT  CHR$ (4);"EXEC EQ"
160   END
170   HGR2
250   ONERR  GOTO 2500
```

(continued)

and then run a program resets all of the variables. Therefore, parameters cannot be passed directly from the portion of the program before the EXEC, to the portion after the EXEC. The second example demonstrates one method of solving this problem.

The second version of the program expands the graphing capabilities to all four quadrants. The mixed-screen format allows the equation and scaling information to be displayed below the graph. Also, the error handling routine is improved.

Description of Complete Program

Line 130 stores the contents of E$ to be passed to the second half of the program. Thus, the equation is written into the file as an ASCII string without a line number. See figure 2 for an example of the contents of an EQ file.

Line 170 is the point where the program is restarted when the 'RUN 170' in the EXEC file is executed. Since a BASIC program is now running again, the EXEC file waits before supplying any more commands. When the INPUT statement requests an entry for E$, the string is taken from the EQ EXEC file, which is still open. Then the EXEC file automatically closes itself because all of the lines in the file have been used. This allows subsequent INPUT statements to accept input from the keyboard.

Line 180 contains the INT function, which is often used to truncate numbers. However, the results of truncating negative numbers may be surprising; INT $(-6.6) = -7$. This increases the magnitude of the negative argument and can actually force points off the screen in this program. This user-defined function corrects this problem and allows negative numbers to be truncated; FN I$(-6.6) = -6$.

Line 260 exchanges values if the ending point of the graph is less than the starting value.

Lines 270-290 calculate the range of X and the displacement of the Y-axis.

Line 300 calculates the increment for X.

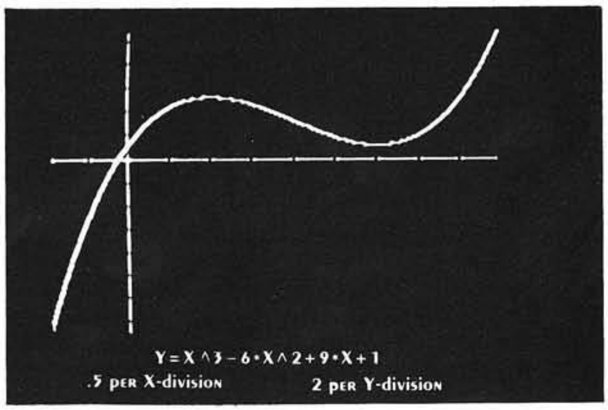
Line 310 scales the displacement of the Y-axis for proper placement on the screen.

Lines 330-380 find the maximum and minimum value of Y, using zero as a default value.

Listing 1 *(continued)*

```
490   HCOLOR= 3
500   HPLOT 0,0 TO 0,191 TO 279,191
520 X = 0
530   GOSUB 1010
540 Y = 191 - Y
550   HPLOT X,Y
560   FOR X = 1 TO 279
570   GOSUB 1010
580 Y = 191 - Y
585   IF Y > 191 OR Y < 0 THEN 600
590   HPLOT  TO X,Y
600   NEXT X
610   PRINT  CHR$ (7): GET N$: TEXT : END
1010  REM  EQUATION GOES HERE
1020  RETURN
2500  POKE 216,0: REM  TURN OFF ONERR
2510  TEXT : HOME
2520  PRINT "CAN'T PLOT": PRINT
2530  PRINT "X = ";X
2540  PRINT : LIST 1010
2550  END
```

Example of Auto Graph — Range of X: -1 to 4.5



***Listing 2:* Auto Graph**

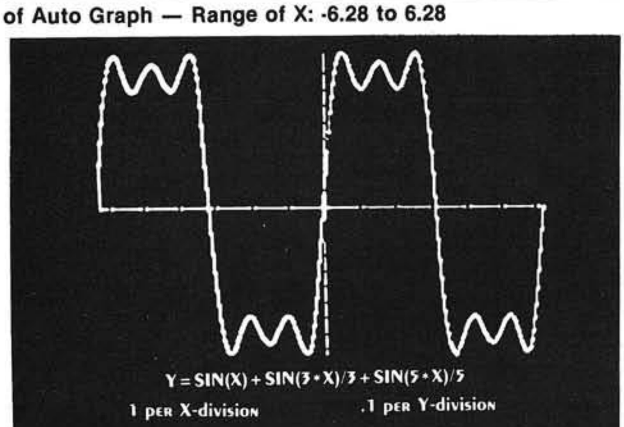
```
10  TEXT : HOME
20  PRINT "ENTER EQUATION (EXAMPLE: Y=X+4)"
30  PRINT
40  INPUT "";E$
50 F$ = "1010 " + E$
60 D$ =  CHR$ (4)
70  PRINT D$;"OPEN EQ"
80  PRINT D$;"DELETE EQ"
90  PRINT D$;"OPEN EQ"
100  PRINT D$;"WRITE EQ"
110  PRINT F$
120  PRINT "RUN 170"
130  PRINT E$
140  PRINT D$;"CLOSE"
150  PRINT  CHR$ (4);"EXEC EQ"
160  END
170  INPUT E$
180  DEF  FN I(X) =  INT (X + (1 -  SGN (X)) / 2.00000001)
190  HOME
200  PRINT E$
210  PRINT
220  INPUT "ENTER STARTING VALUE OF X: ";XS
230  PRINT
240  INPUT "ENTER ENDING VALUE OF X: ";XB
250  ONERR  GOTO 2500
260  IF XB < XS THEN T = XB:XB = XS:XS = T
270  IF XS <  = 0 AND XB > 0 THEN XR = XB - XS:XC =  - XS
280  IF XS < 0 AND XB <  = 0 THEN XR =  - XS:XC =  - XS
290  IF XS > 0 AND XB > 0 THEN XR = XB:XC = 0
300 XI = XR / 279
310 XZ = XC / XI
320  PRINT : PRINT "PLEASE WAIT!"
330 YS = 0:YB = 0
340  FOR X = XS TO XB STEP XI
350  GOSUB 1010
360  IF Y > YB THEN YB = Y
370  IF Y < YS THEN YS = Y
380  NEXT X
390  IF YS <  = 0 AND YB > 0 THEN YR = YB - YS:YC =  - YS
400  IF YS < 0 AND YB <  = 0 THEN YR =  - YS:YC =  - YS
```

(continued)

Listing 2 *(continued)*

```
410   IF YS > 0 AND YB > 0 THEN YR = YB:YC = 0
420 YI = YR / 159
430 YZ = YC / YI
440 YZ = 159 -  INT (YZ)
450  HGR
460  VTAB 22
470  IF  LEN (E$) < 40 THEN  HTAB (40 -  LEN (E$)) / 2
480  PRINT E$
490  HCOLOR= 3
500  HPLOT 0,YZ TO 279,YZ
510  HPLOT XZ,0 TO XZ,159
520 X = XS
530  GOSUB 1010
540 Y = 159 - (Y + YC) / YI
550  HPLOT (X + XC) / XI,Y
560  FOR X = XS + XI TO XB STEP XI
570  GOSUB 1010
580 Y = 159 - (Y + YC) / YI
590  HPLOT  TO (X + XC) / XI,Y
600  NEXT X
610 R = XR
620  GOSUB 770
630  HCOLOR= 0
640  IF XS < 0 AND XB < 0 THEN XB = 0
650  IF XS > 0 AND XB > 0 THEN XS = 0
660  FOR X =  FN I(XS / DI) * DI TO  FN I(XB / DI) * DI STEP DI
670  HPLOT XZ + X / XI,YZ
680  NEXT X
690  VTAB 24: PRINT DI;" PER X-DIVISION";
700 R = YR
710  GOSUB 770
720  FOR Y =  FN I(YS / DI) * DI TO  FN I(YB / DI) * DI STEP DI
730  HPLOT XZ,YZ - Y / YI
740  NEXT Y
750  PRINT  SPC( 4);DI;" PER Y-DIVISION";
760  GOTO 2650
770 K = 0
780  IF R < 1 THEN 840
790  IF R < 10 THEN 870
800 K = K + 1
810 R = R / 10
820  GOTO 790
830  IF R > 1 THEN 870
840 K = K - 1
850 R = R * 10
860  GOTO 830
870 DI =  INT (R) * 10 ^ (K - 1)
880  RETURN
1010  REM  EQUATION GOES HERE
1020  RETURN
2500  POKE 216,0: REM  TURN OFF ONERR
2510 EA =  PEEK (218) +  PEEK (219) * 256
2520 E2$ = "ERROR"
2530 N$ = "":E1$ = ""
2540 IF EA < > 1010 THEN N$ = "NOT ":E1$ = " AT LINE " + STR$(EA)
2550  HOME : VTAB 21
2560 ER =  PEEK (222)
2570  IF E$ = "" THEN  PRINT "ERROR - NO EQUATION.": END
2580  PRINT "ERROR IS ";N$;"IN EQUATION."
2590  PRINT E$
2600  IF EA <  > 1010 THEN 2620
2610  IF ER = 16 OR ER = 163 OR ER = 224 THEN E2$ = "BAD EQUATION"
2620  IF ER = 53 OR ER = 69 OR ER = 133 THEN E2$ = "BAD VALUE"
2630  IF ER = 255 THEN E2$ = "CONTROL-C"
2640  PRINT E2$ + E1$
2650  VTAB (15): PRINT  CHR$ (7): GET N$: TEXT
2660  END
```

Example of Auto Graph — Range of X: -6.28 to 6.28



Example of the Contents of an EQ File

1010 Y = SIN(X) + SIN(X*3)/3 + SIN(X*5)/5
RUN 170
Y = SIN(X) + SIN(X*3)/3 + SIN(X*5)/5

Lines 390-430 determine the Y range and the displacement of the X-axis.

In *lines 440-450,* because mixed-screen high-resolution graphics are being used, the Y values are translated using 159 instead of 191.

Line 460 contains VTAB 22, which positions the cursor in the text window below the graphics display.

In *lines 470-480,* the equation is centered under the graph.

Lines 500-510 draw the axis lines.

Lines 520-550 establish the starting point of the curve. Note that dividing by YI or XI converts from actual equation values to screen values.

Lines 560-600 draw the rest of the graph. See Auto Graph examples.

In *lines 610-750,* black dots are plotted on the axis lines to indicate the scale.

Line 620 uses the subroutine at line 770 to calculate a one-digit value (DI) for marking the axis divisions.

Lines 640-650 assure that the entire axis will be marked.

In *lines 2500-2640,* the error handling routine has been expanded to check the line number where the error occurred (line 2510), and the type of error (line 2560).

Eugene J. Rolfe is a Senior Engineer at the Bendix Kansas City Division. He has an MSEE from the University of Missouri-Columbia. Mr. Rolfe has been teaching minicomputer and microcomputer classes at Longview Community College for several years, and has just started teaching courses at a local microcomputer center called Compu-Aide. You may write Mr. Rolfe at 9009 Evanston Ct., Kansas City, MO 64138.

MICRO

Screen Editor for the OSI 65D Assembler

by Les Cain

Add this powerful screen editor to your OSI 65D assembler and make your programming easier.

Screen Editor
requires:

OSI
with OS65D operating system
and 64-character screen

As personal computing increases in popularity, interest in the machines themselves also increases. BASIC and other high-level languages are more or less portable between different brands of computers and processors, but afford little interaction with the actual workings of the particular computer itself. Machine code, the bare bones effort, turns off all but the hardiest of programmers. The next level is the assembler, which allows direct control of the processor in a friendlier atmosphere.

I am an avid computer user and have started using the assembler. OSI has a fine assembler for their computer; it is easy to use and fairly fast. This article describes a screen-based text editor added to the assembler, which makes changing source files easier and more convenient. As it stands, you have to retype the entire line to make changes. This becomes a nuisance if the changes are minor, such as changing an operand or correcting a typing error. Making changes on the screen among the other lines of source code also allows visual verification as corrections are made. While good programming usually involves commenting as code is written, at times this is not done until later. With the screen editor, adding comments is made simpler by not requiring the entire line to be retyped.

Details of the actual program will not be discussed, as it is internally documented. The implementation and use of the screen editor will be explained.

Type in the program as listed, leaving out comments if desired. Check lines as they are entered; do not rely on the assembler to find your typos. Save often, as nothing is more frustrating than typing for hours and then losing everything due to a power glitch or an inadvertent command.

After everything is in and reasonably correct, do an A1 assembly and correct all errors. Be sure origin value is correct, then do an A3 assembly. At this time the code is assembled into empty space at the top of the assembler and below the extended monitor. Now save this back to disk with a !SA 09,1=1200/5 (5" disk). Even if the code is not one-hundred-percent correct, nothing is lost yet.

Listing 1

```
 10                 ; SCREEN EDITOR FOR THE OSI ASSEMBLER
 20                 ; FOR OS65D3 OPERATING SYSTEMS WITH 64 CHR. SCREEN
 30                 ;
 40                 ; CODE BY LES CAIN
 50                 ; 4625 N. 78TH AV.
 60                 ; PHOENIX AZ   85033
 70                 ;
 80                 ;
 90                 ; ASSEMBLE AT $1580 THEN SAVE (!SA 09,1=1200/5)
100                 ; CHANGE $020C TO JMP $1580 SAVE (!SA 07,1=0200/8)
110                 ; CHANGE $2617 ($2635 FOR 1P) TO $60
120                 ; CHANGE TRACK 0 TO REFLECT CHANGE
125                 ;
130 1580                   *=$1580
140                 ;
150                 ; EXTERNAL ADDRESSES
160                 ;
170 020C=                  GETCH=$020C
180 020F=                  BAKAS=$020F
190 0588=                  INPUT=$0588
200 252B=                  KEYIN=$252B
210 006C=                  SCRLO=$006C
220 006D=                  SCRHI=$006D
230 008D=                  YHOLD=$008D
240 008E=                   TEMP=$008E
250 0002=                  BUFST=$0002
260                 ;
270                 ; JUMP HERE WITH PATCH IN ASSEMBLERS INPUT ROUTINE
280                 ;
290 1580 208805                JSR      INPUT    ; GETS CHAR. FROM ASSEMBLER
300 1583 C90F           CMP    #$0F              ; CONTROL O TURN ON EDITOR
310 1585 F003           BEQ    EDIT              ; YES DO EDIT ROUTINES
320 1587 4C0F02                JMP      BAKAS    ; BACK TO ASSEMBLER
330                 ;
340                 ; MAIN PART OF PROGRAM
350                 ;
360 158A A000  EDIT     LDY    #$00
370 158C 848D           STY    YHOLD             ; INITIALIZE Y
380 158E 846C           STY    SCRLO             ; SAME FOR SCREEN LO BYTE
390 1590 A9D7           LDA    #$D7              ; SCREEN HI BYTE
400 1592 856D           STA    SCRHI
410 1594 A48D  EDITA    LDY    YHOLD             ; GET OFFSET FROM SIDE OF SCRE.
420 1596 B16C           LDA    (SCRLO),Y         ; SAVE CHAR. WHERE CURSOR GOES
430 1598 858E           STA    TEMP              ; SAVE IT
440 159A A9A1           LDA    #$A1              ; BLOCK CURSOR
450 159C 916C           STA    (SCRLO),Y         ; ALWAYS START AT $D700
460                 ;
470                 ; GET KEY ROUTINE
480                 ;
490 159E 202B25         JSR    KEYIN             ; DOS KEYBOARD ROUTINE
500 15A1 48             PHA                      ; SAVE NEW BYTE
510 15A2 A48D           LDY    YHOLD             ; GET CURSOR OFFSET
520 15A4 A58E           LDA    TEMP              ; GET COVERED BYTE
530 15A6 916C           STA    (SCRLO),Y         ; PUT IT BACK
540 15A8 68             PLA                      ; RETREIVE NEW BYTE
550 15A9 A48D           LDY    YHOLD             ; GET CURSOR OFFSET AGAIN
560 15AB D012           BNE    SIDE              ; NOT AT SIDE OF SCREEN
570 15AD C904           CMP    #$04              ; CONTROL D
580 15AF D005           BNE    STCMP             ; NO
590 15B1 48             PHA                      ; SAVE IT
600 15B2 207616         JSR    CTRLD             ; MOVE CURSOR DOWN
610 15B5 68             PLA                      ; RETREIVE THE BYTE
620 15B6 C915  STCMP    CMP    #$15              ; CONTROL U
630 15B8 D005           BNE    SIDE
```

(Continued)

Next, a change is needed in the assembler itself in order for the assembler to jump to the new code when looking for an input. Re-enter the extended monitor and display address $020C. You should find:

Address	Present	Modified
$020C	#$20	#$4C
$020D	#$88	#$80
$020E	#$05	#$15

If the first location is not a $20, do a Q0200 and find a JSR $0588 and change to a JMP $1580. This bypasses the original input routine and goes through the EDITOR before storing input or acting on commands. Once this is changed, re-enter the assembler. If all the code assembled at $1580 is correct, input from the keyboard will function as normal. If so, save this change by doing a !SA 07,1 = 0200/8 (5'' disk).

One more hook is necessary before trying the editor. Re-enter the extended monitor and display the memory location $2617; it should be a $20, or possibly a $60. If it is a $60, then leave it alone and skip the rest of this paragraph. If not, change it to a $60. This is the blank that is written on the bottom line of the screen after a scroll. By changing it to a $60, an end point can be established for distinguishing a space from a blank in determining the end of a line of source code.

Re-enter the assembler and type a Control-O. This should display a square cursor one line above the assembler's period. If this happens, do a carriage return; if the $20 was changed to a $60, as described in the previous paragraph, make the change to Track 0 permanent. To do this, re-enter the extended monitor and call in the Track 0 copier. Read to location $4200, using the EM change location $4617 to a $60. Go back to the Track 0 copier and W4200/2200,8. This will make the changes permanent and the assembler editor is now ready to use. Reboot and get back into the assembler.

Using the newly installed screen editor, the following commands are available:

1. Control O: Turns on the editor and displays a square cursor in the bottom left-hand side of the screen.
2. Control U: Moves cursor up the left side of the screen.
3. Control D: Moves cursor down the left side of the screen.
4. Control R: Moves the cursor right.
5. Control L: Moves the cursor left.
6. Rubout: Deletes character to the left of the cursor.

Listing 1 *(Continued)*

```
640 15BA 48             PHA
650 15BB 205C16         JSR      CTRLU          ; MOVE CURSOR UP
660 15BE 68             PLA
670 15BF C90C  SIDE     CMP      #$0C           ; CONTROL L
680 15C1 D005           BNE      CTR            ; NO
690 15C3 48             PHA
700 15C4 201816         JSR      CTRLL          ; MOVE CURSOR LEFT
710 15C7 68             PLA
720 15C8 C912  CTR      CMP      #$12           ; CONTROL R
730 15CA D005           BNE      CRD            ; NO
740 15CC 48             PHA
750 15CD 200C16         JSR      CTRLR          ; MOVE CURSOR RIGHT
760 15D0 68             PLA
770 15D1 C90D  CRD      CMP      #$0D           ; CARRIAGE RETURN
780 15D3 F06D           BEQ      CR             ; YES
790 15D5 C97F           CMP      #$7F           ; RUBOUT
800 15D7 F047           BEQ      RUBOUT         ; DELETE CHARACTER TO LEFT
810 15D9 C908           CMP      #$0H           ; CONTROL H
820 15DB F059           BEQ      HOME           ; HOME CURSOR $D100
830 15DD C920           CMP      #$20           ; < THAN 32
840 15DF 90B3           BCC      EDITA          ; DISALLOW
850 15E1 C97D           CMP      #$7D           ; > 126
860 15E3 B0AF           BCS      EDITA          ; DISALLOW GRAPHICS
870                ;
880                ; STORE CHARACTER ON SCREEN
890                ;
900 15E5 A48D           LDY      YHOLD          ; CURSOR OFFSET
910 15E7 C038           CPY      #$38           ; LINE LENGTH
920 15E9 F018           BEQ      ST2            ; > BUFFER LENGTH
930 15EB E68D           INC      YHOLD          ; BUMP CURSOR ONE RIGHT
940 15ED 48    ST1      PHA                     ; SAVE NEW CHARACTER
950 15EE B16C           LDA      (SCRLO),Y      ; GET BYTE AT CURSOR POSITION
960 15F0 AA             TAX                     ; SAVE IT
970 15F1 68             PLA                     ; RETREIVE NEW BYTE
980 15F2 916C           STA      (SCRLO),Y      ; PUT IT ON THE SCREEN
990 15F4 C960           CMP      #$60           ; IS IT A END OF LINE BLANK
1000 15F6 F00B          BEQ      ST2            ; YES GO NO FURTHER
1010 15F8 8A            TXA                     ; GET PREVIOUS CHARACTER
1020 15F9 C8            INY                     ; BUMP CURSOR ONE RIGHT
1030 15FA C038          CPY      #$38           ; PAST BUFFER AGAIN
1040 15FC D0EF          BNE      ST1
1050 15FE 88            DEY                     ; YES MOVE BACK ONE
1060 15FF A960          LDA      #$60           ; END OF LINE BLANK
1070 1601 916C          STA      (SCRLO),Y      ; MARK END OF LINE
1080 1603 A48D ST2      LDY      YHOLD          ; PREVIOUS CURSOR POSITION
1090 1605 A58E          LDA      TEMP           ; SAVED SCREEN CHARACTER
1100 1607 916C          STA      (SCRLO),Y      ; DISPLAY IT
1110 1609 18   ST3      CLC
1120 160A 9088          BCC      EDITA          ; JUMP
1130               ;
1140               ; CONTROL R TO MOVE CURSOR RIGHT
1150               ;
1160 160C A58E CTRLR    LDA      TEMP           ; GET COVERED BYTE BACK
1170 160E C960          CMP      #$60           ; BLANK
1180 1610 F005          BEQ      CTR2           ; CAN NOT GO ANY FURTHER
1190 1612 A48D          LDY      YHOLD          ; NO MOVE CURSOR ONE RIGHT
1200 1614 C8            INY
1210 1615 848D CTR1     STY      YHOLD          ; SAVE CURSOR POSITION
1220 1617 60   CTR2     RTS
1230               ;
1240               ; CONTROL L MOVE CURSOR LEFT
1250               ;
1260 1618 A48D CTRLL    LDY      YHOLD          ; CURSOR POSITION
1270 161A F0F9          BEQ      CTR1           ; LEFT EDGE OF SCREEN
1280 161C 88            DEY                     ; MOVE ONE LEFT
1290 161D 18            CLC
1300 161E 90F5          BCC      CTR1           ; SAVE CURSOR AND RETURN
1310               ;
1320               ; RUBOUT DELETES CHARACTER TO LEFT OF CURSOR
1330               ;
1340 1620 A48D RUBOUT   LDY      YHOLD          ; GET CURSOR POSITION
1350 1622 F00F          BEQ      RUB3           ; CANNOT DELETE ANYTHING
1360 1624 B16C RUB1     LDA      (SCRLO),Y      ; GET BYTE TO THE RIGHT
1370 1626 88            DEY
1380 1627 916C          STA      (SCRLO),Y      ; STORE IT ONE LEFT
1390 1629 C960          CMP      #$60           ; END BLANK
1400 162B F004          BEQ      RUB2           ; LINE UPDATE COMPLETE
1410 162D C8            INY                     ; CURSOR RIGHT 2
1420 162E C8            INY
1430 162F D0F3          BNE      RUB1           ; BRANCH ALWAYS
1440 1631 C68D RUB2     DEC      YHOLD          ; BACK ONE
1450 1633 18   RUB3     CLC
1460 1634 90D3          BCC      ST3            ; JUMP
1470               ;
1480               ; CONTROL H HOME CURSOR TO $D100
1490               ;
1500 1636 A900 HOME     LDA      #$00           ; PUT CURSOR A TOP OF SCREEN
1510 1638 856C          STA      SCRLO
1520 163A 858D          STA      YHOLD
1530 163C A9D1          LDA      #$D1
1540 163E 856D          STA      SCRHI
1550 1640 D0C7          BNE      ST3            ; JUMP
1560               ;
1570               ; CARRIAGE RETURN, TIME TO PUT EDITED LINE BACK
1580               ; INTO THE SOURCE CODE
1590               ;
```

Listing 1 *(Continued)*

```
1642 A000   CR     LDY   #$00
1644 B16C   CR1    LDA   (SCRLO),Y     ; LEFT MOST BYTE
1646 C960          CMP   #$60          ; END OF LINE MARKER
1648 F007          BEQ   CR2           ; YES THRU WITH THE LINE
164A 9102          STA   (BUFST),Y     ; STORE IN ASSEMBLER BUFFER
164C C8            INY
164D C038          CPY   #$38          ; TOO LONG TO FIT
164F D0F3          BNE   CR1           ; NO CONTINUE
1651 98     CR2    TYA                 ; CHECK FOR SIDE OF SCREEN
1652 D003          BNE   RETURN        ; NO PUT LINE IN BUFFER
1654 4C0C02        JMP   GETCH         ; YES NO EDIT IS NEEDED
            ;
            ; BACK TO ASSEMBLER WITH EDITED LINES PUT IN BUFFER
            ; REQUIRES A CR TO INTIATE READJUSTMENT
            ;
1657 A90D   RETURN LDA   #$0D          ; CARRIAGE RETURN
1659 4C0F02        JMP   BAKAS         ; GO DO ADJUSTMENTS
            ;
            ; CONTROL U TO MOVE CURSOR UP
            ;
165C A56C   CTRLU  LDA   SCRLO
165E 38            SEC
165F               SBC   #$40          ; UP ONE LINE
1661 AA            TAX                 ; SAVE RESULT
1662 A56D          LDA   SCRHI         ; HI BYTE
1664 B007          BCS   CTL1          ; NO CARRY REQUIRED
1666 B007          CMP   #$D1          ; TOP OF SCREEN
1668 F008          BEQ   CTL2          ; AS FAR AS WE CAN GO
166A 38            SEC
166B E901          SBC   #$01          ; SUBTRACT 1 FROM HI BYTE
166D 856D   CTL1   STA   SCRHI
166F 866C          STX   SCRLO         ; LO BYTE OF SCREEN ADDRESS
1671 60            RTS
1672 A200   CTL2   LDX   #$00          ; MAKE SCREEN ADDRESS $D100
1674 F0F7          BEQ   CTL1          ; JUMP
            ;
            ; CONTROL D MOVE CURSOR DOWN
            ;
1676 A56C   CTRLD  LDA   SCRLO
1678 18            CLC
1679 6940          ADC   #$40          ; DOWN ONE LINE
167B AA            TAX                 ; SAVE RESULT
167C A56D          LDA   SCRHI
167E 6900          ADC   #$00          ; INC HI BYTE IF CARRY SET
1680 C9D7          CMP   #$D7          ; BOTTOM OF SCREEN
1682 D0E9          BNE   CTL1          ; ADD ONE TO SCREEN HI
1684 A200          LDX   #$00          ; YES MAKE SCR. ADDR. $D700
1686 F0E5          BEQ   CTL1          ; JUMP
```

7. CR: Quits the editor and inserts the entire line into the source file.

After the cursor is displayed in the proper place, remember to hold the control key down while moving the cursor. Any character allowed by the assembler may be inserted at the cursor position, except the control keys. To terminate editing, enter a carriage return after each entry or correction. The entire line will be entered even if the cursor is not at the end of the line.

A few warnings are due here to prevent unnecessary errors. If a line that has been entered without the screen editor needs correcting immediately, delete the period immediately to the left of the line of source code, or a CMD error will result. The screen editor puts what it sees into the source code. This can create errors if correcting a line from an A assembly-created text line. These lines have object code in the displayed lines and will be put into the input buffer along with the original source code. Therefore, print the lines which need editing.

This procedure will set the screen editor working. After a few sessions using the editor, you will wonder how you survived without it.

Lester Cain may be contacted at 4625 N. 78th Ave., Phoenix, AZ 85033.

MICRO

Building a Parallel Printer Interface

by Rolf B. Johannesen

Guiding principles for the design of a parallel output interface are discussed, and a circuit is given for interfacing a particular computer-printer combination.

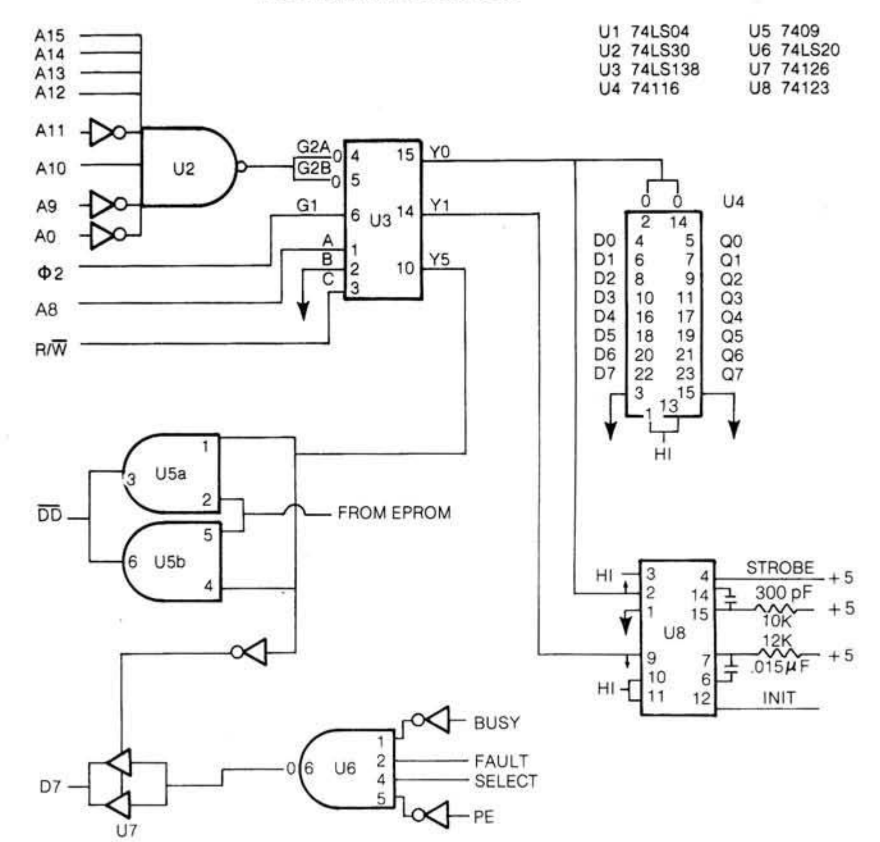
PRINT SUBROUTINE
requires:
OSI C1P
C. Itoh 8510 printer
(Information given to assist with other combinations)

A parallel interface offers a fast, convenient, and relatively inexpensive method of connecting a printer to a microcomputer. Because 8 bits at a time are sent, and there is no coding/decoding to be done at the ends of the line, parallel data transmission is usually much faster than serial. The major advantage of serial (RS-232, for example) transmission is that relatively long lines may be used — some hundreds of feet; whereas parallel lines are generally limited to around ten feet. In a small hobby system it is ordinarily no handicap to require that the printer be nearby. If your computer has a 6522 VIA, or a similar chip providing strobed output, then it is not difficult to connect a printer to the computer (MICRO 44:45). However, many computers that do not offer a VIA output have an expansion connector that provides all the necessary signals to drive the printer through a suitable interface circuit.

The interface described in this article was built specifically to connect an OSI C1P computer to a C. Itoh Model 8510 Dot Matrix Printer. However, the principles involved are generally valid and this interface should be readily adaptable to any computer-printer pair. Except for choice of address, the computer side of the interface should need only minimal alterations for any 6502-based machine. The printer side may be subject to greater variation depending on the particular signals required and furnished by the printer. There are 20 leads used at the computer side — nine address lines (the address is not "fully decoded"), eight data lines, phase 2 of the clock, and the read/write line. The interrupt line is not used in this interface, but could be included if desired. On the printer side are 14 lines — eight data lines, intitialize, strobe, busy, fault, select, and paper empty. An acknowledge signal is returned by the printer but is not used. It could be used to flag an interrupt line if desired, or to clear a flip-flop (preset by the strobe signal) to provide the equivalent of BUSY if that is not provided by the printer.

The 6502 does not have any specific I/O commands, so external communication is handled by memory-mapping. That is, a particular address is chosen as the I/O port. A store command to the chosen address sends a signal out; a load command accepts a signal from the port. The address selected cannot already be in use for any other purpose; it should also avoid any likely further expansion of the machine. The addresses chosen for this interface are \$F400 and \$F401. The upper part of the address is chosen by sending address lines A8-A15 to U2, an 8-input NAND gate. Lines A8, A9, and A11 are inverted in U1 so that the output of the gate is low only when \$F4xx is seen on the address bus. Line A0 provides further address selection. The address lines are not fully decoded since lines A1-A7 are not sampled. Thus, any even address between \$F400 and \$F4FE will elicit the same response as \$F400. There is enough address space available in the C1P so that this multiplicity of

Parallel Printer Interface



addressing is no handicap; a second 8-input NAND gate could, of course, be added if it were desired to obtain a fully decoded address. The address decoding is completed in U3, which receives the output of U2 at its G2A and G2B inputs, phase 2 at its G1 input; and A0, ground and R/$\overline{W}$ at its A, B, and C inputs. Only three of the four possible outputs from U3 are needed. Y0 is activated when A0 and R/$\overline{W}$ are both low; it causes a data word to be sent to the printer. Y1 is activated when A0 is high and R/$\overline{W}$ is low; it causes an initialize signal to be sent to the printer. Y5 is activated when A0 is high and R/$\overline{W}$ is high; it causes the printer status to be returned to the computer.

Let's examine each of these signals in more detail to see what they do and what changes may be needed in other systems. Y0 (STA $F000) activates latch U4. When Y0 is selected (low), the output of the latch follows the data lines. When phase 2 of the clock changes, Y0 goes high and the outputs of U4 will no longer change: they are latched. These are the data lines to the printer at TTL levels. If the printer requires another voltage than TTL, then the appropriate level-changing circuitry must be incorporated. In case the printer accepts only 7-bit ASCII code, the high-order line (Q7) should simply be left open. Line Y0 has a second essential function. It triggers the one-shot on half of U8 to send out a strobe signal that tells the printer to accept the signals on the data lines from the latch. The values of R and C shown here provide a strobe pulse of about 1.5 μs. The requirements for individual printers may vary; as long as pulse width is greater than the required minimum, its exact length is not critical. In case your printer requires a positive strobe, rather than negative, simply use the Q output from U8 rather than Q. Next, Y1 (STA $F401) generates an initialize signal to send to the printer. This returns the printer to the power-on condition and all options assume their default values. (Some printers may require an initialize signal before they will accept any other data.) Notice in this case that the contents of the data bus are immaterial; memory-mapped I/O uses the data lines only if the interface is built to use them. The other half of U8 provides the initialize signal. The same remarks as to timing and polarity of the strobe line apply here as well. In this instance, R and C give an initialize pulse of about 64 μs. Finally, Y5 (STA $401) returns the printer status to the computer.

The circuitry here is strongly dependent on both printer and computer and thus subject to more variation than in other parts of the interface. Four printer lines are combined in gate U6: NOT .BUSY, FAULT, NOT.PE and SELECT. The NOT functions are obtained from the corresponding true signals by inverting them in U1. When all of these signals are high, the printer is ready to receive data and the output of U6 is low. The output of U6 is passed through U7, a tri-state buffer, to data line D7. The controlling gate of U7 is activated by the inverted Y5 signal, so it passes the signal from U6 to the data line only when Y5 is active (low). At the same time, Y5 activates the $\overline{DD}$ (data direction) line in the C1P. The $\overline{DD}$ line is provided at the expansion connector of the C1P for the use of peripheral devices. In its normal state it is held high by a pull-up resistor and the data bus contains signals output from the computer.

When a peripheral device wants to send data to the computer, it must pull $\overline{DD}$ low so that the data lines will be treated as input by the computer. This is done by Y5, acting through the open collector gate U5. U5 combines the signal from Y5 with that from another peripheral device (an EPROM) so that

Listing 1

```
0000          ;              PRINT SUBROUTINE
0000          ;              DEFINITIONS
0000          ;
0000          SFLAG  = $0205      SAVE FLAG
0000          SCREEN = $BF2D      DISPLAY ROUTINE
0000          PROUT  = $F400      PRINTER PORT
0000          PSTAT  = $F401      PRINTER STATUS
0000          ;              PROGRAM STARTS HERE
3FEB                     *=$3FEB
3FEB 48       PRINT PHA           SAVE AC ON STACK
3FEC 202DBF     JSR SCREEN        DISPLAY CHARACTER
3FEF AD0502     LDA SFLAG         TEST SAVE FLAG
3FF2 F00A       BEQ XIT           RETURN IF FLAG=0
3FF4 AD01F4   FTEST LDA PSTAT     TEST PRINTER STATUS
3FF7 30FB       BMI FTEST         REPEAT IF BUSY
3FF9 68         PLA               GET SAVED AC FROM STACK
3FFA 8D00F4     STA PROUT         SEND TO PRINTER
3FFD 48         PHA               BALANCE PUSHES
3FFE 68       XIT PLA             AND PULLS
3FFF 60         RTS               RETURN TO CALLING PROGRAM
```

$\overline{DD}$ can be pulled low by either device. If only one peripheral device is used, U5 can be replaced by a pair of 7417 open collector drivers in parallel. It may not be necessary in all cases to combine four signals from the printer to provide a busy signal to the computer. Some printers will have only one BUSY signal, which will be adequate, and in this case U6 may be dispensed with. However, a tri-state buffer must always be used at U7 to gate data onto the bus in order for the computer to work properly when it is not trying to read the printer status. The printer status is returned to the computer on line D7 (the MSB) so that it can be tested by a BMI or BPL command without requiring any masking or shifting. If desired, each of the signals returned by the printer could be sent on separate data lines to the computer, gating each signal through a tri-state buffer with Y5 as the gating pulse. In this case the computer would have more information about the state of the printer, but it would also require more coding to interpret the several lines.

Construction Details

This circuit was built on a small piece of vectorboard using wire-wrap connections. The pinouts of the computer and the printer are not given since they vary from one machine to another. The total cost of the IC chips should be under $10; the cable connectors may cost more. Although the printer manual specifies multiple-twisted pair cable for the connecting wires, I was able to use flat ribbon cable successfully. The cable is laid out so that signal wires alternate with grounds to avoid cross-talk. The connections at computer and printer will require appropriate cable connectors. Connectors are not necessarily required on the interface board, but they are a great convenience. I have used the 40-pin wire-wrap connectors for this purpose. The expansion connector on the C1P (600 or 610 board) uses 39 of the 40 available pin positions for signals and grounds. I have used the remaining pin for power to the interface card, connecting it to the 5V bus in the computer.

Software

The listing given in the box is a simple routine for sending data to the printer (and screen) instead of to the cassette whenever a SAVE command is typed from BASIC or a JSR $FFEE is sent from an assembly program. It requires that $021A and $021B be set to the starting address of the print routine, either by the keyboard monitor, or by POKE commands. The addresses are specific to the C1P; similar routines are of course available in other machines.

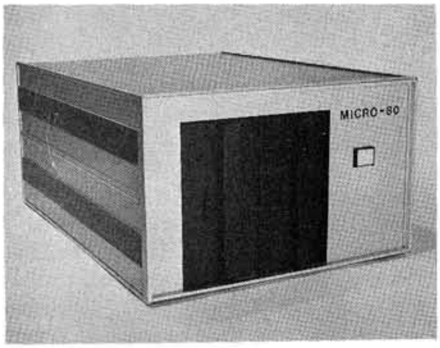
Summary

This article has attempted to discuss the principles of parallel interfacing in enough detail to let the reader adapt the circuit shown here to a particular system. The importance of understanding what signals are available from/required by both computer and printer cannot be overstated. Once this information is known, then design of an interface along the lines given here is quite straightforward.

The author may be contacted at 13917 Congress Drive, Rockville, Maryland 20853.

MICRO

NEW FROM D & N MICRO PRODUCTS, INC.



MICRO-80 COMPUTER

Z80A CPU with 4MHz clock and CP/M 2.2 operating system. 64K of low power static RAM. Calendar real time clock. Centronics type parallel printer interface. Serial interface for terminal communications, dip switch baud rates of 150 to 9600. 4″ cooling fan with air intake on back of computer and discharge through ventilation in the bottom. No holes on computer top or side for entry of foreign object. Two 8″ single or double sided floppy disk drives. IBM single density 3740 format for 243K of storage on each drive. Using double density with 1K sectors 608K of storage is available on a single sided drive or 1.2 meg on a double sided drive. Satin finish extruded aluminum with vinyl woodgrain decorative finish. 8 slot backplane for expansion. 48 pin buss is compatible with most OSI boards. Uses all standard IBM format CP/M software.

Model 80-1200	$2995
2 8″ single sided drives, 1.2 meg of storage	
Model 80-2400	$3495
2 8″ double sided drives, 2.4 meg of storage	
Option 001	$ 95
Serial printer port, dip switch baud rate settings	

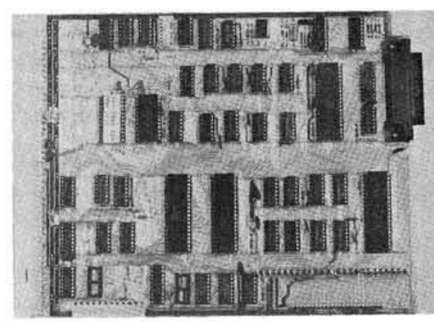
Software available in IBM single density 8″ format.

Microsoft	
Basic-80	$289
Basic Compiler	$329
Fortran-80	$410
Cobol-80	$574
Macro-80	$175
Edit-80	$105
Mu Simp/Mu Math	$224
Mu Lisp-80	$174
Digital Research	
PL/1-80	$459
Mac	$ 85
Sid	$ 78
Z-Sid	$ 95
C Basic-2	$110
Tex	$ 90
DeSpool	$ 50
Ashton-Tate	
dBase II	$595
Micropro	
Wordstar	$299
Mail-Merge	$109
Spellstar	$175
Super Sort I	$195
Pascal	
Pascal/MT +	$429
Pascal Z	$349
Pascal M	$355

Convert almost any static memory OSI machine to CP/M® with the D & N-80 CPU Board.

Z80A CPU with 4MHz clock. 2716 EPROM with monitor and bootstrap loader. RS-232 serial interface for terminal communications or use as a serial printer interface in a VIDEO system. Disk controller is an Intel 8272 chip to provide single or double density disk format. 243K single density or 608K double density of disk storage on a single sided 8″ drive. A double sided drive provides 1.2 meg of storage. DMA used with disk controller to unload CPU during block transfers from the disk drives. Optional Centronics type parallel printer port complete with 10 ft. cable. Optional Real Time Calendar Clock may be set or read using 'CALL' function in high level languages. Power requirements are only 5 volts at 1.4 amps. Available with WORDSTAR for serial terminal systems.

D & N-80	serial	$695
D & N-80	serial w/Wordstar	$795
D & N-80	video	$695
Option 001		$ 80
	parallel printer and real time calendar clock	



D & N-80 CPU BOARD

OTHER OSI COMPATIBLE HARDWARE

IO-CA10X Serial Printer Port $125
Compatible with OS-65U and OS-65D software

IO-CA9 Parallel Printer Port $175
Centronics standard parallel printer interface with 10 ft. flat cable

BP-580 8 Slot Backplane $ 47
Assembled 8 slot backplane for OSI 48 pin buss

24MEM-CM9	**$380**	**24MEM-CM9F**	**$530**
16MEM-CM9	**$300**	**16MEM-CM9F**	**$450**
8MEM-CM9	**$210**	**8MEM-CM9F**	**$360**
BMEM-CM9F	**$ 50**	**FL470**	**$180**

24K memory/floppy controller card supports up to 24K of 2114 memory chips and an OSI type floppy disk controller. Available fully assembled and tested with 8, 16, or 24K of memory, with floppy controller (F). Controller supports 2 drives. Needs separated clock and data inputs. Available Bare (BMEM-CM9F) or controller only (FL-470). Ideal way to upgrade cassette based system

C1P-EXP Expansion Interface $ 65
Expansion for C1P 600 or 610 board to the OSI 48 pin buss. Requires one slot in backplane. Use with BP-580 backplane

BIO-1600 Bare IO card $ 50
Supports 8K of memory, 2 16 bit parallel ports may be used as printer interfaces. 5 RS-232 serial ports, with manual and Molex connectors

DSK-SW Disk Switch $ 29
Extends life of drive and media. Shuts off minifloppy spindle motor when system is not accessing the drive. Complete KIT and manual

D & N Micro Products, Inc.
3684 N. Wells St.
Fort Wayne, Ind. 46808
(219) 485-6414

VISA MasterCard

TERMS $2.50 shipping, Foreign orders add 15%.
Indiana residents add 4% sales tax.

Disk Drives and Cables

8″ Shugart SA801 single sided **$395**
8″ Shugart SA851 double sided **$585**
FLC-6 6ft. cable from D & N or OSI controller to 8″ disk drive **$ 69**
5 1/4″ MPI B51 with cable, power supply and cabinet **$450**
FLC-5 1/4 8 ft. cable for connection to 5 1/4 drive and D & N or OSI controller, with data separator and disk switch **$ 75**

Okidata Microline Printers

ML 82A Dot Matrix Printer **$534**
120 CPS, 80/120 columns, 9.5″ paper width, friction or pin feed

ML 83A Same as 82A except **$895**
16″ paper width, 132/232 columns with tractor feed

ML 84 Same as 82A except 200 CPS, **$1152**
16″ paper width, 132/232 columns, 2K buffer, dot addressable graphics, with tractor feed

The IBM Selectric as an OSI System Printer

by Paul Krieger

A circuit to interface an IBM I/O Selectric printer to a parallel port is presented. Software, including a simple text editor, is included to drive the circuit with an OSI C1P.

Selectric Driver and Editor Routines
requires:
OSI C1P
with interface circuit, 700 series IBM Selectric

One day at the local flea market I stumbled across an IBM Selectric typewriter built into an early attempt at a word processor. I had seen articles (Pytlik and Flystra) on converting these machines to operate with microcomputers. Even though this machine needed some work the price was right so I bought it.

If you want to make BASIC program listings, you need custom, type-element. The ¼, ½, and underline should be replaced with the <, >, and up-arrow, respectively. This costs about $10/character.

If you plan to attempt minor repairs or preventative maintenance on the typewriter, you may find IBM manual part #2415737 useful. This book costs $10.80 and covers I/O typewriter models 735, 745, 775, and others of the 7xx series.

The components necessary to drive an IBM 7xx series typewriter are:

1. A parallel port.
2. A solenoid driver circuit.
3. Appropriate power supplies.
4. Software to type programs and text.

Hardware

The C1P does not come with a parallel port, so this accessory must be added. There are a number of ways to do this. For more information see MICRO 39:97, "Expanding the Superboard," by Jack McDonald, or MICRO 32:65, "Interfacing the 6522 VIA," by Marvin L. DeJong.

I chose to order a memory board with PIA from Aardvark Technical Services, and add the required components myself.

The solenoid driver circuit is shown in figure 1. This circuit decodes the output from the port and pulls the appropriate solenoids to print or space.

You can use most any construction technique for the circuit. I used point-to-point wiring, attaching the wires with solder. The circuit was built on a 5″ × 6″ piece of perf board and attached to the box with standoffs. I recommend sockets for the ICs as they are easily damaged by heat and static electricity. Use a 20- to 30-watt soldering iron and a fine electrical-type solder. Ersin Multicore #24s.w.g. SN60 works very well.

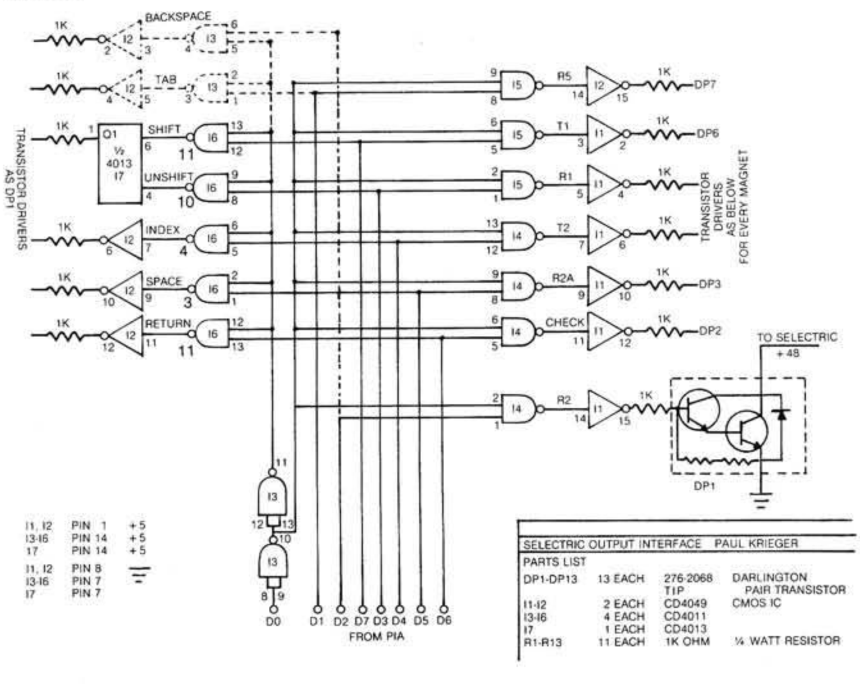
Two power supplies are needed to run the interface. Use 5 volts for the driver circuit, and 43 to 53 volts at 125 to 300 mA to pick the solenoids on the typewriter.

The 5 volts are easily obtained from the superboard power supply since CMOS requires very little power. Run a 24-gauge stranded wire from the black (ground) and from the red (plus 5).

The 48-volt DC power supply need not be regulated. It can be constricted using a 48-volt transformer, a full-wave rectifier, and a 100 μF, 100-volt electrolytic capacitor. The 120-volt side of the circuit should include a fuse and an on/off switch.

What are some of the problems you might run into? Depending on the model, you may find two solenoids instead of one to set up the shift. If you have two solenoids to shift, substitute two 4049 buffers for the 4013, and add an additional driver transistor. An alternative is to disable the lower case magnet armature so the lower case will not latch. The lower case solenoid is

Figure 1



SELECTRIC OUTPUT INTERFACE PAUL KRIEGER

PARTS LIST

DP1-DP13	13 EACH	276-2068 TIP	DARLINGTON PAIR TRANSISTOR
I1-I2	2 EACH	CD4049	CMOS IC
I3-I6	4 EACH	CD4011	
I7	1 EACH	CD4013	
R1-R13	11 EACH	1K OHM	¼ WATT RESISTOR

the one toward the rear of the machine.

With this circuit you must not hold these solenoids in the energized position for more than a minute or so. They are not the continuous duty type and will burn up. Line 56060 in the listing avoids this problem.

Another potential problem could be the connector that attaches the typewriter to the interface. My unit came with a 50-pin connector that seemed impossible to find at any price. To get around this obstacle, two Radio Shack molded nylon connectors (part #274-232) were connected inside the machine where there were similar connectors used for internal wiring. Later I found that the 50-pin is the same connector the telephone company uses for business phones, and is readily available at surplus stores, from IBM, and other sources.

Some of these machines were manufactured to use a binary-coded decimal code, instead of correspondence code. Another possibility is a different type ball than the one used for this article. In either case, you can use any code and any type ball by recalibrating the typeout translate table of lines 55542 to 55730.

The two sets of magnets are known as selection magnets, ones that tilt and rotate the type ball, and operational magnets, the solenoids that operate tab, space, backspace, carriage return, and index. The select magnets are known as T2, CK, T1, R2A, R1, R2, and R5.

If you stand the typewriter up on its back with the keyboard up and away from you, the select magnets are found in order on the left as listed previously. Operational magnets on the right are set up in a row of three: tab, bs, index. A row of two: space, cr.

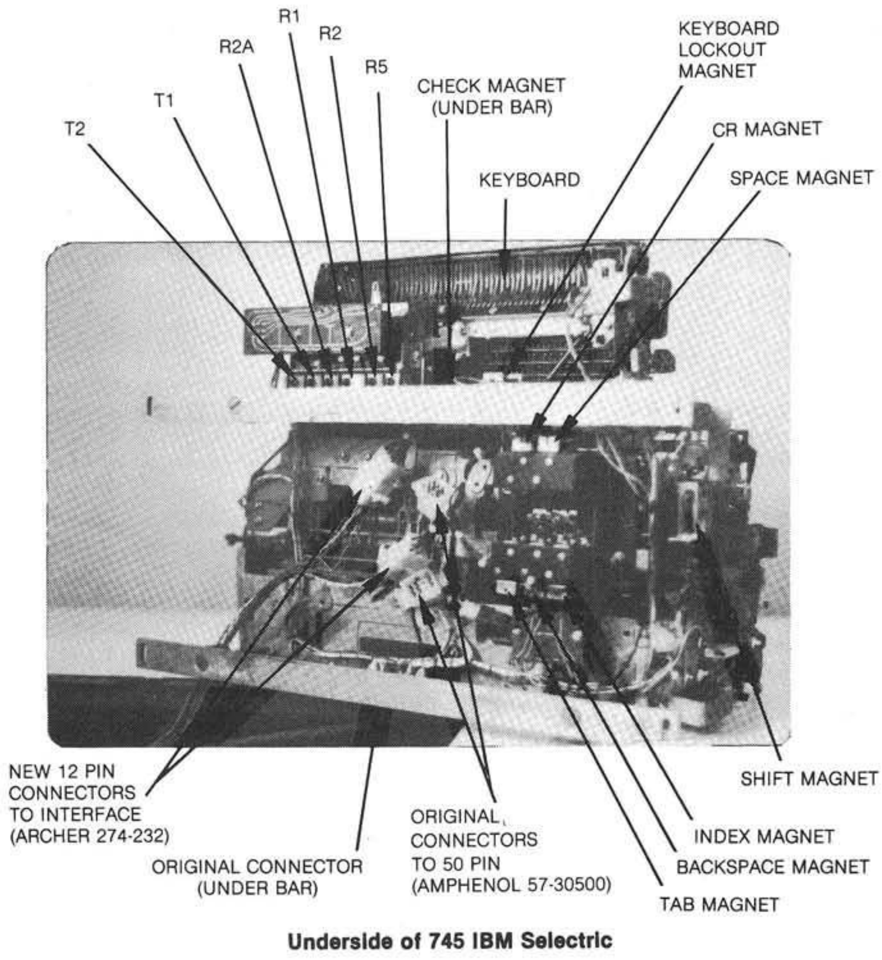
Figure 1 includes two optional circuits — backspace and tab. If you plan to underline you will need the backspace. In this case use another symbol as the up-arrow on the type ball ('¢' perhaps?).

There are two ways to provide synchronization with a typewriter. IBM recommends a closed loop method where timing signals from the typewriter are sensed by the computer to determine when to send the next character. This is certainly the fastest method. If you plan to use your typewriter every day for several hours it may be worth the extra trouble to hook up a feedback circuit.

One way to set this up is to drive the typewriter feedback contacts with a regulated 12-volt power supply (to meet minimum specifications on the contacts), clean up the input from the typewriter with a 4093 or 4584 Schmitt trigger at 12 volts, then convert the output of the trigger to 5 volts through a 10,000 ohm potentiometer. You would probably want three circuits into the other PIA port: ready, busy, and end-of-line.

To change the software for closed loop, replace the delays at 55960, 56000, 56020, 56040, 56046, 56080, and 56100 with WAIT set up for the port that you use.

In open loop mode the software must provide worst case delays for the mechanical action, so the typewriter will run slower. The delays of the above line numbers will need to be adjusted to the speed of your machine. For a 2MHz C1P, use 50, for a 1MHz, use 25. Other machines will be different. Use the smallest number that will print the character sent.



Underside of 745 IBM Selectric

Software

The plan was to write a quick little text editor that would accept lots of character-type information and eventually list it on the typewriter. I also wanted to avoid using a serial printer for listings. This turned out to be tough because you always run into garbage collect if you have enough string data. On the C1P this is a severe problem. Several of the aftermarket suppliers advertising in MICRO offer a replacement ROM #3 to fix this problem. Buy one: someday it will save you a lot of grief! Listing 2 works around the bug by using magnetic tape.

Another time-consuming problem was directing output to the parallel port. LIST with SAVE will get you a serial output — not much help going to a parallel port. The solution to this is the routine in listing 1. This routine pulls the BASIC program out of memory, then transforms the colon into a left bracket and the comma into a right bracket. The two together indicate end-of-tape.

Listing 1 should be keyed into memory first and then saved on cassette tape using LIST40000-40270. To test it, type RUN40000 and you will see it list itself.

To use this routine, first load in the program you wish to type out. Then load listing 1. Both programs are now in memory. If you encounter trouble loading the 40000 routine, try using three additional NULLs when you make its program tape.

This method creates library tapes of frequently used routines. You can load a few of them into memory to create new programs.

To make the print tape, type in SAVE, start your recorder, and type RUN40000. This will list both the routine and the subject program.

Routines list 2 and list 3 are loaded into memory after a cold start.

To read in a program tape for subsequent printing, start the recorder, type RUN, answer the run type message with a 2, and type in *I000-200 or so at the 0*? message. The program to be printed is read into memory until either the end of tape is read or the high range is reached. At that point you initialize the port with a *L000, turn on the typewriter then the driver circuit, and enter *L000-200. The translate at line 55880 will get the appropriate Selectric character and send it out to the typewriter.

If you run out of memory you can print the progam in pieces. Use *I000-090, stop the recorder when you reach 90, and list the program. Then restart with *I000-090 from the current position on the tape. If you reposition the tape, you can list only lines not printed by first entering *P000-020 to see where new lines start and then using *L0nn-nnn as needed. This is equally true of text of any kind. In text mode (reply 1 on startup) use the semicolon as a comma and the plus sign as a

Listing 1

```
40000 REM ROUTINE TO COPY PROGRAM TO TAPE W COLON=$7B COMMA=$7C
40010 DIMIT$(67)
40020 IT$(0)="END":IT$(1)="FOR":IT$(2)="NEXT":IT$(3)="DATA"
40022 IT$(4)="INPUT":IT$(5)="DIM":IT$(6)="READ":IT$(7)="LET"
40024 IT$(8)="GOTO":IT$(9)="RUN":IT$(10)="IF":IT$(11)="RESTORE"
40026 IT$(12)="GOSUB":IT$(13)="RETURN":IT$(14)="REM":IT$(15)="STOP"
40028 IT$(16)="ON":IT$(17)="NULL":IT$(18)="WAIT":IT$(19)="LOAD"
40030 IT$(20)="SAVE":IT$(21)="DEF":IT$(22)="POKE":IT$(23)="PRINT"
40032 IT$(24)="CONT":IT$(25)="LIST":IT$(26)="CLEAR":IT$(27)="NEW"
40034 IT$(28)="TAB":IT$(29)="TO":IT$(30)="FN":IT$(31)="SPC("
40036 IT$(32)="THEN":IT$(33)="NOT":IT$(34)="STEP":IT$(35)="+"
40038 IT$(36)="-":IT$(37)="*":IT$(38)="/":IT$(39)="*"
40040 IT$(40)="AND":IT$(41)="OR":IT$(42)=">":IT$(43)="="
40042 IT$(44)="<":IT$(45)="SGN":IT$(46)="INT":IT$(47)="ABS"
40044 IT$(48)="USR":IT$(49)="FRE":IT$(50)="POS":IT$(51)="SQR"
40046 IT$(52)="RND":IT$(53)="LOG":IT$(54)="EXP":IT$(55)="COS"
40048 IT$(56)="SIN":IT$(57)="TAN":IT$(58)="ATN":IT$(59)="PEEK"
40050 IT$(60)="LEN":IT$(61)="STR$":IT$(62)="VAL":IT$(63)="ASC"
40052 IT$(64)="CHR$":IT$(65)="LEFT$":IT$(66)="RIGHT$"
40054 IT$(67)="MID$"
40100 X=769
40110 NA=PEEK(X+1):REM GET NEXT INST HIGH BYTE
40120 NA=NA*256:REM SETUP HIGH
40130 NB=PEEK(X):NA=NA+NB:REM ADD IN LOW
40140 REM NA IS NEXT ADDRESS
40150 IFNA=0THENPRINTCHR$(123);CHR$(124):END:REM END LIST MARK
40160 Y=X+3:LN=PEEK(Y):LN=LN*256:REM HIGH OF LINE#
40170 Y=X+2:LO=PEEK(Y):LN=LN+LO:PRINTLN;:REM PRINT LINE #
40180 W=X+4
40190 FORI=0TO72
40200 C=PEEK(W+I)
40210 IFC=0THENX=NA:I=73:GOTO40250
40220 IFC<128ORC>195THEN40234
40230 C=C-128:PRINTIT$(C);:GOTO40250
40234 IFC=58THENC=123:REM COLON TRANSLATE
40236 IFC=44THENC=124:REM COMMA TRANSLATE
40240 PRINTCHR$(C);
40250 NEXTI
40260 PRINT
40270 GOTO40110
```

colon This will avoid the EXTRA IGNORED when the line is read from the keyboard or tape, yet it will still type correctly because of the translate.

Nine functions are provided to edit text line by line. A function is a directive to the program on any line indicated by a trigger character, an asterisk, followed by the command character and a range of line numbers or a single line number. You can use a single number in any appropriate command except create, where the range is mandatory. Some examples are shown in table 1.

References

1. Pytlik, William F., "An Inexpensive Word Processor," MICRO 36:65.
2. Flystra, Dan, "Interfacing the IBM Selectric Keyboard Printer," *BYTE*, Vol. 2, No. 6, June 1977, pg. 46.

Contact the author at 3268 S. Cathay Cr., Aurora, CO 80013.

Listing 2

```
1 REM SELECTRIC TYPEWRITER DRIVER/TEXT EDITOR
2 REM BY PAUL KRIEGER, Sept. 15, 1981
5 FORX=1TO10:PRINT:NEXT
9 DIMBF$(497)
10 PRINT"TYPEWRITER DRIVER/TEXT EDITOR"
11 PRINT"    VER 1.0  Sept. 15, 1981"
15 PRINT:PRINT"COMMANDS:"
20 PRINT"*CNNN-NNN 'CREATE' NEW LINES AT NNN-NNN"
30 PRINT"*D        'DELETE' CURRENT LINE"
40 PRINT"*E        'END' RETURN TO BASIC
60 PRINT"*H        'HELP' DISPLAY COMMANDS"
70 PRINT"*INNN-NNN 'INPUT' RANGE FROM TAPE'
80 PRINT"*LNNN-NNN 'LIST' TEXT ON SELECTRIC"
85 PRINT"*ONNN-NNN 'OUTPUT' TO TAPE"
90 PRINT"*PNNN-NNN 'PRINT' TEXT ON TV"
94 PRINT"*SNNN     'SET' LINE COUNTER TO NNN"
95 FORX=1TO5:PRINT:NEXT
96 INPUT"ENTER RUN TYPE; 1=EDIT KEYBOARD, 2=EDIT TAPE";RT
108 FORX=1TO5:PRINT:NEXT
110 P=0
120 PRINTP;:INPUT"*";L$
130 IFLEFT$(L$,1)="*"THEN160
135 BF$(P)=L$
136 P=P+1
137 IFG<PTHENG=P:REM KEEP LARGEST LINE NUMBER FOR DEL
150 GOTO120
160 IFMID$(L$,2,1)="L"THEN55500:REM LIST ON SELECTRIC
170 IFMID$(L$,2,1)="P"THEN1000:REM PRINT ON TV
180 IFMID$(L$,2,1)="S"THEN2000:REM SET TO LINE
190 IFMID$(L$,2,1)="E"THENEND
200 IFMID$(L$,2,1)="D"THEN3000:REM DELETE LINE
210 IFMID$(L$,2,1)="C"THEN4000:REM CREATE NEW LINES
220 IFMID$(L$,2,1)="H"THEN10:REM SHOW COMMANDS
230 IFMID$(L$,2,1)="I"THEN5000:REM INPUT TEXT FROM TAPE
240 IFMID$(L$,2,1)="O"THEN6000:REM OUTPUT TEXT
300 PRINT"UNKNOWN COMMND":GOTO120
1000 GOSUB9000:REM GET VALUE OF RANGE
1010 PRINTQ;
1020 PRINTBF$(Q):Q=Q+1
1030 IFQ<RTHEN1010
1040 GOTO120
2000 GOSUB9000
2020 IFQ<0ORQ>500THEN300
2030 P=Q
2040 GOTO120
3000 REM DELETE LINE ROUTINE
3010 Q=P
3020 BF$(Q)=BF$(Q+1):REM SHIFT NEXT LINE DOWN 1
3030 Q=Q+1
3040 IFQ<GTHEN3020
3050 GOTO120
4000 GOSUB9000:REM CREATE SPACE ROUTINE
4010 H=R-Q
4015 H=H+1
4020 P=P+H
4040 FORDS=PTOR+1STEP-1
4050 BF$(DS)=BF$(DS-H)
4060 NEXTDS
4070 FORDS=QTOR
4080 BF$(DS)=" "
4090 NEXTDS
4095 GOTO120
5000 GOSUB9000:REM INPUT FROM TAPE SUBROUTINE
5004 C$=CHR$(123):C$=C$+CHR$(124):REM CREATE END FILE MARKER
5008 LOAD
5010 FORP=QTOR
5015 PRINTP;
5030 INPUTBF$(P)
5040 IFLEFT$(BF$(P),2)=C$THENP=R:REM SET END OF INPUT
5050 NEXTP
5055 POKE515,0
5060 GOTO120
6000 GOSUB9000:REM OUTPUT TO TAPE SUBROUTINE
6005 REM SET NULL 10 IF 1 MHZ MACHINE
6010 SAVE
6020 FORP=QTOR
6030 PRINTBF$(P)
6040 NEXTP
6050 POKE517,0:REM TURN OFF SAVE
6060 GOTO120
9000 Q$=MID$(L$,3,3):R$=MID$(L$,7,3):REM GET RANGE
9010 Q=VAL(Q$):R=VAL(R$):RETURN
```

Table 1

Print lines 40 to 50.	110 *? *P040-050
Print line 12.	110 *? *P12
List lines 3 to 35 on the Selectric typewriter.	110 *? *L003-035
Delete line 44.	110 *? *S44 44 *? *D 44 *? *S115
Enter text on any line.	115 *? This text entered in. 116 *?
Create 5 empty lines at 2 to 6.	116 *? *C002-006
Go to line 2 and type in new text.	116 *? *S2 2 *? New text on line 2.
Go back to line 116.	2 *? *S116
Clear memory and start anew.	116 *? *E OK RUN

Listing 3

```
I.B.M. SELECTRIC DRIVER BY PAUL KRIEGER 9-12-81
55500 IFF1>0THEN55800
55510 PS=57345:PT=57344
55520 POKEPS,0:POKEPT,255
55530 POKEPS,4:POKEPT,0
55540 DIMTT(94)
55542 TT(0)=65:REM SPACE
55543 REM !"#$%&'()+./ NEXT 13 TABLE ITEMS
55544 TT(1)=255
55546 TT(2)=229
55548 TT(3)=251
55550 TT(4)=215
55552 TT(5)=245
55554 TT(6)=247
55556 TT(7)=228
55558 TT(8)=209
55560 TT(9)=213
55562 TT(10)=243
55563 IFRT=2THENTT(11)=105:REM +=+
55564 IFRT=1THENTT(11)=103:REM +=:
55565 TT(12)=232
55568 TT(13)=64
55570 TT(14)=232
55572 TT(15)=70
55573 REM NUMBERS 0 TO 9 NEXT 10 ITEMS
55574 TT(16)=212
55576 TT(17)=254
55578 TT(18)=248
55580 TT(19)=250
55582 TT(20)=214
55584 TT(21)=244
55586 TT(22)=240
55588 TT(23)=246
55590 TT(24)=242
55592 TT(25)=208
55593 REM :;<=>? COMMERCIAL AT NEXT 7 ITEMS
55594 TT(26)=103
55595 IFRT=2THENTT(27)=102:REM ;=;
55596 IFRT=1THENTT(27)=98:REM ;=,
55598 TT(28)=236
55600 TT(29)=104
55602 TT(30)=237
55604 TT(31)=71
55606 TT(32)=249
55607 REM UPPER CASE A TO Z NEXT 26
55608 TT(33)=227
55610 TT(34)=81
55612 TT(35)=115
55614 TT(36)=119
55616 TT(37)=117
55618 TT(38)=107
55620 TT(39)=111
55622 TT(40)=85
55624 TT(41)=225
55626 TT(42)=109
55628 TT(43)=113
55630 TT(44)=87
55632 TT(45)=239
55634 TT(46)=121
55636 TT(47)=199
55638 TT(48)=101
55640 TT(49)=97
55642 TT(50)=231
```

Listing 3 *(Continued)*

```
55644 TT(51)=197
55646 TT(52)=125
55648 TT(53)=123
55650 TT(54)=235
55652 TT(55)=193
55654 TT(56)=127
55656 TT(57)=69
55658 TT(58)=253
55659 REM SQUARE( BACK/SQ) UP ARROW UNDERLINE
      PERIOD FOLLOW
55660 TT(59)=209
55662 TT(60)=232
55664 TT(61)=212
55665 IFRT=2THENTT(62)=243:REM * TO * ON TYPEWRITER
55666 IFRT=1THENTT(62)=33:REM * INDEX TYPEWRITER
55668 TT(63)=65
55670 TT(64)=232
55671 REM NEXT 26 CHARACTERS ARE LOWER CASE A TO Z
55672 TT(65)=226
55674 TT(66)=80
55676 TT(67)=114
55678 TT(68)=118
55680 TT(69)=116
55682 TT(70)=106
55684 TT(71)=110
55686 TT(72)=84
55688 TT(73)=224
55690 TT(74)=108
55692 TT(75)=112
55694 TT(76)=86
55696 TT(77)=238
55698 TT(78)=120
55700 TT(79)=198
55702 TT(80)=100
55704 TT(81)=96
55706 TT(82)=230
55708 TT(83)=196
55710 TT(84)=124
55712 TT(85)=122
55714 TT(86)=234
55716 TT(87)=192
55718 TT(88)=126
55720 TT(89)=68
55722 TT(90)=252
55724 TT(91)=103:REM TRANS LEFT BR TO COLON
55726 TT(92)=98:REM TRANS RGT BR TO COMMA
55728 TT(93)=232
55730 TT(94)=240
55800 F1=F1+1:REM SELECTRIC DRIVER ROUTINE
55810 GOSUB9000:REM GE RANGE TO PRINT
55820 FORW=QTOR
55830 L=LEN(BF$(W))
55840 FORLP=1TOL
55845 IFL=0THENRS=65:GOTO55890:REM BLANK LINE
55850 WK=ASC(MID$(BF$(W),LP,1))
55860 IFWK<32THENWK=44:REM BAD =.
55870 WK=WK-32:REM SETUP FOR TRANS TABLE
55880 RS=TT(WK):REM GET IBM VALUE
55890 IFRS=65THEN56030:REM SPACE
55900 IFRS=33THEN56030:REM INDEX
55910 IFRS=129THEN56030:REM RETURN
55920 RH=RS:RS=RSAND254:REM SEE IF UPPER OR
      LOWER CASE
55930 IFRH=RSTHEN55980:REM TRUE=LOWER CASE
55940 IFCA=1THEN56030:REM CASE IS UPPER
55950 CA=1:POKEPT,9:REM GOTO UPPER CASE
55960 FORDL=1TO50:NEXTDL
55970 POKEPT,0:GOTO56020
55980 IFCA=0THEN56030:REM CASE IS LOWER
55990 CA=0:POKEPT,17
56000 FORDL=1TO50:NEXTDL
56010 POKEPT,0
56020 FORDL=1TO50:NEXTDL
56030 POKEPT,RS:REM TYPE THE CHARACTER
56040 FORDL=1TO50:NEXTDL
56044 POKEPT,0
56045 IFRS<>129THEN56050
56046 FORDL=1TO1000:NEXTDL:REM RETURN DELAY
56050 NEXTLP
56060 CA=0:POKEPT,17:REM ALWAYS RELEASE SHIFT
      SOLENOID AT END LINE
56070 POKEPT,129:REM ALWAYS RETURN AT END OF LINE
56080 FORDL=1TO50:NEXTDL
56090 POKEPT,0
56100 FORDL=1TO1000:NEXTDL
56110 NEXTW
56120 GOTO120
```

MICRO

Apple Slices

By Tim Osborn

The Pascal Language System has a powerful assembler that includes features like macros and conditional assembly. If you would like to make use of this assembler from DOS 3.3, read on.

Although Pascal runs under its own operating system (which is incompatible with DOS 3.3 or DOS 3.2), data is physically encoded on the Disk II diskettes the same way it is encoded with DOS 3.3. The difference lies in the method used to store the data logically. For example, the directories are structured differently (to prove this, you only need to attempt to CATALOG a language system diskette).

Therefore, if we can physically read the language system diskette from DOS 3.3, then we can examine the data stored on this diskette and make logic decisions based on what we find. In the Apple II system, a physical read is a read of a single sector. In August's "Apple Slices," I presented a method to read any unprotected 16-sector Disk II sector with my AMPERRWTS subroutine. This month's program, BINARY-TRANSFER (listing 1), makes use of AMPERRWTS (see MICRO 51:108) in order to transfer Pascal machine-language files to a DOS 3.3 diskette.

First we must discuss how Pascal logically arranges data on the Disk II diskette. Instead of using sectors of 256 bytes, Pascal uses blocks of 512 bytes. These blocks are made up of two sectors, and are always on the same track. The following table shows the relationship between DOS 3.3 sectors and Pascal blocks:

Relative Block	DOS 3.3 Sector Low (TL%)	DOS 3.3 Sector High (TH%)
0	0	14
1	13	12
2	11	10
3	9	8
4	7	6
5	5	4
6	3	2
7	1	15

The subroutine at lines 6000-6070 (BLOCK READ) will convert any range of blocks into track and sector numbers and read them into memory. Before GOSUBing this subroutine, the beginning of the buffer must be specified by loading BF with the desired value. The starting and ending blocks must be loaded into START and FINISH respectively. The TL% and TH% tables must be initialized *once* before BLOCK READ can be GOSUBed. These tables are BASIC equivalents of the above table and are initialized at line number 1 by a GOSUB to the subroutine at line 4000. Line 6000 simply checks to make sure the file to read into the existing buffer space can fit properly.

This system can handle up to 48 blocks (96 sectors) without stomping on top of AMPERRWTS. Ninety-six sectors can hold 24,576 bytes of code, so it will handle all but the largest of machine-language files. If this becomes a problem, you can modify the program to perform the function in two or more parts. For 99.9% of us though, this is more code than we will ever try to include in one program. Line 6002 controls how many blocks we read by forming the beginning of a FOR-NEXT loop. Line 6005 obtains the track number by taking the integer part of the block number divided by eight. Line 6010 finds the relative block number by taking the modulo eight of the block number. Once we have the relative block number we look up the low sector in the TL% table and read that sector into memory at BF (line 6020). We then bump BF by 256 and look up the high sector in the TH% table and read it into memory at BF (lines 6030-6040). We again bump BF by 256 in preparation to read the low sector of the next block (if necessary). Line 6060 forms the end of the FOR-NEXT loop; if we need more blocks read, we continue or else RETURN.

Figure 1

	$0	$1	$2	$3	$4	$5	$6	$7
$0	SL%	SH%	NO%	N9%	TY%		NL%	
$8	Name Field							
$10								
$18								

SL%, **SH%** — The modulo 256 starting block number in low byte, high byte order.

NO%, **N9%** — The modulo 256 ending block number in low byte, high byte order.

TY% File type code: 02 = code-file, 03 = text-file, 05 = data-file.

NL% File name length

Name field File name

Listing 1

```
0  PRINT  CHR$ (4);"BRUN AMPERRWTS.CODE,A$8000"
1  HIMEM: 7370: GOSUB 4000: REM SET UP BLOCK CONVERSION TBL
2  HOME : HTAB 3: VTAB 1: PRINT "LOAD SOURCE DISKETTE IN DRIVE 1"
3  HTAB 3: VTAB 3: INPUT "HIT RETURN WHEN READY";A$
4 A$ = "":CD$ =  CHR$ (4)
10  DIM CH$(16): DIM HX%(16)
15 BN% = 7424:BF = BN%: REM  BN%=BEGINNING OF BUFFER BF=BUFFER PTR
22  REM ********************** REM
23  REM MODULO EIGHT FUNCTION
25  DEF  FN M8(A) =  INT ((A / 8 -  INT (A / 8)) * 8)
27  REM ********************** REM
28  REM MODULO  256  FUNCTION
29  DEF  FN MDD(A) =  INT ((A / 256 -  INT (A / 256)) * 256 + .05) *  SGN
     (A / 256)
30  REM ********************** REM
32  REM ***GET DIRECTORY***
35 START = 2: REM  BEGINNING BLOCK
40 FINISH = 5: REM  ENDING BLOCK
50  GOSUB 6000: RÉM  READ DIRECTORY
105  REM  ***BELOW ITEMS***
110  REM  ***ARE OFFSETS***
115  REM  ***INTO DIREC.***
120 SL% = 0: REM  START(LOW)
```

(continued)

Transferring Machine-Language Files

To transfer Pascal machine-language code-files, we first must read the directory blocks into memory and search for a code-file. (See figure 1 for the format of the directory entries.) Since the directory blocks are located in blocks two through five, we must read these into memory (lines 35-50) and scan for a code-type file. Code-type files are recognized by a file-type code of two (see line 210). Once we find a code-file we print out the file name (lines 213-215) and ask the user if this is the file to be transferred (line 230). If the user responds with a "Y", we find the beginning and the ending block of the file at lines 290 and 300 respectively.

We then read the file into the buffer (lines 320 through 340). Next we find the length of the code-file (line 410) and check it to make sure it's valid (line 412). Invalidity can occur if the code-file is a p-code file (Pascal compilation), which does not appear to use this same length field. This in turn becomes a check to make sure we are reading a machine-language file. The check only guarantees that the file is not a machine-language file if the length is invalid. If the length is valid, it may still be a p-code file. The simple way to avoid such problems is to know your file name.

Since the first block of the machine-language file is only for system header information, this block is skipped by setting the beginning address of the BSAVE to BF + 512 (line 420). The user is next asked to insert the destination diskette in drive 1 and hit RETURN when ready (lines 422-424). Once RETURN is depressed, a BSAVE is performed using the same file name used under the Pascal system. The length and beginning address have been computed and are in LE% and AD% respectively. Since, under the present buffer configuration, AD% will always be equal to $1F00 and the user may wish to have his program BLOADED at a different address, I have included some code to zap the load address of the binary file. This saves the user from having to BLOAD the file at the desired address and BSAVE it again. I do this by first asking the user if he wants to BLOAD the file at a point other than $1F00. If the response is yes, then the user is asked to input a hex address. The hex address is then validated and converted to decimal (lines 505 through 630). Each catalog sector is read until the desired file is found (lines 650-810). For a discussion of the DOS 3.3 catalog structures, see the August

Listing 1 *(continued)*

```
125 SH% = 1: REM  START(HIGH)
130 NO% = 2: REM   END  (LOW)
140 N9% = 3: REM   END  (HIGH)
145 TY% = 4: REM  TYPE
150 NL% = 6: REM  NAME LENGTH
160 NM% = 7: REM  NAME
170 NF% = 16: REM  NO. FILES
180  REM  *****************
182 LN% = 26: REM  LENGTH OF ENTRIES
185 BF = BN%
188 NF =  PEEK (BF + NF%)
190  IF NF = 0 THEN  HTAB 3: VTAB 5: PRINT "NO FILES ON VOLUME": END
195  FOR I = 1 TO NF: REM  NUMBER OF FILES
200 BF = BF + LN%
210  IF  PEEK (BF + TY%) <  > 2 GOTO 260
213  HOME : HTAB 3: VTAB 5:CH$ = ""
215  FOR J = 1 TO  PEEK (BF + NL%):CH$ =  LEFT$ (CH$,J) +  CHR$ ( PEEK (B
     F + NM% + J - 1)): NEXT
220  PRINT CH$
225  HTAB 3: VTAB 7
230  PRINT "IS THIS THE FILE? (Y)ES OR (N)O ";: GET A$
240  IF A$ = "Y" THEN I = NF: GOTO 260
250  IF A$ <  > "N" GOTO 225
260  NEXT
270  IF A$ <  > "Y" THEN  END
290 BG% =  PEEK (BF + SL%) +  PEEK (BF + SH%) * 256: REM BEGINING BLOCK
300 BE% =  PEEK (BF + NO%) +  PEEK (BF + N9%) * 256: REM ENDING BLOCK
310 BF = BN%: REM   RESET BUFFER PTR
320 START = BG%: REM  BEGINNING BLOCK
330 FINISH = BE% - 2: REM  SKIP LAST BLOCK
340  GOSUB 6000: REM  READ FILE
400 BF = BN%
410 LE% = ( PEEK (BF + 6) +  PEEK (BF + 7) * 256) - 16
412  IF LE% < 0 THEN  HOME : HTAB 3: PRINT "NOT AN OBJECT FILE ": HTAB 4:
      PRINT "CANNOT TRANSFER": END
420 AD% = BF + 512
422  HOME : HTAB 3: VTAB 1: PRINT "LOAD DESTINATION DISKETTE IN DRIVE 1";

424  HTAB 3: VTAB 3: INPUT "HIT RETURN WHEN READY";A$
430  PRINT CD$;"BSAVE ";CH$;",A";AD%;",L";LE%
435 FL$ = CH$: REM SAVE FILE NAME
440  HOME : HTAB 3: VTAB 1: PRINT "FILE SUCCESSFULLY BSAVED"
450  HTAB 3: VTAB 3: PRINT "DO YOU WISH TO BLOAD"
460  HTAB 3: VTAB 4: PRINT "TO OTHER THAN HEX 1F00?"
465  HTAB 3: VTAB 5: PRINT "(Y)ES OR (N)O ";: GET A$
470  IF A$ = "Y" THEN  GOTO 492
480  IF A$ <  > "N" GOTO 465
490  END


492  FOR I = 0 TO 15: READ CH$(I),HX%(I): NEXT
495  HOME
500  HTAB 3: VTAB 1: INPUT "INPUT HEX ADDRESS ";A$
505 AL% =  LEN (A$)
510  IF AL% > 4 THEN  HTAB 3: VTAB 3: PRINT "TOO LONG -TRY AGAIN        ":
      GOTO 500
520  IF AL% < 3 THEN  HTAB 3: VTAB 3: PRINT "TOO SHORT -TRY AGAIN       ":
      GOTO 500
530 HX = 0
570  FOR I = 1 TO AL%
580 MD$ =  MID$ (A$,I,1):FOUND$ = ""
590  FOR J = 0 TO 15
600  IF MD$ = CH$(J) THEN HX = HX + (HX%(J) * 16 ^ (AL% - I)):FOUND$ = "Y
     "

610  NEXT J
620  IF FOUND$ <  > "Y" THEN : HTAB 3: VTAB 3: PRINT "INVALID FOR HEX -TR
     Y AGAIN": GOTO 500
630  NEXT I
635  HOME
639  REM  FIND FIRST CATALOG SEC IN VTOC
640 BF = BN%
650 TRK% = 17:SEC% = 0: & R(TRK%,SEC%,BF)
660 TRK% =  PEEK (BF + 1)
670 SEC% =  PEEK (BF + 2)
680  & R(TRK%,SEC%,BF)
685 TY% = 2: REM  TYPE OFFSET
690 NM% = 3: REM FILE NAME OFFSET FOR CATALOG
700 LN% = 35: REM   LENGTH OF CATALOG ENTRIES
725  FOR I = 1 TO 15: REM  15 CATALOGS
727 BF = BF + 11:CH$ = "":FOUND$ = ""
728  REM BF POINTS AT FIRST CATALOG ENTRY
730  FOR J = 1 TO 7: REM  7 FILES PER CATALOG
740 TRK% =  PEEK (BF)
750  IF TRK% = 0 OR TRK% = 255 OR  PEEK (BF + TY%) <  > 04 GOTO 780
755 S% = 0
760  FOR L = 1 TO 30:CH$ =  LEFT$ (CH$,L) +  CHR$ ( ASC ( CHR$ ( PEEK (BF
      + NM% + L - 1))) - 128)
762  IF  MID$ (CH$,L,1) = " " THEN S% = S% + 1: GOTO 766: REM  COUNT TRAI
     LING SPACES
764 S% = 0
766  NEXT L
769 CH$ =  LEFT$ (CH$,30 - S%): REM  REMOVE TRAILING SPACES
```

Listing 1 *(continued)*

```
770  IF CHS = FLS THEN I = 15:SEC% =  PEEK (BF + 1): GOTO 810
780 BF = BF + LN%:CH$ = ""
790  NEXT J
800 BF = BN%:TRK% =  PEEK (BF + 1):SEC% =  PEEK (BF + 2): & R(TRK%,SEC%,B
     F): REM GET NEXT CATALOG
810  NEXT I
830 BF = BN%: & R(TRK%,SEC%,BF): REM READ FIRST T/S LIST
840 TRK% =  PEEK (BF + 12):SEC% =  PEEK (BF + 13): & R(TRK%,SEC%,BF): REM
     READ FIRST SECTOR OF FILE
850  POKE BF + 1, INT (HX / 256): REM POKE HIGH BYTE
860  POKE BF, FN MOD(HX): REM POKKE LOW BYTE
880  & W(TRK%,SEC%,BF)
890  HOME : HTAB 3: VTAB 1: PRINT "BLOAD ADDRESS CHANGE SUCCESSFULL": END

4000  REM   ***SET UP****
4001  REM   *CONVERSION *
4002  REM   *   TABLE   *
4003  REM   *************
4005  DIM TL%(7),TH%(7)
4030  FOR II = 0 TO 7: READ TL%(II),TH%(II): NEXT
4040  RETURN
4150  DATA  0,14
4160  DATA  13,12
4170  DATA  11,10
4180  DATA  9,8
4190  DATA  7,6
4200  DATA  5,4
4210  DATA  3,2
4220  DATA  1,15
5000  REM  ***HEX CONVERSION TABLE****
5001  DATA "0",0,"1",1,"2",2,"3",3,"4",4,"5",5,"6",6,"7",7,"8",8,"9",9,"A
     ",10,"B",11,"C",12,"D",13,"E",14,"F",15
5998  REM *****************
5999  REM  ** BLOCK READ  **
6000  IF FINISH - START > 48 THEN  HOME : PRINT "FILE TOO LONG FOR BUFFER
     ": END
6002  FOR BK = START TO FINISH
6005 TRK% =  INT (BK / 8)
6010 RB% =  FN M8(BK): REM RELATIVE BLOCK
6020 SEC% = TL%(RB%): & R(TRK%,SEC%,BF)
6030 BF = BF + 256
6040 SEC% = TH%(RB%): & R(TRK%,SEC%,BF)
6050 BF = BF + 256
6060  NEXT
6070  RETURN

]M
```

"Apple Slices" and the DOS manual Appendix C.

When the desired file is found in the catalog, the track-sector list must be read (line 830). At offset 12 and 13 of the track-sector list is the track and sector of the first sector that comprises the file. This sector is read in (line 840) and the BLOAD address that is contained at offset 0 in low byte/high byte order is updated (lines 840-860). Once this is done, the sector is re-written (line 880) and the user is notified that the BLOAD address change was accomplished.

Using the Pascal Assembler

In the language system, all assembler files must start with either a .PROC or .FUNC assembler directive before any assembler commands can be used. This .PROC or .FUNC directive is only to satisfy the assembler; the directive is unnecessary from DOS 3.3's point of view. The .ORG directive is taken as an offset, relative to the start of the assembly file. The location counter is advanced filling the memory space with 0's from where the location counter was to the new location. You can avoid this using the .ABSOLUTE assembler directive as the first statement in the source file. This stops the generation of relocation information and makes the .ORGs act as absolute locations, rather than relative. For more information, read chapter six of the *Apple Pascal Operating System Manual*, paying particular attention to pages 158-161.

Have fun using the Language System assembler with DOS 3.3, and don't be afraid to experiment. What you learn may help others discover new ways to increase the versatility of the Apple II computer.

References:

1. *The DOS Manual*, Apple Computer Inc., Cupertino, California 95014.
2. Walker, Robert D., "RELOC," MICRO, January 1982.
3. Osborn, Tim, "Apple Slices," MICRO, August 1982.
4. *Apple Pascal/Operating System Reference Manual*, Apple Computer Inc., Cupertino, California 95014.

MICRO

Introduction to Turtle Graphics in Apple Pascal

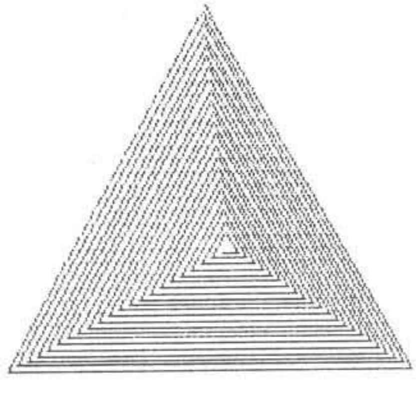
by John R. Raines

This article describes the Turtle graphics routines available with Apple Pascal. A graphics screen dump is also included.

PROCEDURE Turtle
requires:
Apple II
with Pascal language

The Apple Pascal manual, and most of the basic references on the Pascal language, do not discuss Turtle graphics. This group of procedures is not a standard part of Pascal. Many users of Apple Pascal will not have had any exposure to the literature on Turtle graphics, and should learn how to take advantage of Turtle graphics instead of using the Cartesian graphic commands.

Turtle graphics was developed by the LOGO group at MIT, headed by Seymour Papert. Their approach has been to let children program in LOGO with support from teachers. Children use Turtle graphics either to draw on a screen — just as in Apple Pascal — or to control the movement of a small robot "turtle." (Robot turtles are available for the Apple but apparently cannot be controlled from Pascal without creating the necessary software yourself.) A robot turtle catches the imagination of a child more readily than a TV turtle.

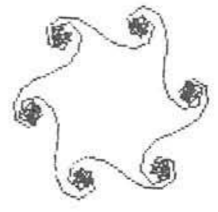


Output from "POLYSPI"

Having the turtle on the floor with the child encourages the child to learn to debug turtle programs by "playing turtle:" following the procedures in the program by physically walking it through.

Over the years, work with Turtle graphics produced some remarkably elegant ways of teaching. The books from the MIT group give examples of fundamental interrelationships between the Turtle graphics picture and the structure of the program to draw the picture. This is a great aid in teaching by discovery.

The algorithms they present are usually written in LOGO, or something very close to it called Turtle Procedure Notation. I've adapted some of these routines to Apple Pascal and will use them to illustrate the power of Turtle graphics.



Output from "INSPI"

The CENT739 Routine

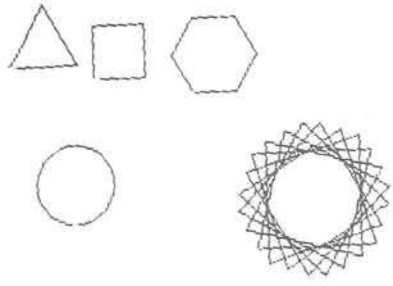
Before I discuss the Turtle graphics procedures, let me explain the routine I used to produce graphics screen dumps. If you have another printer, then you should replace this routine with one of your own. Another good alternative is just to replace it with a delay which lets you look at the screen for a while.

I wrote the short Pascal Procedure described here to dump the high-resolution screen onto a Centronics 739 printer. The printer is put into graphics mode by a sequence of escape characters. Then, each byte sent to it after that represents one vertical column of six dots. Thus 280 bytes of data must be sent for each six-dot high row.

Fortunately, the Pascal graphics package includes a SCREENBIT function that reports the status of a specified dot on the screen. Using this, it was easy to write the routine in Pascal. In our system the hardware communicates at just 300 baud. The Pascal routine is more than fast enough to keep up with the hardware at that rate. The procedure listed here could be used with any Pascal program to dump the high-resolution screen onto the Centronics 739 printer. When it is done, it leaves the printer in 16.7 characters-per-inch monospaced mode.

Drawing Polygons

The POLY procedure listed here is truly elegant. It will draw virtually any regular polygon (and some other interesting variations as shown at the lower right corner of the POLY output). The routine takes the length of one side of the polygon and the angle to be turned between sides as its inputs. Note that procedures like POLY are independent of screen position, so there is no need for shape tables. To teach children, you can use a slightly simpler version of the algorithm at first — by omitting the test for angle = 0 and by using an endless loop. Then add the refinements back in as the child realizes the usefulness of the extra steps. Note also that the angle to produce a triangle is 120 degrees rather than 60 degrees, because the turtle is turning through the outside of the angles rather than the inside.



Output from "POLY"

Pascal PROCEDURE Listing

```
PROGRAM TURTLE (OUTPUT,REMOUT);
USES TURTLEGRAPHICS,APPLESTUFF;
PROCEDURE POLY (SIDE : INTEGER; ANGLE : INTEGER);
VAR I  : INTEGER;
BEGIN
  I := TURTLEANG;
  PENCOLOR(WHITE);
  IF ANGLE <> 0 THEN
    REPEAT
      MOVE (SIDE);
      TURN (ANGLE);
    UNTIL I = TURTLEANG
   ELSE
    WSTRING ('ANGLE IN "POLY" WAS 0.');
  PENCOLOR (BLACK);
END;     (* OF POLY *)
PROCEDURE POLYSPI (SIDE : INTEGER; ANGLE : INTEGER);
VAR I   : INTEGER;
BEGIN
  I := SIDE;
  IF ANGLE <> 0 THEN
    REPEAT
      MOVE (I);
      TURN (ANGLE);
      I := I + 3;
    UNTIL I > 200   (* INEVITABLY SPIRALS OFF SCREEN *)
   ELSE
    WSTRING ('ANGLE IN "POLYSPI" WAS 0.');
END;
PROCEDURE INSPI (SIDE : INTEGER; ANGLE : INTEGER; INC : INTEGER);
VAR I,X,Y  : INTEGER;
BEGIN
  X := TURTLEX;
  Y := TURTLEY;
  I := ANGLE;
  IF ANGLE <> 0 THEN
    REPEAT
      MOVE (SIDE);
      TURN (I);
      I := I + INC;
    UNTIL KEYPRESS
   ELSE
    WSTRING ('ANGLE IN "INSPI" WAS 0.');
END;    (* OF INSPI *)
PROCEDURE BRANCH (LENGTH : INTEGER; LEVEL : INTEGER);
BEGIN
IF LEVEL <> 0 THEN
  BEGIN
  TURN (45);
  MOVE (LENGTH);
  BRANCH (LENGTH DIV 2, LEVEL -1);  (* RECURSE, HALF SIZE AND ONE LESS LEVEL *)
  MOVE (-LENGTH);  (* BACK TO PARENT NODE & THEN TAKE RIGHT HAND BRANCH *)
  TURN (-90);      (* TWO TURNS OF -45 EACH *)
  MOVE (LENGTH);   (* OUT TO RIGHT HAND BRANCH NOW *)
  BRANCH (LENGTH DIV 2, LEVEL -1);  (* AND ALL THAT IS BELOW THE RIGHT HSIDE *)
  MOVE (-LENGTH);  (* BACK UP TO PARENT AGAIN *)
  TURN (45);       (* AND RETURN TO CALLING HEADING *)
  END;
END;    (* OF BRANCH *)
PROCEDURE HILBERT (SIZE:INTEGER; LEVEL:INTEGER; RL:INTEGER);
 BEGIN
 IF LEVEL <> 0 THEN
   BEGIN
   TURN (RL*90);
   HILBERT (SIZE,LEVEL-1,-RL);
   MOVE (SIZE);
   TURN (-RL*90);
   HILBERT (SIZE,LEVEL-1,RL);
   MOVE (SIZE);
   HILBERT (SIZE,LEVEL-1,RL);
   TURN (-RL*90);
   MOVE (SIZE);
   HILBERT (SIZE, LEVEL-1,-RL);
   TURN (RL*90);
   END;
 END;
PROCEDURE CENT739;
 VAR
  GRAPH : TEXT;
  X,Y,I : INTEGER;
  BYTE  : INTEGER;
```

Pascal PROCEDURE Listing *(continued)*

```
BEGIN
REWRITE (GRAPH,'REMOUT:');
WRITE (GRAPH,CHR(13), CHR(27),CHR(37),CHR(48)); (* INIT TO GRAPH MODE *)
Y := 191; X := 0;
REPEAT
  REPEAT
    BYTE := 32;
    IF SCREENBIT (X,Y)   THEN BYTE := BYTE + 1;
    IF SCREENBIT (X,Y-1) THEN BYTE := BYTE + 2;
    IF SCREENBIT (X,Y-2) THEN BYTE := BYTE + 4;
    IF SCREENBIT (X,Y-3) THEN BYTE := BYTE + 8;
    IF SCREENBIT (X,Y-4) THEN BYTE := BYTE + 16;
    IF SCREENBIT (X,Y-5) THEN BYTE := BYTE + 32;
    WRITE (GRAPH, CHR(BYTE));
    X := X+1
  UNTIL X = 280;
  WRITE (GRAPH, CHR(13));
  Y := Y - 6;
  X := 0;
UNTIL (Y < 0);
WRITE (GRAPH,CHR(13),CHR(13),CHR(27),CHR(20));
END;
PROCEDURE GRAFIT;
VAR I,J : INTEGER;
BEGIN
  CENT739;
  INITTURTLE;
  MOVETO (0,0);
END;
 BEGIN       (* MAIN PROGRAM *)
 INITTURTLE;
 FILLSCREEN(BLACK);
 TURNTO (10);
 MOVETO (10,105);
 PENCOLOR(WHITE);
 POLY (40,120);  (* TRIANGLE *)
 MOVETO (60,105);
 POLY (30,90);  (* SQUARE *)
 MOVETO (120,105);
 POLY (25,60);   (* HEXAGON *)
 MOVETO ( 60,20);
 POLY ( 4,10);   (* NEARLY A CIRCLE *)
 MOVETO (170, 20);
 POLY (70,105);  (* 'STAR' *)
 MOVETO (10,0);
 WSTRING ('OUTPUT FROM "POLY"');
 GRAFIT;         (* DUMP GRAPHICS SCREEN ONTO PAPER *)
 MOVETO (140,75);
 TURNTO (0);
 PENCOLOR(WHITE);
 POLYSPI (8,120);
 PENCOLOR(BLACK);
 MOVETO (10,0);
 WSTRING ('OUTPUT FROM "POLYSPI"');
 GRAFIT;
 MOVETO (140,50);
 TURNTO (0);
 PENCOLOR(WHITE);
 INSPI (8,200,15);
 PENCOLOR(BLACK);
 MOVETO (10,0);
 WSTRING ('OUTPUT FROM "INSPI"');
 GRAFIT;
 MOVETO (139,0);
 TURNTO (90);
 PENCOLOR (WHITE);
 BRANCH (70,6);
 GRAFIT;
 PENCOLOR(WHITE);
 HILBERT (3,6,1);
 PENCOLOR(BLACK);
 MOVETO (200,160);
 WSTRING('OUTPUT');
 MOVETO (200,120);
 WSTRING('FROM');
 MOVETO (200,80);
 WSTRING('HILBERT');
 GRAFIT;
 TEXTMODE;
 END.
```

MICRO

Three Faces of Apple LOGO

by Edward H. Carlson

A look at several aspects of LOGO and its application to education.

More than just a computer language, LOGO is really an educational movement. Since 1968, the LOGO Project at the Massachusetts Institute of Technology has been concerned with how computers can help children develop mentally. They made LOGO easy to enter for beginners, but also valuable as a final language for adults. Now we have LOGO for Apples at home and school.

The philosophy behind LOGO is explained in a fascinating book by Seymour Papert called *Mindstorms: Children, Computers and Powerful Ideas* (published by Basic Books, Inc. in 1980). He points out the growth in thinking skills that can occur when people interact with computers in the right way. For him, the computer should be "an-object-to-think-with" rather than a drill master or answer-getter.

LOGO began its life on minicomputers before the era of personal computers. It cannot be stuffed into a small memory and still retain its unique characteristics, so its introduction to personal computers was delayed until the development of inexpensive memory. Now versions of LOGO have been developed for the Texas Instruments 99/4A and the Apple II computer. This article describes the Apple versions of LOGO only. Because of the TI's special hardware, the TI version differs somewhat from the Apple version.

There are two major versions of LOGO for the Apple. The first was written at MIT and is licensed to two suppliers, Krell Software Corp. and Terrapin Inc., which sell slightly different forms. The second version was written by Logo Computer Systems, Inc., and is sold through Apple dealers. I will compare these versions at a later point in this article.

LOGO presents three faces to the world. Which face confronts you depends on the level of programming sophistication.

The face most often mentioned is "TURTLE TALK," a set of graphics commands that serves as a splendid introduction to computing for any beginner, and especially for young children. Removing the turtle graphics mask reveals a general computer language on a par with BASIC. However, LOGO uses procedures to encourage structured programming. The advanced user can remove the second mask and find a powerful LISP-like language. This face of LOGO introduces techniques common to artificial intelligence programming.

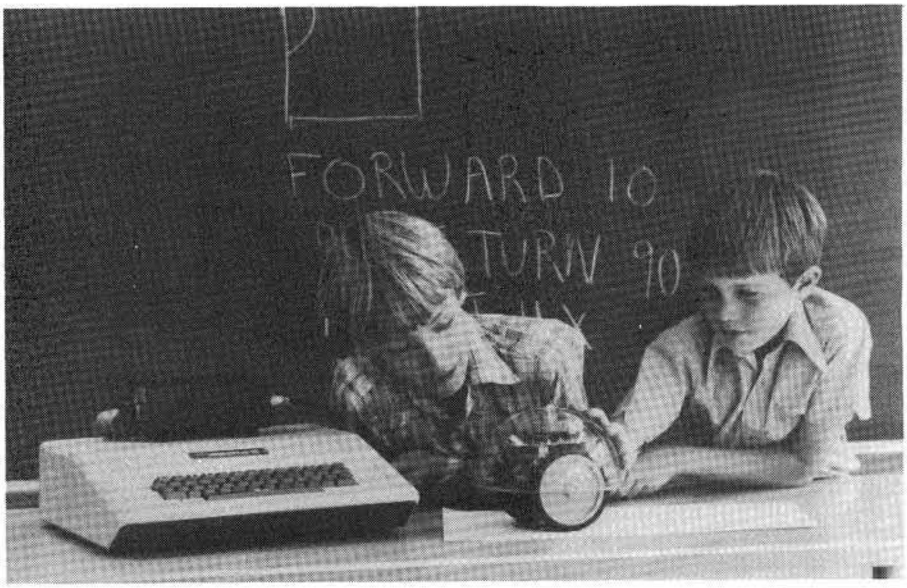
The Face of the Child: Turtle Graphics

TURTLE TALK appeals to children because they like to create pictures. The effects of their commands to the computer are instantly visible in concrete form. They learn that you get results by constructing procedures, that procedures are broken down into subprocedures, that first efforts usually give flawed results, and that debugging is a natural part of creation.

Turtle commands directly emulate the way humans move around in space. Graphics in Applesoft BASIC require the user to be familiar with Cartesian coordinate systems, an awkward entry into programming for young children.

Rather than saying something like "HLIN 15, 17 AT 3", LOGO says "FORWARD 3" and the cursor (turtle) moves three high-resolution spaces ahead in the direction it was facing, leaving a track behind it on the screen. The turtle is turned to the right by a command RIGHT 40, where 40 is the desired turn angle in degrees. Likewise, a turn to the left is commanded by LEFT followed by the angle. In all, there are about 30 graphics commands in LOGO. Others add color, hide the turtle or his track, and output such information as the location and heading of the turtle.

There is even a toddler version of TURTLE TALK that controls the turtle with single keystroke commands.



Children enjoy writing programs to control the Terrapin turtle. This versatile robot draws pictures, senses contact with objects, has controllable "eyes" (lights), and a two-toned horn. Through its sense of touch, it can be programmed to back away from obstacles, and even to run mazes. *(Photo courtesy of Terrapin, Inc.)*

Listing 1

```
A.
 10 REM PRINT HI              ;  BASIC
 20 FOR N = 1 TO 10:REM       ;  A FOR NEXT LOOP
 30 PRINT "HI"
 40 NEXT N
B.
TO PRINT.HI                   ;  LOGO
   REPEAT 10[PRINT "HI]       ;  A "FOR NEXT" LOOP
END
C.
TO PRINT.HI1  :N              ;  TAIL RECURSIVE IN LOGO
   IF :N = 0 STOP             ;  TEST FOR EXIT
   PRINT "HI                  ;  BODY OF THE PROCEDURE
   PRINT.HI1  :N-1            ;  CALLS ITSELF RECURSIVELY
;  This program would be called like this:
;  PRINT.HI1  10
END
D.
 10 REM                       ;  TAIL RECURSIVE IN BASIC
 20 N = 10:GOSUB  100         ;  REM SUBROUTINE CALLED BY LINE
                                 NUMBER
 99 END
100 REM SUBROUTINE            ;  PART OF PROGRAM, NOT SELF-
                                 STANDING
110 IF N = 0 THEN RETURN      ;  REM TEST FOR EXIT
120 PRINT "HI"  :REM          ;  BODY OF THE PROCEDURE
130 N = N-1:GOSUB 100:REM     ;  CALLS ITSELF RECURSIVELY
140 REM NOTE: It works in this program, but you can get into
150 REM trouble with recursive calls in BASIC. If N > 23 then the
    system stack overflows.
```

Young children who want to draw a given figure are encouraged to "play turtle": to conceptualize the desired movement of the turtle by moving their own bodies in a similar way. The direct analogy of body motion with commands to the turtle exemplifies the first of Seymour Papert's "powerful ideas," called "syntonic learning." It can be paraphrased as "Anything is easy to learn if you can assimilate it to your collection of models."

Turtle graphics first appeared in LOGO but have been transported to other languages. A version written by David Krathwohl in Applesoft BASIC is listed and described in *NIBBLE* (Vol. 3, No. 1, 1982, page 23). A disk containing this version is available from *NIBBLE*. Apple Pascal has turtle graphics using somewhat different commands from those of LOGO.

The Face of the Workman: Standard Programming with LOGO

Put on your climbing gear, hook up to our line and we'll start climbing LOGO mountain. We will rest at day's end in the lodge at the top. Then we will sip our hot muddled cider and gaze at the inscription over the fireplace where the LOGO equivalent for each BASIC command is set forth.

LOGO has many elements that make sense to BASIC programmers after they are translated. For example, a LOGO program is a hierarchy of procedures that call one another. Every LOGO procedure is selfstanding like a BASIC program, but also can be called like a BASIC subroutine. Any procedure may call any other procedure, or even call itself recursively. Rather than having one program in memory at a time, as does BASIC, LOGO has many procedures in memory at once. This is a strong point in the LOGO philosophy, as breaking a task into subtasks is another of the "powerful ideas." Each procedure is called by name. By contrast, BASIC calls its subroutines by line number (i.e., "GOSUB 440"), which is much less informative.

LOGO procedures tend to be short and deeply nested. As they call one another, they may form a "tree."

Procedures "built in" to LOGO are called "primitives," and correspond to the commands, statements, and functions of BASIC. However, the style of LOGO is to treat primitives and procedures in the same way.

Any procedure may have zero, one, or more inputs. These are like the arguments of functions in BASIC. A procedure may have one output or no outputs. Procedures that return an output are very commonly used in the "input" spot of another procedure. This is analogous to the FN function in BASIC, which sits in an expression and returns a value to it. Such usage is much more common in LOGO than in BASIC. Procedures may carry out many complex tasks before returning, just as do BASIC subroutines.

LOGO has loops (using the command REPEAT) and IF...THEN...ELSE structures, and these will be easily understood by BASIC users. Another way of looping is "tail recursion," where a procedure has a call to itself at its end. The first command after reentering the procedure is a test for exit. Listing 1 shows a program to print "HI" 10 times. Two versions each are written in BASIC and in LOGO. One is in the style familiar to BASIC programmers, and the other uses tail recursion. The number of times it prints "HI" is prescribed by an input variable.

So far, our climb up LOGO mountain seems much like the climb up BASIC mountain we made earlier in our careers.

There seem to be four data types in LOGO, but they reduce to two: word and list. The most basic is the word. It is like a string constant in BASIC. The central position of word in LOGO is not surprising; "logos" means "word" in Greek.

The two numerical data types, integer and floating point number, are special cases of the word data type. If a word is composed of numerical characters, it will be recognized as a number by procedures that require numerical input. Numbers may be integers betweeen ± 2 billion, or floating point between 10^{-38} and 10^{38}. No functions are needed to change data from strings to numbers (such as the VAL and STR$ of BASIC) or from integers to floating point numbers. All this is done automatically, and error messages are printed if the data type is inappropriate.

The other data type is the list. This has no counterpart in BASIC, but is an essential feature that makes LOGO like LISP. A list is a list of elements that may be words or other lists. It is a way to make complicated data types. A list literal is enclosed in square brackets, like [GETUP]. With list comes the first queasy feeling to the mind of the brainwashed BASIC programmer that there may not be a lodge at the top of LOGO mountain!

LOGO has no counterpart to the array in BASIC. The list can take over the duties of the array, but is really designed to work in an entirely different way.

Every variable in BASIC is "global." Once it has been assigned a value, it can be used anywhere in the program and will have the assigned value.

In contrast, variables in LOGO are generally "local" to the procedure in which they appear. The same name can occur in more than one procedure and there is no interference between them. You can define a variable to be "global" just as in BASIC, but this is somewhat out of tune with the style of LOGO.

Each time a procedure is called, a local library is set up and values of local variables are put in it. Calling other procedures (or even another call to the same procedure) that have the variable will not change the value in the original local library. When the procedure is exited, its local library is erased.

LOGO rarely uses line numbers or identifiers, and uses almost no punctuation. This makes a LOGO program look rather featureless to the seasoned BASIC programmer. In fact, the program looks like a list of words. (This is a clue, Sherlock!)

A "bare" name like SING calls a procedure that was previously defined at a TO SING line. A name with quotes on the front, like "TED, is a "word." Its BASIC equivalent would be the string constant "TED". A name with a colon in front (a colon is called "dots" in LOGO) is ... a "thing." A "thing?" *Hold on!* Dig in your ice axe; we're falling off the mountain!

In BASIC, a variable has a "value" that can be set using the LET command, such as "LET A=5.5". In LOGO, a "word" may have a "thing" (value) associated with it. The "thing" may be a "word" or a "list." One way to assign a thing to a word is to use the MAKE primitive: MAKE "TED "ANGRY. From then on, you refer to the "thing" associated with the word "TED by marking it with "dots," like this :TED. So :TED = "ANGRY.

But it doesn't stop there! Unlike the case in BASIC, you can have a "thing of a thing." Using MAKE "ANGRY "REDFACED, you now have :ANGRY = "REDFACED to add to your :TED = "ANGRY. You are starting to build another tree, one where "things" are the nodes.

More strangely, even the numerical variables, being words too, can have "things." The thing belonging to "49.8 would be referred to as :49.8 and could be the word "PRESENT.AGE.

Well, the lodge at the top of the mountain is a snowy mirage. BASIC and LOGO only partially correspond. of course, like all extant programming languages, they are members of the group that Douglas Hofstadter called "FLOOP" in his book *Godel, Escher and Bach: An Eternal Golden Braid.* Theoretically, they are equally powerful and differ only in how they mimic the mindset of their human users. Practically, each specializes in certain types of tasks.

LOGO can be used to write straightforward programs. It brings clarity and structure to some tasks, while it handles data awkwardly in others. But the major reason a BASIC programmer should learn LOGO is for the new perspective it reveals about the art of thinking. Let's move on to the last face.

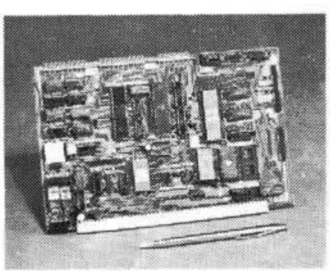
The Face of the Sage: LOGO and LISP

Trees. The procedures are trees. The data are trees in the form of lists of lists. This sets up the unifying trick of LOGO: the primitives DEFINE and RUN make a data list into a procedure. The distinction between "data" and "procedure" is deliberately erased in LOGO, which is designed to write self-modifying code.

Although I find this kind of programming intriguing, I will have little to say about it here as I am not yet an expert. It does seem certain that mastering this high-level LOGO will provide ideas and programming skills that will carry over into programming in other languages.

Literature About LOGO

Papert's *Mindstorms,* which describes the LOGO philosophy and provides interesting examples of its use by children, is not intended to be a LOGO description.



Two books by Professor Harold Abelson, *Apple LOGO* and *LOGO for the Apple II*, are adult tutorials for LCSI and MIT LOGO, respectively." However, I found too little information in it about the differences between LOGO and languages like BASIC. It does explain how lists are used in LOGO versions of two famous old programs: DOCTOR, which acts like a Rogerian "non-directive" psychologist, and ANIMALS, which "learns" about more animals as the user plays with it. A BASIC version of the ANIMALS program is included on the DOS 3.3 SYSTEM MASTER disk.

Abelson refers the reader to another book: *LISP* by Winston and Horn, Addison-Wesley, 1981; for further training in the use of list-oriented languages.

Turtle Geometry, by Harold Abelson and Andrea di Sessa, starts at the high school geometry level and touches many other topics, such as vectors, topology, symmetry groups, and curved space. Published by MIT Press, 1980, $20.00.

Special Technology for Special Children by E. Paul Goldenberg treats use of the computer in special education for intellectually and physically handicapped children, and by clinicians. University Park Press, Baltimore, 1979, $12.95.

"Turtle News" is published by the Young People's LOGO Association, 1208 Hillsdale Drive, Richardson, Texas 75081. It comes out monthly and gives news about activity in LOGO using the Apple and Texas Instruments computers, but also about other turtle systems such as the Atari PILOT.

Apple LOGO Requirements

LOGO on the Apple requires a disk drive and a 16K memory card. Like BASIC, Apple LOGO is an interpreted language.

MIT LOGO

This package consists of two disks and a 55-page reference manual, called *LOGO for the Apple: Technical Manual*. One disk is the LOGO Language Disk, which is copy protected. The other is the Utilities Disk, which has 17 files in DOS 3.3 binary-file format. There are about 115 primitives in this package, which is not available directly from MIT, but only (at present) from the two sources mentioned below.

Among the utilities is an assembler to write machine-language programs. This version of LOGO interfaces easily to machine language. You can write "labels" in the assembly code and reach them directly from LOGO by name!

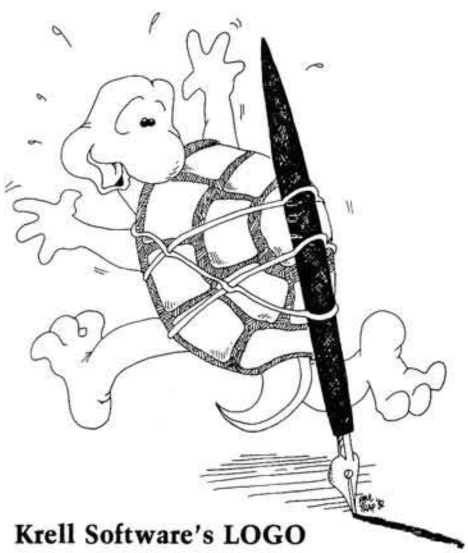
There is also a file utility program, the ANIMAL program, a MUSIC program, DYNATRACK, which teaches Newtonian motion by trial and error, and INSTANT, which makes LOGO into a "one key" system for very young users.

The TERRAPIN Version of LOGO

The Terrapin Inc. version of LOGO consists of the MIT package plus nine more utility files, and a tutorial that is presently the Abelson book described above. They expect to replace this with their own tutorial when it is ready. The cost is $149.95 and a backup to the Language Disk is available for $15.00 more.

Among the extra utilities is a driver for the hardware turtle that Terrapin sells. A hardware turtle would be a happy addition in the lower grades of a school system.

The rest of the extra utilities are small additions to the system to make its use easier. The most visible of these makes the LOGO editor useful as a general text editor.



Krell Software's LOGO

Krell Software Corporation's version of MIT Apple LOGO has four disks, the Technical Manual, and a large wall chart with the LOGO commands on it. There are two copies of the Language Disk and one each of the Utility Disk (with 52 extra files, of which 24 are alternate shapes for the turtle cursor), and Krell's own "Alice In Logoland," a tutorial program. The package price is $149.95.

There are 12 utilities that make it easy to use the system. The "Instant Logo Tutor," read into workspace beside your current file, gives a summary of keyboard, editing, graphics, and file commands when you enter HELP.

"Alice In Logoland" contains 20 chapters; each demonstrates a feature of LOGO. There is no direct teaching, but you can follow the programs in the TRACE mode of LOGO and see how the programs work.

Editors note: Krell is selling an economy version without "Alice" for $99 and a deluxe version with a "sprite" board for about $500.00.

Logo Computer Systems' LOGO

This version, distributed by Apple Co. through their dealers, consists of LOGO on a single disk, with one backup disk. The language differs from the MIT version both in the number and nature of primitives (about 150). There are high-level entities, such as "package" and "property list," that are not available in the MIT version. There is no assembler, and files are in DOS 3.3. format.

The set includes two manuals: *Apple Logo Reference Manual*, and *Introduction to Programming Through Turtle Graphics*. Both are written at the adult level in a clear style. The price is $175.

Hardware Turtles

Terrapin Inc. sells its hardware turtle assembled or in kit form. The turtle interfaces to a computer through a cable. Interface hardware and software are available for several different computers. The turtle is about 8" in diameter. Its movements, pen, "eyes" that light, and horn can all be controlled by the computer, and it can detect when it has run into an obstacle. Expect to pay in the range of $400 to $850.

HERBUG is a highly stripped-down turtle available in kit form from Herbach & Rademan, Inc., 401 East Erie Avenue, Philadelphia, PA 19134. It is sensitive to touching an obstacle (and will automatically back away) but otherwise has none of the smarts of the Terrapin version and doesn't hook up to a computer. It costs $49.50.

If you are handy in the shop, you can buy just the motorized dual drive and gear train ($4.95) and wheels ($3.00 for the pair) and design your own turtle.

Conclusion

LOGO for the Apple allows you to participate in an active and exciting area of present day computing and education.

I wish to thank Paul Trap for drawing the cartoon that accompanies this article.

You may contact Professor Carlson at the Physics Dept., Michigan State University, East Lansing, Michigan 48824.

MICRO

Short Subjects/Education

The Computer in Special Education

By Florence M. Taber, Ed.D. and Alonzo E. Hannaford, Ed.D.

The microcomputer provides a valuable key to unlock individualization in the classroom. Let us look at where special education is in its use of microcomputers and how this tool can assist handicapped individuals, and then project our thoughts into the not too distant future, for the microcomputer is changing education. Like the snowball rolling down hill, its effects are growing at an ever increasing rate.

Probably the greatest and most exciting use for the microcomputer in special education today is seen with those whose handicaps involve communication. For example, people who have never been able to talk are doing so through a voice synthesizer. A very thrilling experience occurred about a year ago at a conference in Grand Rapids, Michigan, when the main speaker delivered his presentation through the use of a voice synthesizer that was connected to a microcomputer attached to a wheelchair. The speech was excellent and easily understood. Menus provide other handicapped persons with speech whereby they only have to indicate which word they want to communicate by a head pointer (a device like a headband with a lightpen attached) or a joystick.

The microcomputer is also allowing handicapped people to live more independently. Programs allow the handicapped person to answer the phone or doorbell from a distance, cook meals, sense darkness and adjust light accordingly, and communicate emergencies like break-ins or fire to the appropriate agency.

Those who are hearing-impaired can use a program to "visualize" speech, both in person or on a telephone. Programs are also available that graphically portray speech as a pattern. This permits the speaker to adjust speech to match the stimulus pattern and thus to develop better speech.

The visually impaired can use the microcomputer in another way, by turning print or touch into speech. For example, the Kurzweil Reading Machine uses a visual scanner to turn the print on pages into speech. Talking calculators and typewriters have been around for a number of years.

Besides uses already indicated, special education can use the microcomputer to reach students. For example, in a large metropolitan area in the southeastern part of the United States, a bright young teacher of autistic children taped her voice to be similar to that of a robot and used it with a microcomputer software program. Eventually, she withdrew the computer and increased the children's communication with the outside world. Emotionally disturbed children have also perceived the microcomputer to be non-threatening, and have accepted learning through educationally-effective Computer Assisted Instruction (CAI). Through flexible branching (the capability of the microcomputer to present material designed to reach specific objectives at various functioning levels), students who have difficulty learning can be presented material they can comprehend. In other words, after the objectives are set for a class, each learner can be taught at his own individual learning rate, and reading and conceptual level.

The classroom of the future can totally individualize rates and approaches, leaving the teacher free to design and evaluate programs and interact with learners based on their needs. When objectives are established and mastery of levels set, all students — retarded through gifted — can progress successfully through the school curriculum. No student will proceed to the next skill or concept until mastery level is achieved.

But, although technology is heading in this direction, society is not ready to move that fast, and costs are still high for many specialized programs and microcomputer adapters. On the other hand, it will happen! And as teachers, parents, citizens, we can all help this change take place for the educational benefit for all.

Dr. Taber taught special education for fourteen years prior to receiving her doctorate in education from Western Michigan University. Since that time she has directed the learning disabilities teacher education program at Nazareth College and presently is Director of Educational Prescriptions, Inc., an educational assessment clinic, and is the Director of MCE, Inc., a company that develops software for the special needs audience.

Dr. Hannaford taught industrial arts for special education students prior to receiving his doctorate from University of Northern Colorado. He is presently a professor at Western Michigan University in the Special Education Department.

The authors may be contacted at MCE, 157 S. Kalamazoo Mall, Kalamazoo, MI 49007.

Chemistry and the Microcomputer

By Judy Toop

The use of microcomputers in classrooms is becoming more commonplace nationwide. Chemistry students at the Auburn Adventist Academy in Auburn, Washington, have been using an Apple II Plus for one year now. It has not been used every day, nor for every subject covered in the Chemistry curriculum. But it has caused some interesting and significant reactions by students. And there have been some surprises.

I started using the Apple II Plus in my Chemistry classes in late October. I began tentatively, unsure whether it would be best used for individualization or by the entire class at once.

We first studied the oxidation numbers of elements, which students had to memorize as part of their homework. I originated a simple program that presented the element and gave the student a chance to type in the oxidation number. As soon as the monitor was turned on, the students clustered around, each apparently wanting to be the first to give the correct answer. I typed it in and they were rewarded with a compliment. It was simple drill, but they took to it eagerly.

A questionnaire given out after six months of computer use helped to clarify student feelings about the computer. A majority (69%) perceived the use of the computer as improving their education. Sixteen percent of them perceived it as a method of saving teacher time, and another 15% thought it was a fad that would pass.

The computer made learning easier for 73%, while 3% thought it became harder. The remaining 27% suggested that using the computer was about the same difficulty as other methods of learning.

Since the programs used were teacher-made, I asked them about the quality of the programs themselves. Eighty percent thought the programs were interesting, 18% thought they were rather dull, and the remaining 2% were sure the programs were just too complicated. Thirty-two percent of the respondents would like to help make up some chemistry programs. Forty-one percent conceded that the programs needed some improving and 24% suggested that we get some new, commercially distributed programs. Good suggestion. Now if we can locate the best ones at affordable prices, we'll be on the way.

There may be several reasons for the positive acceptance of the microcomputer in our Chemistry classroom. The screen presents one idea all at once, with several lines of data or diagrams. The student can read at his own speed, and go back over an idea right then, before reading the remainder of the screen or the next screen. The slower student may read it through once, the quicker student reads it several times and comprehends the idea thoroughly. By contrast, writing the same information on an overhead is slower and may try the patience of the quicker students who read much faster than the teacher can write.

Use of the computer allows the teacher to pace the class more efficiently than in a lecture. With a computer, student comprehension is checked immediately. In a lecture it is all too easy to go on, assuming everyone understands the material, only to find out the next day that you lost them in the first three minutes.

A second use of the computer in the classroom may be even more significant: make-up work. There are days when you present a vital concept that takes some time to explain with appropriate illustrations and information. But one or more students may be absent that day. They miss that presentation, so are completely lost on the subsequent topic when they return to the classroom. This presents an especially serious problem in math and science where information builds on previous learning. If the teacher has time, a tutoring session can catch the student up on missed work in a matter of a few minutes. But if the teacher's free time and the student's available class periods do not coincide, then it is likely that frustration will build as further classes attended are meaningless and the student falls farther behind. The computer can remedy this problem. The teacher hands the student a diskette, indicates the program to be run, points him to the computer, and goes on with another class.

Although I am thoroughly sold on computer-assisted instruction in Chemistry, there are some drawbacks and cautions, based on my experience this year.

The video monitor has been a limitation. We use a black-and-white television set with a 26-inch screen. The quality of reproduction is not as good as it would be with a monitor-only piece of equipment, and the whole class can't see the screen.

In the questionnaire, two students suggested that the computer might be about to replace the teachers. Nobody else mentioned it, and I'm not sure that it is a probability. What I do know is that it can be a valuable aid to make classwork interesting and intelligible. It can be a student-saver in make-up work, helping to coach the timid and encourage the one who has missed some days. It can be a teacher-saver, allowing student tutoring by machine for general help, freeing the teacher for special help sessions with some other students.

The students believe the future of the computer includes having more of them and more use of them in all classes. They perceive them as being friendly and helpful, and an aid to the teacher. Several of them wish there were more units, so they could get some time on them just for review.

I hope the budget for next year includes purchase of a couple of new units for exclusive use in the Chemistry department. My experience and the students' reactions would indicate it would be money well spent.

Judith Toop is a Chemistry teacher at Auburn Adventist Academy, Auburn, WA 98002.

MICRO

One Teacher's PET

by Gloria Stein

The author, a fifth grade teacher, discusses the growing involvment of PET computers in her classroom. As long as the teacher remains involved and aware, the results are nearly always better than with more conventional methods.

Catapaulted into the electronic age by a former student who burst into my classroom demanding, "Mrs. Stein, you *have* to get a terminal so I can show your class...!", my computer expertise manifested itself in my reply: "OK, Bruce. Calm down. (Pause) What's a terminal?"

I found out. Soon, the clanking terminal prattled messages from the computer while excited fifth graders jabbered at each other, determined to produce the correct response required by the program. I marveled at their heightened motivation. Previously, these same students had not *begged* for spelling, grammar, and math.

The fact that the judicious use of programs is imperative if the computer is to provide a genuine learning experience became patently clear at the onset. The students, who had pencil/paper experience with different math bases, located a program that computed problems in bases other than ten. When I put this program off-limits because "the computer does all the work," the children countered with self-assigned homework. Many did ten to thirty non-base ten problems a night, declared they were working, and pleaded to use the computer program to check their computations. The groans of agony heard when the computer indicated they were in error formed a sharp contrast to the resigned silence with which these same students received a corrected paper/pencil assignment from me with incorrect answers checked. These children convinced me the computer is an incredible learning tool.

That was six years ago. Now, Lawton School houses six perky PETs used in some way at most grade levels. Daily, the PETs, on their special carts, zing down the halls propelled by eager computer pushers heading for a specific classroom. But that didn't happen overnight.

During the 1979 school year, Lawton was designated as one of two Ann Arbor schools to pilot the use of micros at the elementary level. The first PET arrived late in the year, and while the children embraced it at once, the staff was reluctant to face the new monster during that busy season. That allowed time for the PET to train me. Consequently, I was one step ahead of the rest which, in education, automatically qualifies you as a resource person. Anxious to share a good thing, I wrote a beginner's user guide for the staff, trained computer tutors to work with primary students, and produced programs using data specified by the teachers. This involvement allowed me to witness the introduction of micros at all grade levels. No observation attenuated my original conviction regarding their inordinate value.

By the end of the two-year pilot program, however, I had become adamant on one point. For the micro to be truly effective, a teacher must have a PET and a selected set of programs available on a full-time basis. During the 1981 school year, on an experimental basis, my students and I enjoyed such luxury, which afforded the implementation of numerous applications.

Drill and Practice

"Don't use a computer like an electronic workbook." Variations on that theme are often heard from educational computerists who work in lofty places. I have yet to hear it from an elementary teacher who is directly accountable for students learning all that dull stuff.

Mea culpa! I believe drill and practice is an excellent use of a micro — if needed. Most children regard workbooks as exciting as mushroom and spinach pancakes, and many care not whether their output is correct. They "do the work" but do not learn.

To test this theory, I once reviewed a language-arts concept in depth. A day or two later, I assigned the related workbook page. As always, the concept involved was clearly printed at the top, followed by simple directions. When finished, I asked the children to close their books and identify the subject matter. Not one out of twenty-eight bright children could do so. Each had immediately spotted a pattern, followed that pattern, skipped the title and directions and, consequently, gained nothing.

Not so with the PET. My most recent battle was fought on the Field of Punctuation. After workbooks and dittos galore, the same habitual errors persisted in student's written work. (It is appropriate to note that if traditional workbooks and dittos really produced, the children would have mastered basic punctuation prior to fifth grade.)

So, the PET was deployed amid the fray. As in the "workbook test," I merely assigned a specific program. Whatever appeared on the screen, including directions, was read with intense concentration. Each one cared and was motivated to succeed. Many of the first scores were dismal. Some correctly punctuated only two out of twelve sentences; none achieved a perfect score; all requested a second chance. Magnanimously, I consented. On the second round, the lowest score was ten out of twelve. At this point, I introduced a second program with new data. Virtually everyone attained a perfect score the first time. Meanwhile, their written work reflected upgraded skills. I am hard pressed to apologize for using the PET for drill and practice when it is unequivocally indicated and success is achieved.

In addition to concentrating on subject matter, precision and awareness are spin-offs. Careless errors may be legion in a given student's paper, but when dealing with the PET, perfection is the only goal. In one punctuation program, many students did not space after a semicolon. When told they were incorrect, they bristled, "Mrs. Stein, there's something *wrong* with this computer." I explained the program demanded the space but I did not regard this as a punctuation error. The children did. Nothing to do but go back and try to match the PET's precision. As their teacher, I do not mind viewing this phenomenon one bit.

The most graphic example of computer-engendered awareness occurred when I worked with a younster who used non-standard English. Eager to learn, she responded to my written or verbal corrections but there was no carry-over in conversation. I wrote a simple, multiple choice program using her own statements as data. Working with the PET, she was forced to think about grammar and chose the correct response each time. At her request, she worked through "her" program several times during the next few days. Subsequently, she began to correct herself.

Gerald T. Gleason, University of Wisconsin, wrote an extensive article for *Educational Technology* (March 1981). Referring to his investigation into then current research in computer-assisted instruction, he makes three summary statements that deserve reflection:

"1. CAI can be used successfully to assist learners in attaining specified instructional objectives.

"2. There appears to be a substantial savings in time (20 percent to 40 percent) required for learning as compared to 'conventional' instruction.

"3. Retention following CAI is at least as good if not superior to retention following conventional instruction."

Three Is Not A Crowd

Contradicting the common assumption that micros in education find one silent child facing the PET, research directed by Dr. Carl Berger, University of Michigan, indicates that the optimum number is three. The interaction among the students while working with the PET elicits increased learning. Two or four students showed greater gains than one or five. These conclusions are easy to confirm during classroom use.

Programs demanding problem-solving skills are a natural for small groups. Appropriate for all students, this use is particularly applicable for gifted children. Bright youngsters are often denied dialogue with each other because of the nature of the school's set curriculum. Completing an assignment long before the others, they pick up a "free-reading" book, often unchallenging. Gather a few of these children, give them a PET, and all is changed. Observing their reactions to the micro and each other, I watch their brains wake up. This is a far more scintilating experience than the enrichment acitivities normally provided by teachers.

Social studies simulations produce increased interest in history, geography, and economics — subjects with which the average fifth grader is less than enchanted.

Language arts "games" often provoke thoughtful use of phoneme/grapheme relationships and, no matter what the discipline, reading is upgraded because the students read for their specific purpose, as opposed to a reading assignment dished out by the teacher.

Students enjoy working with each other in any discipline if the program uses a quiz format. Again, the goal is to "beat the computer" and I have yet to meet the group who is satisfied with a passing grade; a perfect score must be attained.

Everyone In The Pool!

When the PET is combined with a large monitor, the entire class benefits from its magical, stimulating capacities. It provides maximum use of computer time, allows for a high degree of teacher input and, for a blessed change, low teacher visibility. With one or two students manipulating the keyboard while the others concentrate on the monitor, the teacher can be in back of the class. Because of the PET's impartiality, the children's eager acceptance of the micro and a generally relaxed atmosphere, learning is achieved in a completely non-threatening almost social setting.

Further, the PET/monitor provides ideal conditions to fight, en masse, specific hang-ups. For example, I have yet to meet a new group of fifth graders who have not internalized its self-ordained thirteenth commandment: "Thou shalt be good in math only if thou art the first one finished." A story-problem program helps. When the story appears on the screen, each student's task is to write the equation needed to solve the problem. No answers allowed! Student equations are discussed and the correct one identified. Only then is the solution entered to gain approval from the PET. The children become far more thoughtful about mathematical concepts.

Subject matter may be any discipline. Many programs tempt the user to produce the correct, or optimum, results in the least number of turns or in the shortest possible time. Problem-solving skills and logical judgements based on knowledge of the subject must be employed. I have often seen insecure students enthralled with this activity who would normally reject the same material if presented in written form because "It's too hard" or, "I don't get it."

Last spring during a special Community Day, the president of our Chamber of Commerce visited the classroom while the children used the PET/monitor and a program that demanded sharp problem-solving skills. A few days later, I received a letter that included the following: "I was very much impressed with the students' ability to interact with one another with the computer acting as the focus of their problem solving. If various community and corporate boards could approach problem solving with the same focus, enthusiasm, and spirit of cooperation, we would indeed live in a better world."

PET Specials

On occasion, Lawton's PETs are gathered in the classroom for special projects. One in particular challenges the children to pull forth their keenest problem-solving skills. Five computers, armed with the same social studies simulation, are manned by small groups. Each group not only strives to outwit the program but also to produce better results than the others. Sometimes I can almost detect steam wafting from the ears of the working brains.

An afternoon soiree is fun. Again, all computers are used but each with a different, carefully selected program that addresses concepts currently, or previously presented. Small groups are assigned to each computer. After a specified time (thirty minutes is comfortable) the groups rotate to the next PET. As usual, the children apply greater concentration, work diligently for correct responses, are far more excited, involved, and motivated. Succinctly, they work hard. When they meet their friends after school, however, I invariably hear remarks such as: "We played with the PETs all afternoon. We didn't do any work at all." I never spoil it by telling them otherwise.

Late in the school year, many students need a major review of math concepts. On occasion, my fifth grade counterpart and I have split the grade level depending on needs. One of us provides the review; the other offers enrichment activities using the PET/monitor. The groups are not static. A student may opt into the review group for one or two concepts and then join the enrichment group. It is interesting to observe the students who need review because of previous poor application on their part. Suddenly it becomes very important to master a concept in order to "get to work with the computer.

Software is Hard

The hitch in this positive picture is software, because ninety percent of all educational software is trash. Variations on that theme are heard whenever three or more gather in the name of educational computing. I question the high percentage since it is what you do with the program that counts. Teachers often face the challenge of using a poor text in such a way that a good learning experience is provided. So it is with computer programs. The day is gone when I preview a program and reject it forthwith. I have learned to question what intrinsic merit it may have. For example: "NAME THE ELEMENTS." Fifth graders should learn the chemical symbols for some common elements but the identification of the obscure ones becomes an excellent dictionary exercise. In this fashion, I've used many programs advantageously.

The real software problem is lack of time. Previewing is a must; promotional literature merely puts you in the ballpark. This fact of computer life is so obvious Ann Arbor is considering an Elementary Software Committee whose sole function will be to preview programs, annotate those of merit, and recommend purchase. Hopefully, this will relieve individual teachers from this time-consuming responsibility and facilitate the procurement of effective programs for our classrooms.

Conversely, some of my most succesful programs are public domain, usually written by teachers or amateur programmers. Disseminated through user groups or shared teacher-to-teacher, I'm always on the alert for these.

Professional or not, I use few programs without changing them in some way. The simple removal of directions when no longer needed saves LOAD/RUN time and eliminates student frustration.

During Lawton's pilot program, we discovered children do not care if there are graphics in academic programs. The removal of same is also a time saver.

Few programs make use of the PET's internal clock. A mere two-line addition to a program adds another motivational dimension. It's fun to be told how long it takes to achieve a specified goal and, gratifyingly, the students' self-imposed task is to RUN the program again and beat their own time.

Think Computer

I must always "think computer." Without constant teacher supervision/input, a classroom computer can be reduced to a meaningless toy in a matter of days. Non-academic programs — true games — are bootlegged in or nonsense programs written to the benefit of none. Have you ever watched children at the blackboard during indoor recess? You don't find weighty academic achievement there, either. Micros cannot be learning tools if parked in the room and the children turned loose, any more than learning can be anticipated by merely plunking children in a roomful of books. This fact in no way diminishes the impact micros will have on education and knowledgeable computerists should help spread the good news.

Adults are conditioned to feel that education cannot "be fun." Micros have changed all that. To the children, the most exacting program is a game. Adults, denied the opportunity to see the increased investment of attention, concentration, and effort elicited by the PET, suspect the youngsters are fritterin' away their time.

Recently, it was necessary for a substitute to replace me. My plans clearly stated that each child was to work through a given program. To my chagrin, the substitute told the students they "must have their work done" before they could use the computer. My *priority* item had been the work with the PET. Kids are fine. Others need education in "thinking computer."

Here To Stay

Over the years, the PETs have lost none of their electronic charisma. The PETs have multiplied and so has student interest. We like our PET. It is cute, efficient, exciting, and durable to the point of being care-free. It is mobile, portable, and bless its little CPU, easy to program.

One of our PETs has the original calculator keyboard. Most of the key labels are worn away. That's use. A keyboard guide was pasted under the sceen. Several of my students refer to it only rarely. They have the keyboard memorized. That's motivation!

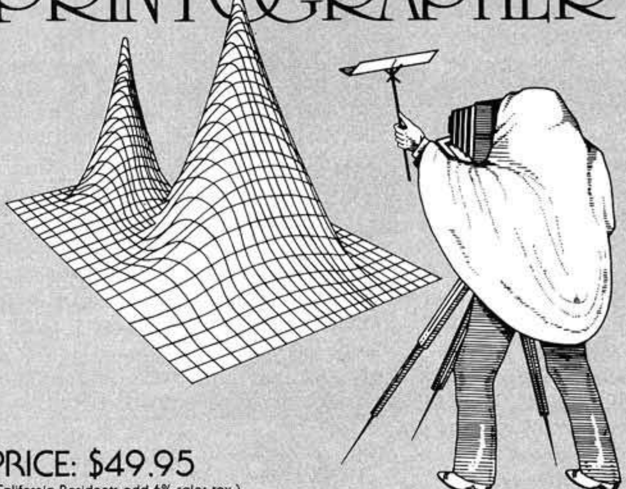
Mrs. Stein holds an AB in Sociology and a Master's in Social Work. She teaches fifth and sixth grade in Ann Arbor, MI. You may contact Mrs. Stein at Lawton School, 2250 S. Seventh St., Ann Arbor, MI 48103.

MICRO

TIRED OF ALL THE "EXCEPT FOR. . ."S?

WITH THE NEW PRINTOGRAPHER GRAPHICS PRINTING SYSTEM, WE'VE GOT YOU COVERED!

PRINTOGRAPHER™



PRICE: $49.95
(California Residents add 6% sales tax.)

The **PRINTOGRAPHER** is designed to fill all of your graphic printing needs, without having to worry about running into the problem of it almost working, "except on your printer", or "except for the lack of that particular feature". Whether you have a daisy wheel or dot matrix printer, the standard version of **PRINTOGRAPHER** works on any printer and interface combination with graphics capabilities. In many cases, this includes printers you may not even have thought could print graphics.

Just a FEW of the possible printers include: EPSON, PAPER TIGER, ANADEX, NEC, DIABLO, QUME, MPI, SILENTYPE, OKIDATA, MALIBU; interface cards include: APPLE, SSM, CCS, MTN COMP. CPS, MPI, GRAPPLER, TYMAC, PROMETHEUS and more!

In addition to versatile print options (easy cropping, variable magnifications, normal/reverse inking, vertical/horizontal format, etc.) **PRINTOGRAPHER** offers such unique features as the ability to print pictures directly from disk (without loading a file), spooling via our **DOUBLETIME PRINTER** package, or sending pictures over a phone line using **ASCII EXPRESS**. You can even put graphics in your text documents with our text editor software, **THE CORRESPONDENT**. As if that wasn't enough, we've made it easy to put the **PRINTOGRAPHER** routines right in your own programs to do Hi-Res printing immediately during their operation, without having to save screen images to disk!

We also know you see a lot of advertising these days for a truly overwhelming volume of software, all claiming to be the best, so we make this simple guarantee:

IF YOU CAN FIND A BETTER PACKAGE THAN (OR ARE AT ALL UNSATISFIED WITH) THE PRINTOGRAPHER WITHIN 30 DAYS OF PURCHASE, SIMPLY RETURN THE PACKAGE FOR A COMPLETE REFUND, NO QUESTIONS ASKED!

For more information, see your local dealer, or write **SOUTHWESTERN DATA SYSTEMS** for a free catalog. If your dealer is out of stock, we can ship **PRINTOGRAPHER** to him within 24 hours of a call to our offices.

REMEMBER: WITH PRINTOGRAPHER, YOU'RE PICTURE PERFECT!

southwestern data systems™

10761-E Woodside Avenue • Santee, California 92071
Telephone: 714/562-3670

ARK COMPUTING

Your Salvation In The Sea Of Inflation.

P.O. Box 2025 • Corona, CA • 91720

Computer Stop 16K Ram Board
$79.95

This high quality 16K Ram board acts like a Language Card when plugged into slot 0 of your APPLE II. Compatible with Basic, DOS 3.3, CP/M, PASCAL, LISA 2.5, and VISICALC. With this card you get high quality, low price, compatibility, and a 1 Year Manufacturer Warranty all for $79.95.

LAZER Lower Case +Plus III
$34.95

The best lower case adapter available for the APPLE II. This feature packed board has twice the features of competing boards. The Lazer Lower Case +Plus III is the only lower case adapter that works with VISICALC and is recommended by Stoneware for DB MASTER. The Lower Case Plus III is expandable to 4 character sets (2 on board), has inverse lower case, includes ASCII and Graphics, and is compatible with most word processors. NOTE: For REV. 6 and earlier order Lower Case +Plus at $44.95.

LAZER Keyboard +Plus
$69.95

The Lazer Microsystems Keyboard +Plus has a 64 character type ahead buffer. The buffer can be cleared or disabled. The Keyboard +Plus lets you use the shift-key as a typewriter shift-key, allowing you to enter directly the 128 ASCII character set from the APPLE keyboard. The Keyboard +Plus may be installed on any APPLE II.

Computer Stop Omnivision
$129.95

Looking for an 80-column card? Look no further! Now all your Basic, CP/M, and Pascal programs can take advantage of a full 80-column display.

Wizard-BPO
(Buffered Printer Interface)
$139.95

This outstanding parallel interface card comes with 16K of Ram installed, expandable to 32K This card also has Graftrax-like features. So now you no longer need to wait to use your APPLE II while your printer is busy. Compatible with Basic, CP/M, Pascal.

Microtek Parallel Printer Interface
$59.95

This popular printer interface card, manufactured by Microtek, is a steal at $59.95. The printer card comes complete with a cable and a Centronics compatible connector (Amphenol). Works with basic, CP/M, and Pascal. This card also has graphics capabilities.

ORDERING INFORMATION

We accept: VISA/MASTERCARD (include card #, expiration date, and signature), Cashier or Certified Checks, Money Orders, or Personnal Checks (please allow 10 business days to clear). We also accept COD's (please include $2.00 COD charge).

Please add 3% for shipping and handling (minimum $2.00). Foreign orders please add 10% for shipping and handling (minimum $10.00).

California residents add 6% sales tax. All equipment is subject to price change and availability without notice. All equipment is new and complete with manufacturer's warranty.

(714) 735-2250

Microcomputers in a College Teaching Laboratory — Part I

by Howard Saltsburg, Richard Heist, and Thor Olsen

At the University of Rochester, microcomputers save students time and the college money. In the first article of a four-part series, the authors give an overview of the involvement of microcomputers in their teaching laboratory. The remaining articles will cover in detail some specific applications.

Typically, the use of computers in an engineering curriculum involves numerical solution of mathematical problems. In many cases the complexity of these problems makes the use of anything other than a large computer impractical. Mainframe or minicomputer use in a laboratory environment, on the other hand, is often precluded by the cost per station and the (perceived) need for complex and expensive interfacing. In addition, the intricacy of many time-sharing computer operating systems tends to discourage laboratory use.

The evolution of the microcomputer into a powerful, relatively inexpensive, stand-alone device has changed the complexion of the problem for laboratory use by forcing a clearer examination of what one really needs from a computer in such a situation. The problem still remains, however, of exploiting the potential of the microcomputer with only minimal funds and without resident expertise.

For the past three years the Department of Chemical Engineering at the University of Rochester has been developing an undergraduate laboratory program built around the use of the microcomputer. This article provides an overview of that program.

Objectives of the Program

The primary objective in installing computers in the laboratory was to improve the scientific and chemical engineering content of the laboratory experience. This was accomplished by reducing the tedium of extensive data acquisition and data reduction, and through the introduction of experiments that had not been feasible without computer assistance. At the same time, we expected the laboratory experience would demonstrate to the students the more general utility of the computer as an engineering tool.

Selecting the Computer

A review of the laboratory program made it clear that most experiments involved slow acquisition of modest amounts of relatively low-precision data. Thus, it was clear that a large computer with massive file storage capacity and high-speed computation capability was really not necessary. Given the poor ability of a multi-user system to accommodate the real-time needs of a laboratory, and the high costs of terminals and linkage, the microcomputer appeared to be the best choice for our application.

For the program to be successful, the chosen system would have to be user friendly; beginners had to be able to work with it effectively after only a short introduction. This requirement eliminated the single board computer as a possible choice. The self-contained microcomputer with built-in high-level language, machine coding capability, and sufficient I/O facilities, seemed to meet our needs most adequately. The fact that these devices are highly interactive made them even more desirable for the teaching situation.

The final choice of hardware was dictated both by intrinsic features and by the need for a cost-effective solution. Program and data file storage had to be reliable, and the high cost of disk drives made a tape-based system desirable. The need for easy I/O operations, simple file manipulation, and machine coding capability with easy interfacing to a high-level language, all combined in a sturdy, readily transportable unit, led to the choice of the Commodore PET. The subsequent introduction of an educational discount made the PET even more attractive. Although the lack of high-resolution graphics appeared to be a problem initially, this has proven not to be the case. It has been sufficient to display crude (quarter-cursor) graphics on the video screen. When high-resolution output is needed, inexpensive dot-

University of Rochester Chemical Engineering students during a laboratory session. The three computers in the foreground are each being used with a separate experiment to make temperature measurements. Each of the four computers in the background are interfaced to gas chromatographs and are being used for data acquisition and analysis. Both applications involve the QM-100-amplifier combination discussed in the text for A/D conversion.

matrix printers are used to make it available as hard copy.

Interfacing

A common barrier to laboratory use of computers is the problem of getting analog signals into the computer and using digital signals to control analog devices. The design of the sophisticated converters needed for high speed and high precision is not simple, but when the parameters of an actual system are carefully considered, the problem often becomes more tractable. Typical outputs of analog transducers in use were voltages that had been monitored with chart recorders and voltmeters. However, most experiments in our laboratory actually required only slow (<10 Hz), low-resolution (8-bit) data conversion. (Interestingly, many industrial and research applications have similar characteristics.) Thus, while A/D and D/A conversions are essential to the laboratory program, their level of sophistication or complexity need not be very high.

With only limited electronic experience, and no desire to become electronics experts, what was needed was to find a literature intended for the builder and user, rather than for the designer of interfacing devices. The microcomputer magazines filled this need and proved to be a rich source of information on both hardware and software.

For instance, a three-channel A/D converter utilizing a pulse-width technique is available (advertised in microcomputer literature) for less than $100.00 (QM-100, Analog Systems, Tucson, AZ). The QM-100 provides 8-, 10-, or 12-bit resolution with conversion rates up to 50 per second — more than fast enough for most of our purposes. The design utilizes the internal counters of the computer to measure elapsed time between the start of a voltage comparison (input *vs.* QM-100-generated linear ramp) and the point of match. Only two bits on the I/O port are required per channel, one to trigger the converter, and one for the end-of-conversion signal. The basic circuit board of the QM-100 comes already assembled, so making the final connections and adding a power supply is simple, even for the beginner.

In many typical applications for such a converter, e.g., reading a thermocouple voltage, the transducer signal is too small for the A/D converter, often by a factor of 1000. In such cases, an operational amplifier must be added to the analog circuit to bring the input voltage into the usable range. Again the microcomputer magazine literature enabled us to construct simple devices to provide the needed signal boost.

Since temperature is one of the important parameters of chemical engineering experiments, an article in MICRO (26:31) describing the use of the 555 timer integrated circuit as an inexpensive A/D converter for resistance transducers, including thermistors, was particularly helpful. The use of thermistors instead of thermocouples resulted in significantly lower cost of the temperature-measuring interfaces, and the need for signal amplification was eliminated. The 555 interface also uses a pulse-width technique, so the necessary software required little change from that used with the QM-100. Moreover, the addition of a single integrated circuit, a multiplexer, made it possible to construct a 16-channel digital thermometer and still use only one 8-bit I/O port. Since most of our experiments either are steady-state or involve only slow changes with time, the data acquisition rate is not important.

Other articles dealing with D/A converters based on a resistance network (R-2R) and AC power control, both on/off and variable output, have made it possible to add these features to several experiments.

The Laboratory Program

The undergraduate laboratory program begins in the sophomore year with a course designed to provide background material appropriate to understanding how the computer works and how it can be used in a laboratory environment. An introduction to digital electronics is provided and used to discuss the operation of the computer and interfacing devices. The structure and logic of programming languages are also discussed, and aspects of computational techniques are introduced and practiced.

Initially, laboratory sessions provide practice in computer use: programming, file handling (data and program), controlling the I/O ports and special features, such as the built-in clock. Next, elements of machine-language programming are covered in sufficient detail to enable the student to write programs for simple data acquisition and manipulation.

The first I/O experiment utilizes an array of eight light-emitting diodes to map the parallel port output. This LED array has proven to be a very effective teaching tool. The message is understood very quickly: if you can selectively control these LEDs, you can control anything that can be switched electrically.

One section of the course is used to introduce structured programming languages, rather than languages such as BASIC and FORTRAN. The desire to avoid using disks in the laboratory has restricted the choice of languages. Since it was important that the language taught in the introductory course should be available for the students' future laboratory work, the University of Waterloo Structured BASIC (available in ROM for the PET) was chosen. Both Pascal and FORTH are available to interested students, but on a somewhat limited scale.

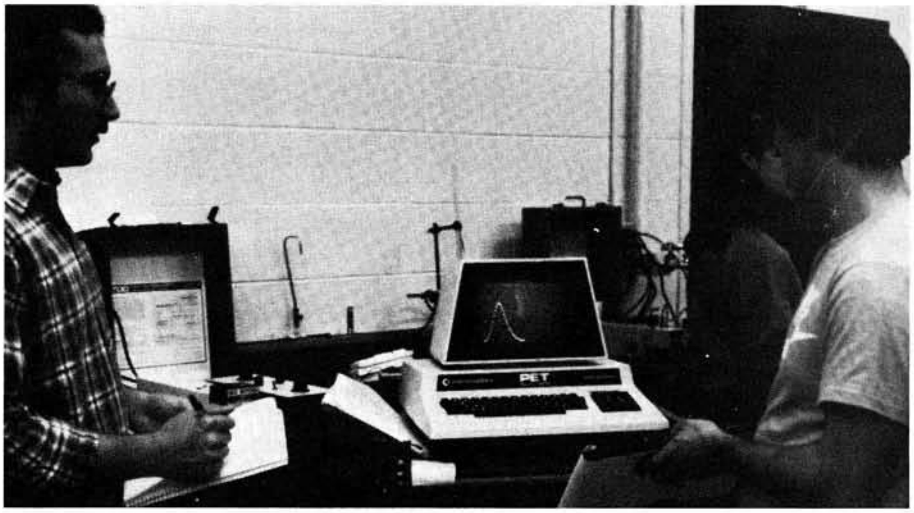
The introductory course concludes with an experiment demonstrating how a computer might be used to control a chemical process device. Although chemical engineering students typically do not see control theory until they are upper classmen, a simple laboratory experiment was developed to illustrate elementary forms of process control. The device, a small, recirculating air heater, is self-contained and requires only standard AC power. The electronics consist of a 555-thermistor temperature sensor and an opto-coupled triac for AC power (heater) control. The dynamics are plotted on the screen or with a dot-matrix printer so that the temporal behavior of the air bath temperature can be readily observed, and direct comparison can be made of alternative control strategies. The students are expected to write their own versions of the high-level language applications program (the control algorithm). They are given the machine-language routine for data acquisition and the thermistor calibration data to save time.

Subsequent laboratory courses involve typical studies of engineering principles and processes. Heat-transfer experiments that involve measurement of a steady-state temperature have been simplified considerably, as suggested in the previous discussion of interfacing. One study of transient heat transfer illustrates how the computer can enhance the engineering content of an experiment. Previously, you could only study slow transients because the experimental conditions were limited by the response time of the chart recorder. With the computer and a relatively fast commercial A/D converter used for data acquisition, transient responses over a few seconds can be followed in detail, and the digital data can be stored on magnetic tape, readily available for future analysis. Thus, phenomena that could not be resolved by conventional laboratory equipment, can now be routinely studied.

Chemical reaction experiments constitute an important part of our laboratory program. One study, in which the progress of a chemical reaction was followed by observing the fading of a pink color, required a simple, but costly colorimeter. By replacing the colorimeter with a green LED and a photo resistor — and using the 555 A/D converter — it was possible to interface the experiment to the computer and, at the same time, substantially reduce the cost of the apparatus. The colorimeter is now so simple that it requires no maintenance.

The gas chromatograph, an instrument designed to carry out chemical analysis of mixtures of gases or vapors, is a mainstay of the analytical component of our laboratory. Conventionally, data reduction consists of measuring areas under curves produced by a chart recorder, a tedious task that is prone to errors. The computer is now used to both acquire and analyze the data on line, and the disadvantages are virtually eliminated. The interface is a standard op-amp and QM-100. A fairly complex program, written by a student, is used to carry out the data reduction. The results, which include areas and appearance (retention) times of peaks, as well as a plot of the curves, are available on the screen, or as high-resolution hard copy on a dot-matrix printer. With the tedium of data collection virtually eliminated, more attention can now be devoted to the interpretation of results.

Since the implementation of the undergraduate program, there has been increasing interest in use of microcomputers in the research laboratories. For instance, spectrophotometers used in research projects have been interfaced to computers using the op-amp/QM-100 combination. High-resolution printouts of optical spectra are comparable to the output of much more expensive instruments.



Chemical Engineering students using a computer interfaced to a gas chromatograph. The students use gas chromatographs to analyze mixtures of chemicals. The computer is used both to acquire and analyze the data. Using the computer saves hours over conventional methods.

The Universal Instrument

The combination of computer, A/D converter, graphics printer, and cassette recorder represents a kind of universal laboratory instrument. When it is advantageous to interface an experiment or an instrument to the computer, the same system can generally be used. The universality has simplified the ongoing addition of new experiments to the laboratory program. It has also made duplication of set-ups and maintenance of spare components cheaper — an important consideration in the face of rising engineering enrollment. An example is our recent adaptation of an experiment to study the dispersion of dye in liquid flowing in a pipe. Eight sensors of the LED-photocell type were clamped around a 50-foot, clear plastic tube. These sensors are interfaced *via* the standard 555 timer A/D to yield a system that permits continual monitoring of the dye concentration at different points along the flow path without disturbing the flow characteristics of the system.

The hardware standardization also simplifies the writing of new software, since most of a program needed for a new experiment can be taken from programs that already have been written. In the case of the dye-dispersion experiment, for instance, the only programming change required was to accommodate the number of sensors used and add appropriate calibration constants.

Concluding Remarks

The development of the laboratory program has been aided by the computer magazines, an often overlooked resource for both academic and industrial applications. Although several available books are very helpful, the articles in the popular microcomputer magazines are often more appropriate both for beginners who wish to use computer-aided laboratory operations without spending excessive amounts of time or money on the project, and for more experienced users who can take advantage of more sophisticated applications.

Although this article has stressed the undergraduate laboratory program, it is important to note that the successful operation of that program has had consequences outside the laboratory. It has created awareness of the power and utility of the microcomputer, both among students and faculty. Thus expansion of microcomputer use into classrooms and research laboratories has begun, and independent use of the computers by students is rapidly increasing. Business-oriented applications have also become more popular as the availability and ease of use of the system is repeatedly demonstrated. It is clear that the program is expanding in scope and popularity.

Details of the specific laboratory applications mentioned above, as well as other applications, will be discussed in future articles.

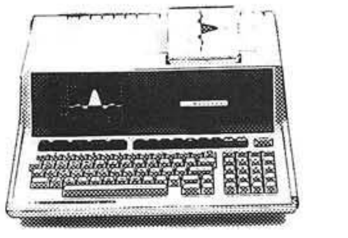
Selected Literature Resources

Magazines:
BYTE; COMPUTE!; MICRO, The 6502/ 6809 Journal; Microcomputing; The Transactor (Canada); *Popular Electronics, Radio Electronics.*

Books:
1. De Jong, Marvin L.: *Programming and Interfacing the 6502, with Experiments;* Howard W. Sams & Co., Inc.
2. Downey, James M. and Rogers, Steven M.: *PET Interfacing;* Howard W. Sams & Co., Inc.
3. Hampshire, Nick: *The PET Revealed;* Nick Hampshire Publication.
4. Leventhal, Lance A.: *6502 Assembly Language Programming;* Osborne/ McGraw-Hill.
5. Texas Instruments Learning Center Book Series; Radio Shack.
6. West, Raeto Collin: *Programming the PET/CBM, The Reference Encyclopedia for Commodore PET/ CBM Users;* Compute! Books.
7. Zaks, Rodnay, *Programming the 6502:* Sybex.

MICRO

HEWLETT PACKARD



HP•85 $1969

HP•87	$1799.00
HP•125	$1999.00
HP•85 16K Memory Module	$169.00
5¼" Dual Master Disc Drive	$1799.00
Hard Disk w/ Floppy	$4349.00
Hard Disk	$3549.00
"Sweet Lips" Plotter	$1199.00
80 Column Printer	$649.00

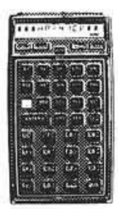
CBM 8032 $999

Commodore Business Machines

CBM 64	CALL
4032	$969.00
8096 Upgrade Kit	$369.00
Super Pet	$1599.00
2031	$529.00
8250 Doubled Sided Disk Drive	$1699.00
D9060 5 Megabyte Hard Disk	$2399.00
D9090 7.5 Megabyte Hard Disk	$2699.00
8050	$1299.00
4040	$969.00
8300 (Letter Quality)	$1799.00
8023	$769.00
4022	$499.00
Pet to IEEE Cable	$37.00
IEEE to IEEE Cable	$46.00
Tractor Feed for 8300	$240.00

Word Pro 5+	$319.00
Word Pro 4+	$299.00
Word Pro 3+	$199.00
The Administrator	$379.00
InfoPro Plus	$219.00
Power	$79.00

HEWLETT PACKARD



HP 41CV CALCULATOR $239

HP 41C	$189.00
HP 11C	$79.00
HP 12C	$114.00
NEW 15C	$119.00
NEW 16C	$125.00

HPIL PERIPHERALS IN STOCK!

Televideo Terminals

910	$579
912C	$699
920C	$749
925C	$749
960	$950
802	$Call
802H	$Call
816	$Call
806	$Call

VIC 20

$179

VIC 1530 Commodore Datassette	$69.00
VIC 1540 Disk Drive	$499.00
VIC 1515 VIC Graphic Printer	$339.00
VIC 1210 3K Memory Expander	$32.00
VIC 1110 8K Memory Expander	$53.00
16K VIC Expansion	$94.00
VIC 1011 RS232C Terminal Interface	$43.00
VIC 1112VIC IEEE-488 Interface	$86.00
VIC 1211 VIC 20 Super Expander	$53.00
VIC Mother Board	$99.00

EPSON PRINTERS

MX 80 w/Graftrax	$449
MX 80 FT III	CALL
MX 100	CALL
ADA 1600 Parallel Printer to CBM	$119.00
ATC-1 Parallel Printer to Atari	$29.00
AP-80 Apple Parallel Card & Cable	$69.00
IBM-1 Parallel Printer to IBM	$32.00

TEC PRINTERS

Starwriter	
F10-40CPS	$1399.00
F10-55CPS	$1749.00
Prowriter 8510A	$499.00
Prowriter 1550	$799.00
ADA 1450 Serial Printer to CBM	$119.00
ATC-2 Serial Printer to Atari	$29.00
AP-S10 Apple Serial Card & Cable	$95.00

Smith-Corona

TP-I

$649

Letter Quality
Daisy Wheel
12 CPS Text Printer

In-stock items shipped same day you call. No risk, no deposit on C.O.D. orders. Pre-paid orders receive free shipping within the continental United States with no waiting period for certified checks or money orders. All advertised prices include a 3% cash discount from normal credit card prices. NV and PA residents add sales tax. All items subject to availability and price change.

CENTRONICS PRINTERS

739-1	$519
739-3	$619
2 Meter RS232-RS232 Cables	$29.95

— ALSO —

Diablo 630 Special	$1799.00
Talley 8024-L	$1629.00
IDS Prism	CALL
Daisywriter	CALL

OKIDATA PRINTERS

82A	$469.00
83A	$599.00
84 Parallel	$1099.00
84 Serial	$1249.00

NOTE
Okidata 82A and 83A Printers come equipped with both parallel ports & RS232 Serial ports.

ADDITIONAL MANUFACTURER'S DISCOUNTS AVAILABLE TO QUALIFIED EDUCATIONAL INSTITUTIONS

800-233-8950 east

477 E.
THIRD ST.
Williamsport
PA 17701

IN PA
CALL
(717)327 9575

computer mail order east

A Personal Computer for Untrained Users

by Lynda B.M. Ellis and John R. Raines

The authors describe programs to assist users in operating a computer in an unsupervised environment. Human engineering concepts for the success of this system are considered. Although in Applesoft BASIC, they could easily be translated to other BASICs with cursor control.

MENU and HELP
require:
Apple II Plus
with at least 32K RAM and
one disk drive

Can the average person, without typing or other special skills, use a conventional personal computer system for the first time without human instruction? More specifically, can an unprepared person sit down at a strange computer, interact with it for a few minutes, and walk away with useful information? This person cannot be expected to read a manual, or even to follow computer-displayed instructions correctly at all times. Language skills and coordination may be poor, attention span may be short, and the environment may be distracting. However, the cost of a human instructor or assistant would be much more than the cost of providing access to the computer system.

We have developed and implemented a system that can be used with minimal supervision. It has been in use for more than two years and is well received by its users. We would like to describe the design features of this successful system.

Setting

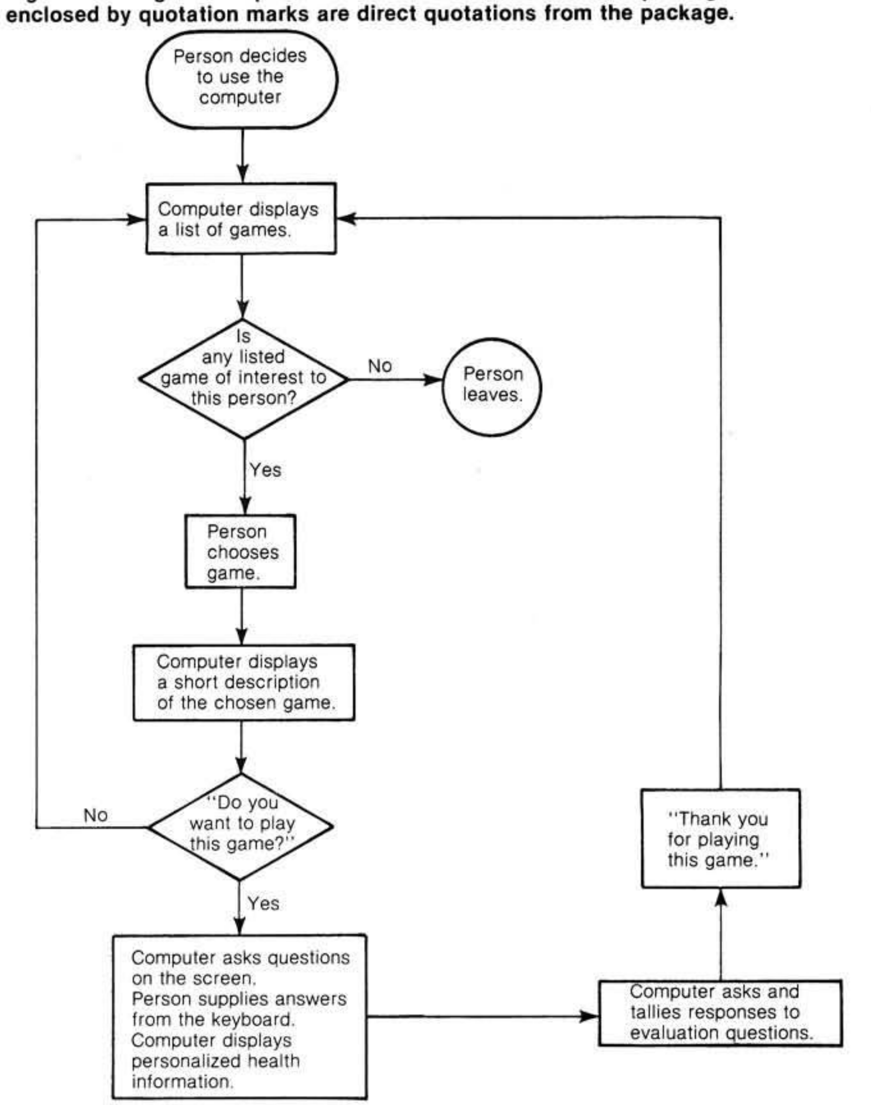
We have developed a health awareness package that includes six lessons on health topics. Within this package, a MENU program allows users to choose a lesson, then collects evaluation data on each lesson. A HELP program describes the computer and keyboard[1,2,3,4]. This package has been voluntarily used by hundreds of completely untrained people with minimal supervision and assistance. A user's encounter with the package is diagrammed in figure 1.

The computer system is located in the waiting room of the Community University Health Care Center, a general medical clinic serving a low income, urban community in south-central Minneapolis. Patients are free to use the system before and after appointments. A few are directly referred to specific lessons by their health care providers[4].

The patients are primarily under 40, and often un- or under-employed. Many have not completed high school. Approximately half are members of a minority group. A clinic receptionist oversees the computer and turns the system on at the start of the day. The receptionist's computer training consists of being shown the location of the power switches and being encouraged to use the system for herself.

The receptionist does *not* shoo

***Figure 1:* A diagram of a person's use of the health awareness package. The items enclosed by quotation marks are direct quotations from the package.**



children away from the computer; indeed, the youngest ones enjoy spelling their names on the TV screen. Without purposefully following directions, they don't get far into a lesson and the system can be re-started when they leave. The children may attract adults to the computer, as they tend to dispel any fear that the system will "bite."

In this setting, our system has been voluntarily used approximately ten times a day with minimal problems.

The computer system was placed on a typewriter stand, several inches lower than a standard desk, to improve keyboard access. An unanticipated advantage of this lower placement is that the TV, which rests directly on the computer, is less easily read by persons behind the user. Thus the system is somewhat more private, but remains in a prominent position in the waiting room. A colorful poster emphasizes its location. The system was placed in the direct line of sight of the receptionist, to prevent vandalism or theft.

We had two alternatives for a video display unit: a standard color TV set or black-and-white TV monitor. Color would certainly add visual interest to some lessons and would attract more attention than a black-and-white image. However, text is more difficult to read on a color TV, and presenting readable text was of greatest importance in this application. Also, a color TV might be more likely to be stolen than a black-and-white monitor, and a TV with a channel selector may be diverted to the watching of soap operas, etc. We have been pleased with our choice of the black-and-white monitor.

One system constraint not under our direct control was hardware reliability. To minimize down time, we did arrange for a complete maintenance contract, including use of a "loaner" machine in the event of a major problem. This loaner has not yet been needed; luckily, the only hardware failure in our first two years of operation was a burned-out power on signal lamp.

Software Design

The software design principles we have tried to use throughout the package will be discussed using examples from the programs. The package was repeatedly tested on volunteers with one of us looking over the user's shoulder (but trying to keep assistance to a minimum) before it was placed in the unsupervised clinic environment. We could not expect clinic users or the receptionist to fill out Software Problem Report forms. The package is written primarily in Applesoft BASIC, with

MENU Listing

```
10  GOSUB 63600: TEXT : SPEED= 255:C$ =  CHR$ (4):DR$ = "": IF FLAG < 1 OR
     FLAG > 7 THEN 200
15  DATA   "HELP","SMOKING","EXERCISE WEIGHT","LIFE EXPECTANCY","BIRTH CO
     NTROL","CORONARY RISK","LIFESTYLE"
20  FOR I = 1 TO FLAG: READ N$: NEXT
25  TEXT : DIM Z(7,4),C(7,5):F$ = "ACCESS DATA": VTAB 23: PRINT C$;"NOMON
     I,C,O"
30  HOME : PRINT "THIS ENDS THE ";N$;" GAME.": VTAB 5: PRINT "PLEASE HELP
     US TO HELP YOU BETTER.": VTAB 8: PRINT "HOW HELPFUL WAS THIS GAME F
     OR YOU?"
35  PRINT : PRINT  TAB( 3);"1  NOT HELPFUL, A WASTE OF TIME": PRINT : PRINT
     TAB( 3);"2  SLIGHTLY HELPFUL": PRINT : PRINT  TAB( 3);"3  MODERATEL
     Y HELPFUL": PRINT : PRINT  TAB( 3);"4  VERY HELPFUL, I'D RECOMMEND I
     T"
39  VTAB 19: HTAB 6: PRINT "HIT RETURN AFTER YOUR ANSWER.";: HTAB 1: INPUT
     D$:D$ =  LEFT$ (D$,5): REM  MORE THAN 20 DIGITS CAN CAUSE ERROR IN V
     AL
40 D1 =  VAL (D$): IF D1 < 1 OR D1 > 4 THEN  VTAB 20: PRINT  TAB( 5);"PLE
     ASE TYPE 1, 2, 3 OR 4": GOTO 39
45  HOME : VTAB 3: PRINT "WILL YOU CHANGE ANY LIVING HABITS": PRINT : PRINT
     "BECAUSE OF THIS GAME?": PRINT
50  PRINT  TAB( 3);"1  DEFINITELY NOT": PRINT : PRINT  TAB( 3);"2  PROBAB
     LY NOT": PRINT : PRINT  TAB( 3);"3  PERHAPS": PRINT : PRINT  TAB( 3)
     ;"4  PROBABLY YES": PRINT : PRINT  TAB( 3);"5  DEFINITELY YES": PRINT

54  VTAB 20: HTAB 6: PRINT "HIT RETURN AFTER YOUR ANSWER.";: HTAB 1: INPUT
     D$:D$ =  LEFT$ (D$,5)
55 D2 =  VAL (D$): IF D2 < 1 OR D2 > 5 THEN  VTAB 21: PRINT  TAB( 5);"PLE
     ASE TYPE 1, 2, 3, 4 OR 5": GOTO 54
60  PRINT : PRINT  TAB( 4);"THANK YOU FOR PLAYING THIS GAME."
62  PRINT C$;"OPEN";F$: PRINT C$;"READ";F$: INPUT DA$
65  FOR I = 1 TO 7: FOR J = 1 TO 4: INPUT Z(I,J): NEXT J: NEXT I: FOR I =
     1 TO 7: FOR J = 1 TO 5: INPUT C(I,J): NEXT J: NEXT I
72  PRINT C$;"CLOSE";F$
74  PRINT C$;"OPEN";F$:Z(FLAG,D1) = Z(FLAG,D1) + 1
75 C(FLAG,D2) = C(FLAG,D2) + 1
77  PRINT C$;"WRITE";F$: PRINT DA$
80  FOR I = 1 TO 7: FOR J = 1 TO 4: PRINT Z(I,J): NEXT J: NEXT I: FOR I =
     1 TO 7: FOR J = 1 TO 5: PRINT C(I,J): NEXT J: NEXT I
85  PRINT C$;"CLOSE": CLEAR :C$ =  CHR$ (4):
90  GOTO 200
100  HOME : VTAB 5: RETURN
110  VTAB 22: PRINT "PRESS RETURN TO CONTINUE";: INPUT D$: GOSUB 100: RETURN

120  VTAB 21: HTAB 4: PRINT "WOULD YOU LIKE TO TRY THIS GAME?": PRINT : PRINT
      " (TYPE Y OR N AND HIT RETURN) ";
130  POKE 216,0: INPUT D$:D$ =  LEFT$ (D$,1): IF D$ = "N" THEN  GOTO 200
140  IF D$ <  > "Y" THEN  VTAB 23: PRINT " PLEASE TYPE Y OR N AND PRESS R
      ETURN";: GOTO 130
145  HOME : VTAB 3: PRINT "NOW GOING TO  "E$" ....."
150  IF HT = 1 THEN  POKE 103,1: POKE 104,64: POKE 16384,0: REM  LOAD ABO
      VE HI-RES SCREEN 1
160  PRINT C$"RUN"E$
200  HOME :HT = 0: PRINT  TAB( 9);"HEALTH AWARENESS GAMES": PRINT : PRINT
      TAB( 6);"COLLECTED BY DR. LYNDA ELLIS": PRINT : PRINT "COPYRIGHT 19
      80, UNIVERSITY OF MINNESOTA"
205 GAMES = 7: REM  "MORE GAMES" 2 DISK OPTION IS CONDITIONED ON GAMES=8

210 P = 2:Q = 24
220  VTAB 10: PRINT  TAB( P);"1  EXPLAIN COMPUTER"; TAB( Q);"5  BIRTH CON
      TROL"
230  PRINT : PRINT  TAB( P)"2  WHY DO YOU SMOKE?"; TAB( Q);"6  CORONARY R
      ISK"
240  PRINT : PRINT  TAB( P)"3  EXERCISE WEIGHT"; TAB( Q);"7  LIFESTYLE"
250  PRINT : PRINT  TAB( P)"4  LIFE EXPECTANCY";: IF GAMES = 8 THEN  PRINT
      TAB( Q);"8  MORE GAMES"
260  PRINT
270  VTAB 22: HTAB 1: PRINT "TYPE THE NUMBER OF YOUR SELECTION.";
272 X =  PEEK (49387): REM  FIX FOR BUG IN APPLE PLUS
275  GET D$
280 I =  VAL (D$): IF I <  = GAMES THEN  ON I GOTO 6000,2000,3000,4000,80
      00,1000,5000,7000
325  IF D$ = "S" THEN  GET D$: IF D$ = "T" THEN  GET D$: IF D$ = "O" THEN
      GET D$: IF D$ = "P" THEN  END
330  VTAB 22: HTAB 1: PRINT "   PLEASE TYPE";: FOR I = 1 TO GAMES - 1: PRINT
      " ";I;: NEXT : PRINT " OR ";GAMES; SPC( 6): HTAB 33: GOTO 275
340  GOTO 275
1000 E$ = "CORONARY RISK"
1010  HOME : VTAB 2: PRINT  TAB( 13);E$: VTAB 5
1020  PRINT "CORONARY RISK WILL DETERMINE YOUR RISK": PRINT : PRINT "OF C
      ORONARY HEART DISEASE IN THE NEXT": PRINT : PRINT "FIVE YEARS, BASED
      ON YOUR AGE, HEIGHT,": PRINT : PRINT "WEIGHT, SEX, BLOOD PRESSURE A
      ND SERUM": PRINT : PRINT "CHOLESTEROL LEVEL."
1030  PRINT : PRINT : PRINT "YOU DON'T HAVE TO KNOW YOUR CHOLESTEROL": PRINT
      : PRINT "OR BLOOD PRESSURE TO USE THIS GAME."
```

(continued)

MENU Listing *(continued)*

```
1050  GOTO 120
2000 E$ = "WHY DO YOU SMOKE?"
2010  HOME : VTAB 3: PRINT  TAB( 10);E$: VTAB 6
2030  PRINT "THIS GAME WILL HELP CIGARETTE SMOKERS": PRINT : PRINT "DETER
     MINE THEIR REASONS FOR SMOKING.": PRINT
2040  PRINT "IF WANTED, IT WILL ALSO GIVE TIPS ": PRINT : PRINT "ON HOW T
     O QUIT OR CUT DOWN."
2050  GOTO 120
3000 E$ = "EXERCISE WEIGHT"
3010  HOME : VTAB 3: PRINT  TAB( 12);E$: VTAB 6
3020  PRINT  TAB( 2);"EXERCISE WEIGHT WILL DETERMINE YOUR": PRINT : PRINT
     TAB( 2);"DESIRABLE UNCLOTHED WEIGHT, BASED": PRINT : PRINT  TAB( 2)
     ;"ON LIFE INSURANCE TABLES."
3030  PRINT : PRINT : PRINT  TAB( 2);"IT WILL ALSO DETERMINE YOUR": PRINT
     : PRINT  TAB( 2);"DAILY CALORIE CONSUMPTION AND,": PRINT : PRINT  TAB(
     2);"IF YOU WISH TO GAIN OR LOSE WEIGHT,": PRINT : PRINT  TAB( 2);"IT
     WILL HELP YOU PLAN A DIET."
3040  GOTO 120
4000 E$ = "LIFE EXPECTANCY"
4010  HOME : VTAB 3: PRINT  TAB( 12);E$: VTAB 6
4020  PRINT  TAB( 3);"LIFE EXPECTANCY WILL ESTIMATE YOUR ": PRINT : PRINT
     TAB( 6);"APPROXIMATE LIFE EXPECTANCY": PRINT : PRINT  TAB( 8);"BASE
     D ONYOUR NSWERS TO": PRINT : PRINT  TAB( 11);"TWELVE QUESTIONS."
4030  GOO 120
5000 E$ = "IESTYLE"
5009  HOME : VTAB : PRINT  TAB( 15);E$: VTAB 6
5020  PRINT "LIFESTYLE WILL DETERMINE YOUR HEALTH": PRINT : PRINT "RISK S
     CORE BASED ON YOUR ANSWERS TO": PRINT : PRINT  TAB( 4);"QUESTIONS AB
     OUT YOUR HEALTH": PRINT : PRINT  TAB( 10);"AND LIFE HABITS."
5030  GOTO 120
6000 E$ = "HELP"
6010  HOME : VTAB 3: PRINT  TAB( 10);"EXPLAIN COMPUTER": VTAB 6
6020  PRINT "'EXPLAIN COMPUTER' WILL HELP YOU USE": PRINT : PRINT "THE AP
     PLE COMPUTER TO PLAY THE OTHER": PRINT : PRINT "HEALTH GAMES."
6030  VTAB 21: HTAB 4: PRINT "WOULD YOU LIKE TO TRY THIS GAME?": PRINT : PRINT
     " ( TYPE Y OR N ) ";
6040  POKE 216,0:C = 1200 GOSUB 7900: IF AN > 128 THEN N = AN - 128
6045 D$ =  CHR$ (AN): IF D$ = "N" THEN 200
6050  IF (D$ <  > "Y") AND (C <  > I) THEN  HTAB 1: PRINT " PLEASE TYPE Y
     OR N ";: GOTO 6040
6055  HOME : VTAB 3: PRINT "NOW GOING TO  EXPLAIN COMPUTER ...."
6060  PRINT " ": PRINT C$;"RUN ";E$
6070  STOP
7000  REM  MORE GAMES COMES HERE
7700  HOME : VTAB 3: PRINT "LOOKING FOR MORE GAMES....."
7710 FLAG = 0
7720  REM  NOW FIND OUT WHERE DOS' DEVICE TABLE IS:
7730 L = 47080:HIGH =  PEEK (116) * 256 +  PEEK (115): IF HIGH = 38400 THEN
     7760
7740  IF HIGH = 22016 THEN L = 30696: GOTO 7760
7750  HOME : GOTO 200: REM  CAN'T FIGURE OUT MACHINE SIZE
7760 CK =  PEEK (L + 1): IF (CK - 16 *  INT (CK / 16)) <' > 0 THEN X =  =
     : REM ERROR IF DEVICE BLOCK NOT THERE
7770 X = 3 -  PEEK (L + 2): IF (X > 2) OR X < 1 THEN X =  = :: REM   IF O
     LD DRIVE NUMBER WAS 1 NEW IS 2 AND VICE VERSA.  OTHERS ARE AN ERROR
7775 DR$ = "D2"
7780  PRINT C$"RUN MENU,D"X
7790  STOP
7900 ZZ =  PEEK ( - 16384):I = 1
7910 I = I + 1:AN =  PEEK ( - 16384): POKE  - 16368,0: IF (I = C) OR (AN <
     > ZZ) THEN  RETURN
7920  GOTO 7910
8000 E$ = "BIRTH CONTROL"
8010  HOME : VTAB 3: PRINT  TAB( 15);E$: VTAB 6
8020  PRINT  TAB( 2);"THIS GAME CHECKS YOUR KNOWLEDGE": PRINT : PRINT  TAB(
     2);"OF BIRTH CONTROL FACTS.  IT HELPS YOU"
8030  PRINT : PRINT  TAB( 2);"LEARN HOW TO AVOID GETTING PREGNANT."
8090  GOTO 120
63600  POKE 768,104: POKE 769,168: POKE 770,104: POKE 771,166: POKE 772,2
     23: POKE 773,154: POKE 774,72: POKE 775,152: POKE 776,72: POKE 777,9
     6
63610  ONERR  GOTO 63620
63615  RETURN
63620 CD$ =  CHR$ (4):JR =  PEEK (222): POKE 216,0: IF JR = 255 THEN  STOP
     : REM   CTRL/C
63635  IF (JR = 8) AND DR$ = "D2" THEN DR$ = "": PRINT CD$;"RUN MENU,D1"
63640 DA$ = "DAYFILE": PRINT CD$;"CLOSE": PRINT CD$"OPEN"DA$: PRINT CD$"A
     PPEND"DA$: PRINT CD$"WRITE"DA$
63650  PRINT "ERROR ";JR;" AT LINE "; PEEK (218) + PEEK 29) * 256;" I
     EN"
63660  PRINT CD$;"CLOSE": POKE 216,0: PRINT : PRINT : PRINT : PRINT "I'M
     SORRY I'VE GOTTEN VERY CONFUSED.": PRINT CD$"RUN MENU"
```

a few assembly-language routines.

Display format

We feel that display symmetry (or "composition" in the sense that artists use that term) is important. This is especially true for frequently seen displays, such as the initial display described below. Successful composition draws the eye to specific features of the picture; obviously there are limits to what can be done in text displays, but the display shouldn't lead the eye off the screen and should give information quickly without forcing the user to hunt for it. Consistency in format of displays, at least within a single lesson, is also important, and easier to program.

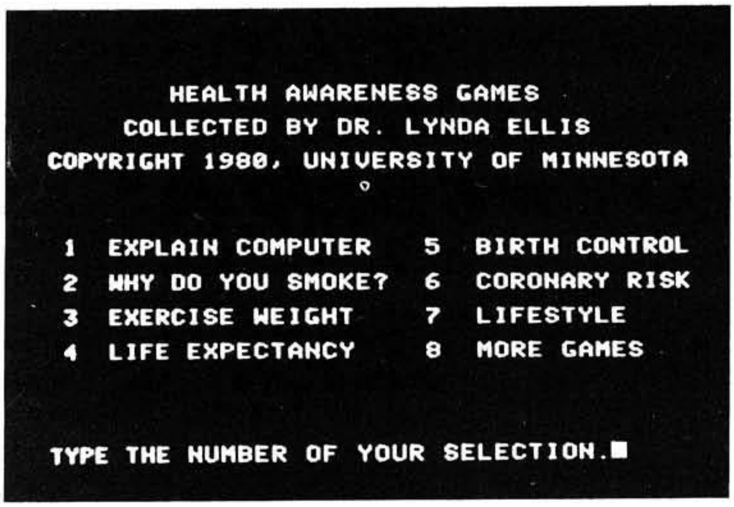
Displays should also be short. Huge blocks of text discourage a poor reader from continuing a lesson. In this respect, the limitations of the Apple's TV display have turned out to be something of a blessing in disguise. The display holds 24 lines with 40 characters on each line. For readability, however, we almost always double-spaced so that we had just 12 lines to work with. Initially this seemed to be a great drawback, but in practice it forces us to edit our text carefully. Careful rewording often reduces both average word length and sentence length. Sometimes a message must be split into two displays and this, too, can help the clarity of the lesson.

The initial display-part (figure 2) of the MENU program listed demonstrates our display philosophy. Since it is the first thing each new user should see, it will attract (or repel) users more than any other display in the package. One consicously attractive element in this display is the title "HEALTH AWARENESS GAMES." This title emphasizes the enjoyable aspects of the computer encounter. Health professionals are sometimes put off by our choice of the word "games," but we did not mean to imply that the lessons are foolish or trivial. "Game" simply had the best connotation of the alternatives we considered (which included "lesson," "program," "module," "quiz," and "test").

User Control

The system should be easy to use. For example, MENU easily permits users to select the other lessons. Users are likely to be more receptive to a freely-chosen lesson than one that is forced on them. The MENU program describes the other programs and, as each is completed, collects the user's evaluation of it.

Figure 2: **The Initial Display in MENU**



The initial commands to the computer are given by selecting from a numbered list. Watching first-time users interact with the computer has convinced us that numbered options are far better than lettered options. Locating a specific alphabetic key can be an ordeal for someone unfamiliar with a keyboard. Numbers may be located more easily.

As soon as the user presses a key in this display, something will happen. If the key does not correspond to one of the options, the sentence "PLEASE TYPE 1 2 3 4 5 6 7 or 8" will remind the user of the instructions, and the input prompt will be repeated.

If the key pressed is a number from 1 to 7, a short description of the chosen lesson is given. For example, a choice of "1" would produce figure 3. One of the advantages of a dedicated personal computer over a timesharing system for this application is that the change from one screen display to another can be much faster (using memory-mapped video output) than transfer at some fixed baud rate.

Though numbered multiple choice questions are preferred, they are not appropriate for all responses. However, when words are requested, we almost always allow the first letter to replace the whole word. People with minimal keyboard skill are extremely slow at entering multiple letter responses (numbers, as mentioned before, are easier). Also, multiple striking of the same key or a very hard single press can cause two of the same characters to appear where only one was intended. Selecting the left-most, non-blank character is the easiest satisfactory solution, and permits N, NNO, NO, NOPE, and N0 to all be correctly identified as negative responses. (It also allows for the wise guy who, when presented with a question with instructions to "TYPE YES OR NO," types "YES OR NO.")

System Expansion

The last option in the initial MENU display ("8 MORE GAMES") is more complicated from a programming standpoint. The MENU program contains all the information needed to use any of the programs on its own disk. However, this disk will only hold about 100K and, because lessons may contain a great deal of text, the space limitation is constricting. Our users cannot be expected to insert additional disks. Two or more disk drives could be used, with a program being run on one drive calling a program on another drive. The computer in our clinic waiting room has just one disk drive at present, and this part of the MENU program is not implemented there.

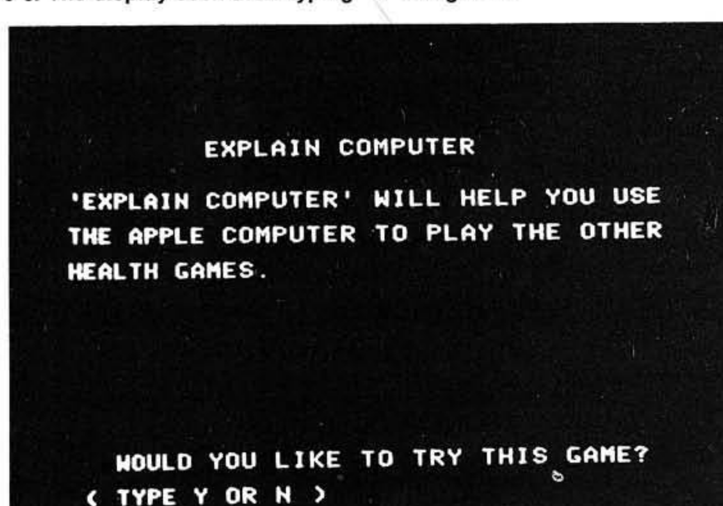
The two-disk-drive MENU program listed here is set up so that if two drives are available, each disk will have its own MENU program listing the programs available on that disk. Each disk will store the results of the user evaluation of the programs that are on it. The Apple Disk Operating System (DOS) uses the most recently selected disk drive as the default drive unless a drive is specifically requested. Thus the entire package can be written without regard to what drive it will be physically located on. When the last option is selected, the MENU program discovers which drive is currently in use and then RUNs the MENU program on the other drive. (This is accomplished by looking at the Input/Output Block of the DOS RWTS subroutine[5]). The addresses PEEKed in the version of MENU here are specific for a 48K Apple II.

User Motivation

Users need encouragement to continue with a computer encounter. Thus the benefit(s) from such an encounter should be given early and often. Also, all information requested from the user should be related to these benefits.

As an example, an early version of MENU had a "two-part" introduction. The initial display contained the title, credits, and a request to "PRESS THE RETURN KEY ON THE RIGHT TO PROCEED." The second display contained a simpler list of lessons and a request to choose an option or type the letter "S" to stop. Typing "S" would

Figure 3: **The display seen after typing "1" in figure 2.**



increment a counter of the number of people who had used the package and return to the initial display.

We hoped that this method would estimate the number of lessons run per encounter. However, unsupervised persons rarely would select the "S" option — they just walked away when they were done. We then developed the single display frame, which contains the most important information from both of the previous displays (figure 2). (A timer mechanism to distinguish the break between users is also unworkable. Some users will sit and talk or think after a lesson before selecting the next. On other occasions a line may form; then the time between one user leaving and another starting may be shorter than the time between lessons with a single user.)

One possible reason for this lack of user cooperation with a computer instruction is that the user can see little or no gain in typing "S". It appears to be an arbitrary requirement and is easily ignored. But requests for more personal information such as sex, age, height, and weight, almost always received a response, since they were readily understood as being necessary in a personal health lesson. Thus we attempt to relate requests for user action directly to user benefit.

This concern with personal relevance encourages a user to complete the lesson, and promotes cooperation that will carry over into the evaluation questions. These latter questions are of less immediate user benefit, but are of great benefit to us as lesson designers. Though we would like to ask many, they have been limited to two (figure 4) to avoid loss of user interest.

***Figure 4:* Evaluation Questions Used in the Health Awareness Package.**

a. How helpful was this game for you?
   1. Not helpful, a waste of time
   2. Slightly helpful
   3. Moderately helpful
   4. Very helpful, I'd recommend it

b. Will you change any living habits because of this game?
   1. Definitely not
   2. Probably not
   3. Perhaps
   4. Probably yes
   5. Definitely yes

HELP

Each lesson should provide assistance to users, but a separate HELP section is required as well to reassure the very timid and satisfy the curious. This material is relevant to all lessons and is too wordy to incorporate into each. An earlier version of MENU listed this differently from the other lessons: "TYPE THE NUMBER OF A GAME OR TYPE THE LETTER H FOR HELP," but placing it as the first choice on the regular numbered menu is easier for those who might need help most.

If a user does select this choice, figure 3 appears. In this display, if the key "N" is pressed, the user is returned to the main menu (figure 2). If a key other than "N" or "Y" is pressed, the message "NOW GOING TO EXPLAIN COMPUTER" is displayed while the HELP program is loaded and begun.

The response to a single keystroke is not usually as rewarding in other programs as it is in the first display of MENU. Since other programs sometimes require answers not as well-suited to a multiple-choice format (age to the nearest year, height, weight, etc.), the user must learn to type a response and then press the RETURN key. This is the first lesson of the HELP program. It also includes the location of the number keys, the difference between the number 0 and the letter O, how to correct mistakes using the back-arrow key, the evaluation process, the noise the disk drive makes, and the fact that, as described earlier, most whole-word answers can be shortened to the first letter.

HELP concludes with an optional summary (figure 5), and then is evaluated (in program MENU) just like the other lessons. Even though the second question (figure 4b) is not really applicable to HELP, we felt that the routine should be the same for all lessons.

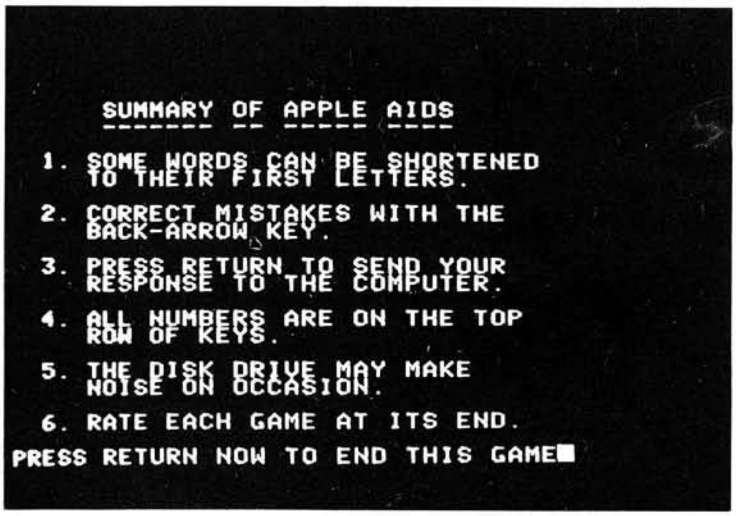
Meaningful Diagnostic Messages and User Prompts

HELP was not meant to be a prerequisite to the other lessons. Indeed, attempts to make it mandatory failed, since persons who, correctly or not, did not feel they required HELP were dissatisfied or bored. Also, as shown in figure 1, a person could use more than one lesson at a sitting. A repeat user would neither need nor want HELP.

Thus, all other lessons attempt to remind the user in at least the first few responses to press RETURN. This begins in the MENU program, where all program descriptions, except the one for HELP, end by asking "DO YOU WISH TO PLAY THIS GAME? (TYPE Y OR N AND PRESS RETURN)." Similarly, an alphabetic answer to a question that requests numbers (e.g., age, height, weight) will usually generate the response "ALL THE NUMBERS ARE IN THE TOP ROW." However, the information that a word response may be shortened to its first letter, and that the back-arrow key can be used to correct mistakes, is available only in HELP or through trial and error.

Diagnostic messages are written at specific places on the TV screen, instead of being scrolled underneath. When scrolling is used, a few persons delight in repeatedly answering incorrectly and watching the screen text scroll upwards into oblivion. When the message never changes position, most of the fun goes out of asking for more than a few repetitions.

***Figure 5:* The Summary Display in HELP**



Fail-soft

The term "fail-soft" describes redundant hardware that transfers processing in the event of a system failure. We can neither afford nor do we need such exhaustive failure protection.

We have described our provision for hardware failure earlier. However, software failures are much more frequent and potentially just as disruptive. For example, we described the expansion capability to use multiple disk drives. Use of that option on a single disk system will create an I/O error that could stop the system and require human intervention to get it restarted. Thus this error is checked for in the MENU program. The existing drive is then reselected and the MENU program is restarted to correct the situation. A similar error-trapping routine is used in the lessons. Errors that are not specifically corrected are coded and written into a disk file, along with the name of the program and line at which they occurred. The message "I'm sorry I've gotten very confused," is displayed. Then the MENU program is run.

Of course, programs are exhaustively tested before being installed in the unsupervised setting. But there is always "one last bug," and errors can also creep in either as a result of a faulty disk copy or through deterioration of the disk over time.

Evaluation and Response to Evaluation

The final role of the MENU program is lesson evaluation. The main goal of our project is to impart health information to users. The computer requests users to evaluate the lesson(s) they complete. The two evaluation questions we use (figure 4) appear to come at the end of each lesson. Actually they reside at the beginning of the MENU program.

Responses are tallied and the aggregate sum for each question is recorded on the disk. This is the only user information stored by the package. Questions of privacy and confidentiality of responses are especially important where health information is concerned.

We also evaluate the clinic staff's reactions to the system and have solicited more detailed comments from patients as part of a patient satisfaction survey. Results of these evaluations have been reported elsewhere[2,3,4]. Multiple cycles of evaluation, change, and reevaluation are necessary to incorporate a new technology in a setting such as ours.

Creation versus *Adaption*

While some of the health lessons in-

HELP Listing

```
70  GOSUB 63600
80 C$ =  CHR$ (4)
90  GOTO 200
100  HOME : VTAB 5: RETURN
110  VTAB 22: INPUT "PRESS RETURN TO CONTINUE";D$: GOSUB 100: RETURN
200  HOME : VTAB 5: PRINT  TAB( 5);"WELCOME TO THE HEALTH GAMES": VTAB 10
     : PRINT "PRESS THE RETURN KEY FOR HELP IN USING": PRINT : PRINT "THE
      APPLE COMPUTER TO PLAY THESE GAMES."
210  VTAB 20: PRINT "(THE RETURN KEY IS ON THE RIGHT SIDE": PRINT : INPUT
     "OF THE KEYBOARD.)";D$
600  GOSUB 100: PRINT "INTRODUCTION TO THE APPLE KEYBOARD": PRINT "------
     ------ -- --- ----- --------": PRINT : PRINT "YOU ALREADY KNOW THE M
     OST IMPORTANT": PRINT : PRINT "KEY ON THE KEYBOARD:"
610  VTAB 14: PRINT  TAB( 10);"THE RETURN KEY !!": GOSUB 110
620  PRINT "THE RETURN KEY IS USED TO 'SEND' YOUR ": PRINT : PRINT "MESSA
     GE TO THE COMPUTER.": VTAB 10: PRINT "THE COMPUTER DOESN'T LOOK AT Y
     OUR": PRINT : PRINT "ANSWERS TO QUESTIONS UNTIL YOU PRESS": PRINT : PRINT
      TAB( 10);"THE RETURN KEY."
622  GOSUB 110: PRINT "THIS MEANS THAT EVERY TIME YOU TYPE ": PRINT : PRINT
     "SOMETHING, LIKE THE WORD 'YES' OR THE": PRINT : PRINT "NUMBER '52',
      YOU MUST PRESS": VTAB 12: HTAB 10: PRINT "THE RETURN KEY": VTAB 15
624  PRINT "TO TELL THE COMPUTER THAT YOU ARE": PRINT : PRINT "THROUGH TY
     PING AND IT IS OK FOR IT TO": PRINT : PRINT "LOOK AT YOUR RESPONSE."

630  GOSUB 110: PRINT "THIS ALSO MEANS THAT IF YOU MAKE A": PRINT : PRINT
     "TYPING MISTEAK (OOPS!) YOU CAN CORRECT":: PRINT : PRINT "IT--": VTAB
     12: PRINT  TAB( 3);"IF YOU HAVEN'T YET PRESSED RETURN.": GOSUB 110
635  PRINT "MISTAKES CAN BE CORRECTED USING": PRINT : PRINT  TAB( 6);"THE
      BACK-ARROW KEY (<-)": PRINT : PRINT "LOCATED JUST BELOW THE RETURN
     KEY."
640  VTAB 15: PRINT "DO YOU WANT TO PRACTICE USING THIS": PRINT : PRINT "
     ARROW KEY NOW (TYPE YES OR NO AND THEN": PRINT : PRINT "PRESS THE RE
     TURN KEY)";
645  INPUT D$:D$ =  LEFT$ (D$,1)
650  IF D$ = "N" THEN 800
660  IF D$ <  > "Y" THEN  VTAB 21: PRINT "PLEASE TYPE YES OR NO AND THEN"
     : PRINT "PRESS THE RETURN KEY";: CALL  - 868: GOTO 645
700  GOSUB 100: PRINT "TYPE A WORD AND THEN USE THE BACK-ARROW": PRINT : PRINT
     "KEY (<-) TO BACKSPACE OVER ALL OR PART": PRINT : PRINT "OF THIS WOR
     D. THEN RETYPE TO CHANGE IT."
705  PRINT : PRINT : PRINT "FOR EXAMPLE, TYPE THE WORD 'YESH',": PRINT : PRINT
     "THEN PRESS THE <- KEY": PRINT
710  PRINT "ONCE TO CHANGE IT TO 'YES' .": PRINT : PRINT : PRINT "THEN PR
     ESS RETURN TO SHOW THE CHANGED": PRINT : PRINT "WORD TO THE COMPUTER
     ."
720  VTAB 23: INPUT D$: IF D$ = "C" THEN 800
730  GOSUB 100: PRINT "THE COMPUTER SAW THE WORD '";D$;"'": VTAB 9: PRINT
     "TYPE ANOTHER WORD AND CHANGE IT WITH": PRINT : PRINT "THE ARROW KEY
     ,  OR TYPE THE LETTER 'C'": PRINT : PRINT "TO GO ON."
740  PRINT : PRINT "(REMEMBER TO PRESS THE RETURN KEY!)": GOTO 720
800  HOME : PRINT "SOMETIMES YOU  NEED TO TYPE LONGER": PRINT : PRINT "ME
     SSAGES TO THE COMPUTER, SUCH AS": PRINT : PRINT "THE WORDS 'YES' OR
     'NO'."
810  VTAB 8: PRINT "THESE WORDS, AND SOME OTHERS, CAN BE": PRINT : PRINT
     "SHORTENED TO THEIR FIRST LETTERS.": PRINT : PRINT "THAT IS, YOU CAN
      TYPE THE LETTER 'Y'": PRINT : PRINT "INSTEAD OF 'YES' OR 'N' INSTEA
     D OF 'NO'"
820  VTAB 17: PRINT "DO YOU UNDERSTAND THIS (YES OR NO)?": PRINT : PRINT
     "(TYPE ONLY THE FIRST LETTER THIS TIME,"
830  PRINT : INPUT "AND THEN PRESS THE RETURN KEY.)";D$: IF D$ = "Y" THEN
     880
840  VTAB 19: PRINT "(TYPE THE LETTER 'Y' ALONE"; SPC( 13): PRINT : GOTO
     830
880  HOME : VTAB 2: PRINT "SOME MESSAGES TO THE COMPUTER ARE": PRINT
890  PRINT "NUMBERS.  ALL NUMBERS ON THE APPLE": PRINT : PRINT "KEYBOARD
     ARE ON THE TOP ROW OF KEYS.": VTAB 10
900  PRINT "THIS IS IMPORTANT TO REMEMBER": PRINT : PRINT "BECAUSE SOME N
     UMBER KEYS (0 OR 1)": PRINT : PRINT "ARE SOMETIMES CONFUSED WITH SOM
     E": PRINT : PRINT "LETTER KEYS (O, I OR L)."
905  VTAB 19: INPUT "TYPE THE NUMBER 0 TO CONTINUE. ";D$
907  IF D$ <  > "0" THEN  VTAB 21: PRINT "ALL NUMBER KEYS ARE ON THE TOP
     ROW.": VTAB 22: CALL  - 868: INPUT D$: GOTO 907
910  HOME : VTAB 2: PRINT "OTHER APPLE HINTS": PRINT "----- ----- -----":
      VTAB 5: PRINT "OCCASIONALLY THE COMPUTER HAS TO USE": PRINT : PRINT
     "THE DISK DRIVE."
920  VTAB 10: PRINT "WHEN THAT HAPPENS, YOU WILL HEAR A": PRINT : PRINT "
     SLIGHT NOISE, AND THE COMPUTER MIGHT"
925  PRINT : PRINT "NOT RESPOND TO YOU FOR A SHORT TIME."
927  PRINT : PRINT "THIS IS NORMAL.  IN JUST A FEW SECONDS,": PRINT : PRINT
     "THE COMPUTER WILL BE READY TO HELP YOU": PRINT : PRINT "AGAIN.": GOSUB
     110
930  VTAB 2: PRINT "FINALLY,  AT THE END OF EVERY HEALTH": PRINT : PRINT
     "GAME, INCLUDING THIS ONE, WE WILL ASK": PRINT : PRINT "YOU TO RATE
     THE GAME TO TELL US HOW": PRINT : PRINT "YOU LIKED IT."
```

(continued)

HELP Listing *(continued)*

```
940  VTAB 11: PRINT "THIS IS DONE IN TWO QUESTIONS, WHICH": PRINT : PRINT
     "YOU WILL SEE AT THE END OF THIS GAME.": PRINT : PRINT "PLEASE ANSWE
     R THESE RATING QUESTIONS": PRINT : PRINT "HONESTLY, TO HELP US HELP
     YOU BETTER.": GOSUB 110
950  PRINT "WOULD YOU LIKE TO SEE A SUMMARY": PRINT : PRINT "OF WHAT WAS
     MENTIONED IN THIS GAME": PRINT : PRINT "(YES OR NO)";
960  INPUT D$:D$ =  LEFT$ (D$,1): IF D$ = "N" THEN 3000
970  IF D$ <  > "Y" THEN  VTAB 10: PRINT "PLEASE TYPE YES OR NO";: GOTO 9
     60
980  HOME : PRINT : PRINT  TAB( 7);"SUMMARY OF APPLE AIDS": PRINT  TAB( 7
     );"------- -- ----- -----": VTAB 5
982  PRINT  TAB( 3);"1. SOME WORDS CAN BE SHORTENED": PRINT  TAB( 6);"TO
     THEIR FIRST LETTERS.": PRINT
985  PRINT  TAB( 3);"2. CORRECT MISTAKES WITH THE": PRINT  TAB( 6);"BACK-
     ARROW KEY."
990  PRINT : PRINT  TAB( 3);"3. PRESS RETURN TO SEND YOUR": PRINT  TAB( 6
     );"RESPONSE TO THE COMPUTER."
1000  PRINT : PRINT  TAB( 3);"4. ALL NUMBERS ARE ON THE TOP": PRINT  TAB(
     6);"ROW OF KEYS."
1010  PRINT : PRINT  TAB( 3);"5. THE DISK DRIVE MAY MAKE": RINT  TAB( 6)
     ;"NOISE ON OCCASION."
1015  PRINT : PRINT  TAB( 3;"6. RATE EACH GAME AT ITS END"
1020  VTAB 22: INPUT "PRESS RETURN NOW TO END THIS GAME";D$

3000 C$ =  CHR$ (4): PRINT C$;"NOMON I,C,O": VTAB  PEEK (37): CALL  - 868
     :FLAG = 1: PRINT C$;"BLOAD APPLESOFT CHAIN,A520"
3010  CALL 520"MENU"
63600  POKE 768,104: POKE 769,168: POKE 770,104: POKE 771,166: POKE 772,2
     23: POKE 773,154: POKE 774,72: POKE 775,152: POKE 776,72: POKE 777,9
     6
63610  ONERR  GOTO 63620
63615  RETURN
63620 CD$ =  CHR$ (4):JR =  PEEK (222): POKE 216,0
63630  IF JR = 255 THEN  STOP : REM      CTRL/C
63640  PRINT CD$;"CLOSE": PRINT CD$;"OPEN DAYFILE": PRINT CD$;"APPEND DAY
     FILE": PRINT CD$;"WRITE DAYFILE"
63650  PRINT "ERROR ";JR;" AT LINE "; PEEK (218) +  PEEK (219) * 256;" IN
     HELP"
63660  PRINT CD$;"CLOSE": TEXT : PRINT : PRINT : PRINT : PRINT : PRINT "I
     'M SORRY I'VE GOTTEN VERY CONFUSED.": PRINT CD$"RUN MENU"
```

cluded in the package were written specifically for this project, others were adapted from material originally developed for a teletypwriter-based timesharing system. This latter route initially appeared to be the method of choice, since these were properly-working programs. However, timesharing systems with their sign-on procedures require more sophisticated users than those found in our environment. In addition, instructions in the timesharing programs were often too cryptic, required keyboard literacy, or were in other ways unsuitable. Also, programs developed on a slow, printer-based system very rarely repeated instructions (hard copy was assumed). Thus few display statements were directly transferable to a rapid screen-oriented system, and what control structure did survive would have probably been improved by rewriting. These programs were partially rewritten to incorporate simple graphics and other features not appropriate for a printer-based system. We actually saved little if any time by translating existing software rather than completely rewriting the programs.

Acknowledgments

We thank the staff and patients at the Community University Health Care Center for help with the development of this package. This work has been supported in part by the National Library of Medicine and the Foundation for Computer-Based Education. The version of this package described in reference 4 is available through Biomedical Graphics, University of Minnesota, B-192 Phillips Wangensteen Building, Minneapolis, MN 55455.

References

1. Ellis, L.B.M., Raines, J.R., and Brown, J.W., "Teaching Health Risk Concepts Using Microcomputers," in *Proc. Third Annual Symposium on Computer Applications in Medical Care* (ed. R.A. Dunn), pp 159-163, Institute of Electrical and Electronics Engineers, Inc., New Jersey, 1979.

2. Ellis, L.B.M. and Raines, J.R., "Health Education Using Microcomputers: Initial Acceptability," *Preventative Medicine* (1981) 10:77-84.

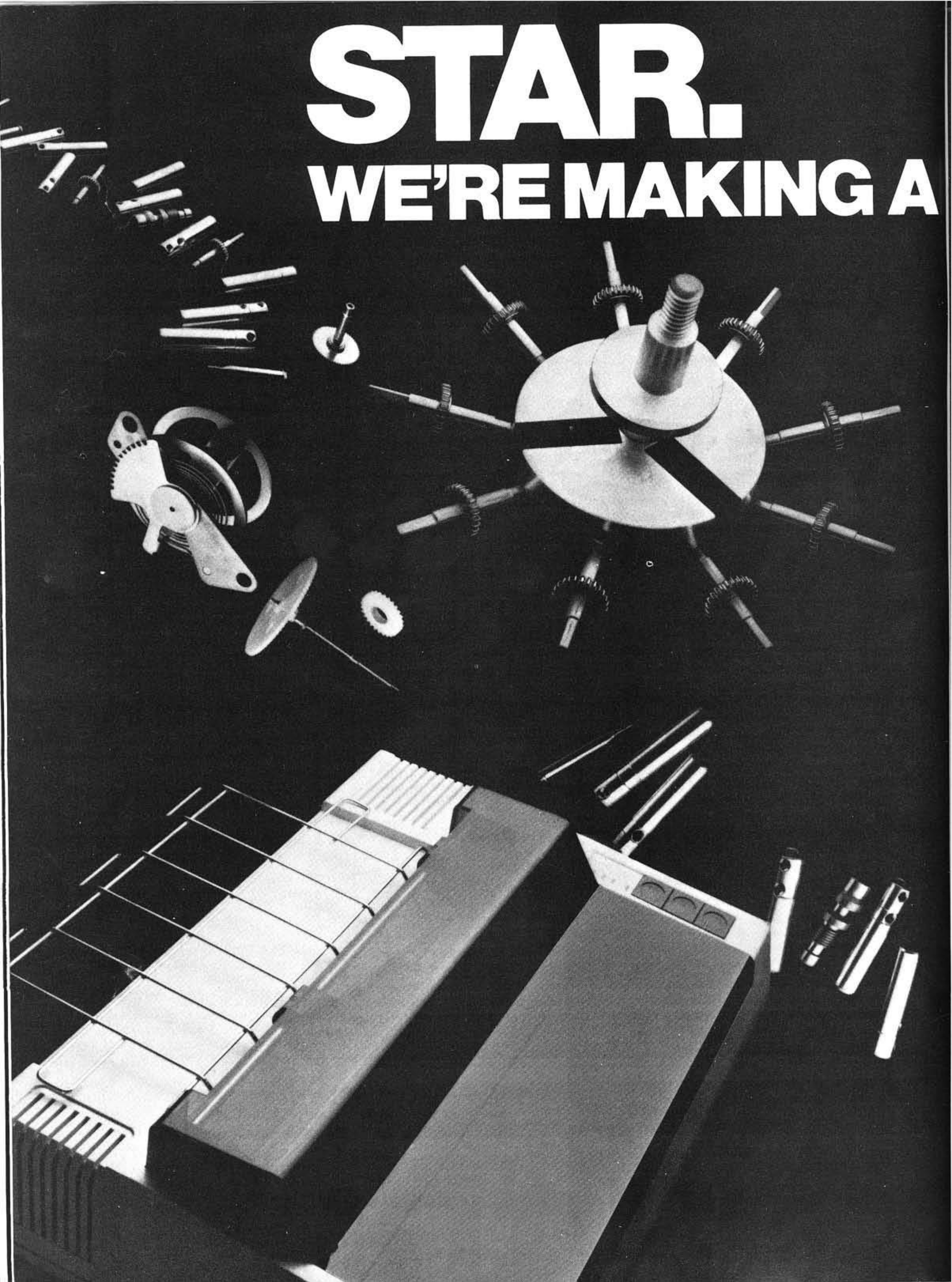
3. Ellis, L.B.M., Raines, J.R., and Hakanson, N., "The Microcomputer in the Waiting Room,'" in *Proc. Fourth Annual Symposium on Computer Applications in Medical Care* (ed. J.T. O'Neill), pp 1412-1415, Institute of Electrical and Electronics Engineers, Inc., New Jersey, 1980.

4. Ellis, L.B.M., Raines, J.R., and Hakanson, N., "Health Education Using Microcomputers: One Year in the Clinic," *Preventative Medicine* (1982) 11:212-224.

5. Cole, P. and Howard, B., *The Do's and Don'ts of DOS: DOS 3.2 Instructional and Reference Manual,* Apple Computer, Cupertino, California, 1979, pp 94-98.

The authors may be contacted at the Division of Health Computer Sciences, Box 511 Mayo Memorial Building, University of Minnesota, Minneapolis, Minnesota 55455.

MICRO


STAR.
WE'RE MAKING A

PET Vet

By Loren Wright

A Better BASIC

The University of Waterloo has made a major advance in the teaching of computer programming with the introduction of the SuperPET and its interpreted high-level languages.

I have touched on various aspects of the SuperPET in previous columns (July, 1981: announcement; April, 1982: architecture and 6809 assembler/linker). Here I'd like to concentrate on Waterloo microBASIC.

PET BASIC is easy to learn, but it is a poor language for learning good programming techniques. Apple Pascal is a good language for teaching, but the Apple's limited memory forces the student to spend too much time loading, saving, and debugging, and not enough actually learning. Until recently, microcomputers just didn't have enough memory to run conveniently a good high-level language. The SuperPET, of course, has the memory to pull this feat off — 64K for each language. The result is an excellent, low-cost teaching system.

microBASIC

Like the other Waterloo languages, microBASIC must be loaded from the system disk. This takes several seconds, but once it is stashed away in the SuperPET's bank-switched memory, you don't have to load anything from the disk. Like the other languages currently available for the SuperPET, microBASIC is an interpreter. This means the user finds out right away about his mistakes (great, in an educational environment), but it also means program execution won't set any speed records.

MicroBASIC works very much like PET BASIC. The editor is the same, except the cursor moves slower and the delete key works backwards. Like the PET editor (see my May, 1982 column — 48:25), program changes can be made simply by altering the screen copy of a line and hitting return. Additional features in the editor are auto-numbering, renumbering, and delete functions. Although it is much less convenient, the microEditor may also be used to work on BASIC programs. It has a number of powerful commands, including search-and-replace.

Most of PET BASIC will work with microBASIC in exactly the same way. Character string and Boolean functions perform quite differently, though. Substrings are specified simply with the beginning and ending character positions. For instance, if A$ is "abcdef", then A$(2:4) is "bcd". Assignment of substrings is made in the same way. The statement, A$(2:4) = "xyz", changes A$ to "axyzef", and A$(5:5) = "ghi" changes this new A$ to "axyzghif". Expressions may be used for the indices, and there are a few more subtleties. Boolean functions work on a strictly logical, rather than a bit-by-bit basis. You can't say 'IF A AND 128' and expect to get a 'TRUE' result when A has bit $80 on.

MicroBASIC has matrix functions, like the bigger BASICs on mainframe computers. A matrix is a multidimensional array, just like you're used to in PET BASIC, but matrix functions allow you to read in the elements conveniently and to output all or part of the matrix. Other matrix manipulation functions include addition, subtraction, multiplication, and transposition.

The big area of improvement is structuring. Structuring means different things to different people, but basically a structured program is one whose flow is easy to understand. This means that variables and parts of the program should have meaningful names, rather than two-letter abbreviations and cryptic line numbers. It usually means that there are some more sophisticated structures available to control program flow. Highly structured languages (Pascal, for instance) require very rigid structure, such as variable declarations. MicroBASIC will support almost all of the unstructured features of PET BASIC. (The manual groups them in a chapter titled Primitive Control.) There are GOTO, GOSUB, RETURN, ON...GOTO, and ON...GOSUB statements.

microBASIC's Control Structures

The simplest structure is LOOP...ENDLOOP. Everything between these two statements is repeated until the STOP key is pressed or until a QUIT is encountered. QUIT passes control out of the structure (not out of the program), or to a special part of the structure. The following program will print the numbers 1 to 20, followed by the word DONE.

```
10 DIGIT = 1
20 LOOP
30   PRINT DIGIT
40   DIGIT = DIGIT + 1
50   IF DIGIT > 20 THEN QUIT
60 ENDLOOP
70 PRINT "DONE"
```

If line 50 had been omitted, then the program would continue cranking out numbers until the STOP key was pressed, until the numbers got too high, or until the power was turned off. Note that we can call our variable a meaningful DIGIT, rather than an obscure X. The indented format makes the stucture easier to understand, and the spaces will not be compressed out, as in PET BASIC.

The next program works the same as the previous example, except that each number has "EVEN" or "ODD" printed after it on the same line.

```
 10 DIGIT = 1
 20 LOOP
 30   PRINT DIGIT;
 40   IF DIGIT - 2*INT(DIGIT/2) = 0
 50     PRINT "EVEN"
 60   ELSE
 70     PRINT "ODD"
 80   ENDIF
 90   DIGIT = DIGIT + 1
100   IF DIGIT > 20 THEN QUIT
110 ENDLOOP
120 PRINT "DONE"
```

The IF...THEN...ELSE structure makes an "either or" choice easy to program, and easy to understand in the listing. To accomplish the same thing in PET BASIC, you would have to use one or two GOTO statements. It would get even more complicated if you wanted to test yet another condition if the first was not satisfied. MicroBASIC has an ELSEIF statement for these multiple-choice situations.

The following example is the same as the previous example, except that I have removed the odd/even determination to a procedure.

```
 10 DIGIT=1
 20 LOOP
 30   PRINT DIGIT;
 40   CALL ODDEVEN (DIGIT)
 50   IF ODDEVEN=0
 60     PRINT "EVEN"
 70   ELSE
 80     PRINT "ODD"
 90   ENDIF
100   DIGIT=DIGIT+1
110   IF DIGIT>20 THEN QUIT
120 ENDLOOP
130 PRINT "DONE"
140 !
150 PROC ODDEVEN (X)
160   ODDEVEN=X-2*INT(X/2)
170 ENDPROC
```

DIGIT is passed to the procedure and replaced there by X. ODDEVEN is set within the procedure to 1 or 0 and this is tested in line 50, as before. Instead of DIGIT, I could have used any valid expression. Also, I could have used a whole list of expressions in the CALL statement, as long as there were corresponding variables in the PROC heading. Procedures may be recursive (call themselves), and the variables within the procedure are local. That is, there could be another variable X elsewhere in the program above, and its value would not be disturbed by a CALL of ODDEVEN.

Each of the three examples below illustrates another structure available in microBASIC.

WHILE...ENDLOOP

```
10 DIGIT=1
20 WHILE DIGIT<21
30   PRINT DIGIT
40   DIGIT=DIGIT+1
50 ENDLOOP
60 PRINT "DONE"
```

LOOP...UNTIL

```
10 DIGIT=1
20 LOOP
30   PRINT DIGIT
40   DIGIT=DIGIT+1
50 UNTIL DIGIT>20
60 PRINT "DONE"
```

FOR...NEXT

```
10 FOR DIGIT=1 TO 20
20   PRINT DIGIT
30 NEXT DIGIT
40 PRINT "DONE"
```

Each program produces the same result as my very first example. Probably the most familiar is the FOR...NEXT, which works exactly the same as in PET BASIC, except that the index variable is destroyed when the loop is exited. WHILE...LOOP allows a condition to be tested before executing the included group of statements, and LOOP...UNTIL performs the test after executing the group of statements. WHILE and UNTIL may be combined to allow tests at both the beginning and end of the same structure. Also, there is a GUESS...ADMIT...ENDGUESS structure that is less restrictive than IF...ELSE...ELSEIF...ENDIF in multiple-choice situations.

MicroBASIC has some useful debugging features, such as a STEP mode and informative error messages. Because of its interpreted and structured natures, it is an excellent language for learning good techniques. The same interpreter is available on the IBM Personal Computer and several mainframe and minicomputers. Also, a compiler will soon be made available. These will make microBASIC suitable for serious application programming, as well.

MICRO

VIC Fast Locator

Michael B. Casella, 277 E. 10th Avenue,
Chico, California 95926

As an owner of Commodore's new VIC-20, you probably know that, to date, there is no disk drive available for your computer. This limits you to cassette storage if you want to save your programs. There is nothing wrong with cassette storage, unless you have a tape full of programs. If you need to access a program near, or at, the end of the tape, you must LOAD "<file name>," press PLAY, and wait. Depending on how long your tape is, you could be waiting for as lng as twenty minutes.

Recently I found an article and program called "The Case of the Missing Tape Counter," by William F. Pytlik, which utilized the PET's realtime clock and allowed you to use fast forward to accesss your files. What it does is allot ten seconds of fast forward for each program. Even though you will waste some tape, you will find that space to be enough room for just about any program.

When I typed the program into my VIC-20 from Mr. Pytlik's article, I found that it didn't work. A few locations need to be changed to make the conversion. These locations have been POKED with values in lines 180 and 240, which initialize the cassette's motor off, and line 210, which initializes the cassette's motor on.

The program is self explanatory. The fast forward time can be changed by changing the number 10 in line 230; otherwise it will fast forward for ten seconds.

The locator program is saved at the beginning of each tape. When you save new programs, you include the data statements corresponding to the new program and resave the locator program at the beginning of the tape.

You will find that this program will waste some tape, but it will certainly save a lot of loading time. You can also use it as a subroutine in other programs to save or access files.

The following is a list of variables used in the "VIC FAST LOCATOR" program:

C$(X)	DIMensions the total number of program title strings,
C$(I)	The array of program title strings (loaded from DATA statements),
F$	Carriage Return following fast forward press to start cassette and time sequence,
I	READ DATA and increment loop,
R$	"Load" or "Save" variable,
T	time increment (10 seconds... user adjustable).
TI	sets real time clock,
WP	program choice input variable,
X	number of programs on LOCATOR list (user adjustable).

```
10 RESTORE
20 PRINT"■"
30 PRINT"***VIC FAST LOCATOR***"
40 PRINT"■◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆■"
50 READ X
60 DIM C$(X)
70 FOR I=1 TO X
80 READ C$(I)
90 PRINT C$(I)
100 NEXT I
110 FOR X=1 TO 100: NEXT X
120 PRINT"■LOAD OR SAVE(L OR S)"
130 GETR$:IFR$=""THEN130
140 PRINT
150 INPUT"WHICH PROGRAM 1,2...";WP
160 IF WP<1 OR WP>10 THEN 150
170 IF WP=1 THEN GOTO 260
180 POKE 37148,1
190 PRINT"■PRESS F.FWD AND HIT   RETURN WHEN READY"
200 GETF$:IFF$=""THEN200
210 POKE 37148,254
220 T=TI
230 IF TI<T+(10*60*(WP-1)) THEN 230
240 POKE 37148,1
250 PRINT
260 IF R$="L" THEN PRINT"■■■HIT STOP AND LOAD FILE■"
   : GOTO 290
270 IF R$="S" THEN PRINT"■■■HIT STOP/EJECT": PRINT
280 PRINT"TAPE IS NOW READY TO SAVE NEW PROGRAM/FILE"
290 DATA 10
300 DATA"1 VIC FAST LOCATOR"
310 DATA"2 PROGRAM 2"
320 DATA"3 VIC FAST LOCATOR"
330 DATA"4 PROGRAM 4"
340 DATA"5 PROGRAM 5"
350 DATA"6 PROGRAM 6"
360 DATA"7 PROGRAM 7"
370 DATA"8 PROGRAM 8"
380 DATA"9 PROGRAM 9"
390 DATA"10 ETC."
400 END
READY.
```

Processor Status Flags

Peter A. Cook, 1443 N. 24th Street, Mesa, Arizona 85203

MICRO readers may find this table useful in interpreting machine language monitor displays. It shows all the possible combinations of the 6502's flags in the status register. Just look up the decimal or hex number and the table will show you all the flags that are set.

6502 Processor Status Flags

N	NEGATIVE	D	DECIMAL
V	OVERFLOW	I	INTERRUPT
U	UNUSED	Z	ZERO
B	BREAK	C	CARRY

Status Register Dec.	Hex	Flags Set	Status Register Dec.	Hex	Flags Set	Status Register Dec.	Hex	Flags Set	Status Register Dec.	Hex	Flags Set
0	00	--------	64	40	-V------	128	80	N-------	192	C0	NV------
1	01	-------C	65	41	-V-----C	129	81	N------C	193	C1	NV-----C
2	02	------Z-	66	42	-V----Z-	130	82	N-----Z-	194	C2	NV----Z-
3	03	------ZC	67	43	-V----ZC	131	83	N-----ZC	195	C3	NV----ZC
4	04	-----I--	68	44	-V---I--	132	84	N----I--	196	C4	NV---I--
5	05	-----I-C	69	45	-V---I-C	133	85	N----I-C	197	C5	NV---I-C
6	06	-----IZ-	70	46	-V---IZ-	134	86	N----IZ-	198	C6	NV---IZ-
7	07	-----IZC	71	47	-V---IZC	135	87	N----IZC	199	C7	NV---IZC
8	08	----D---	72	48	-V--D---	136	88	N---D---	200	C8	NV--D---
9	09	----D--C	73	49	-V--D--C	137	89	N---D--C	201	C9	NV--D--C
10	0A	----D-Z-	74	4A	-V--D-Z-	138	8A	N---D-Z-	202	CA	NV--D-Z-
11	0B	----D-ZC	75	4B	-V--D-ZC	139	8B	N---D-ZC	203	CB	NV--D-ZC
12	0C	----DI--	76	4C	-V--DI--	140	8C	N---DI--	204	CC	NV--DI--
13	0D	----DI-C	77	4D	-V--DI-C	141	8D	N---DI-C	205	CD	NV--DI-C
14	0E	----DIZ-	78	4E	-V--DIZ-	142	8E	N---DIZ-	206	CE	NV--DIZ-
15	0F	----DIZC	79	4F	-V--DIZC	143	8F	N---DIZC	207	CF	NV--DIZC
16	10	---B----	80	50	-V-B----	144	90	N--B----	208	D0	NV-B----
17	11	---B---C	81	51	-V-B---C	145	91	N--B---C	209	D1	NV-B---C
18	12	---B--Z-	82	52	-V-B--Z-	146	92	N--B--Z-	210	D2	NV-B--Z-
19	13	---B--ZC	83	53	-V-B--ZC	147	93	N--B--ZC	211	D3	NV-B--ZC
20	14	---B-I--	84	54	-V-B-I--	148	94	N--B-I--	212	D4	NV-B-I--
21	15	---B-I-C	85	55	-V-B-I-C	149	95	N--B-I-C	213	D5	NV-B-I-C
22	16	---B-IZ-	86	56	-V-B-IZ-	150	96	N--B-IZ-	214	D6	NV-B-IZ-
23	17	---B-IZC	87	57	-V-B-IZC	151	97	N--B-IZC	215	D7	NV-B-IZC
24	18	---BD---	88	58	-V-BD---	152	98	N--BD---	216	D8	NV-BD---
25	19	---BD--C	89	59	-V-BD--C	153	99	N--BD--C	217	D9	NV-BD--C
26	1A	---BD-Z-	90	5A	-V-BD-Z-	154	9A	N--BD-Z-	218	DA	NV-BD-Z-
27	1B	---BD-ZC	91	5B	-V-BD-ZC	155	9B	N--BD-ZC	219	DB	NV-BD-ZC
28	1C	---BDI--	92	5C	-V-BDI--	156	9C	N--BDI--	220	DC	NV-BDI--
29	1D	---BDI-C	93	5D	-V-BDI-C	157	9D	N--BDI-C	221	DD	NV-BDI-C
30	1E	---BDIZ-	94	5E	-V-BDIZ-	158	9E	N--BDIZ-	222	DE	NV-BDIZ-
31	1F	---BDIZC	95	5F	-V-BDIZC	159	9F	N--BDIZC	223	DF	NV-BDIZC
32	20	--U-----	96	60	-VU-----	160	A0	N-U-----	224	E0	NVU-----
33	21	--U----C	97	61	-VU----C	161	A1	N-U----C	225	E1	NVU----C
34	22	--U---Z-	98	62	-VU---Z-	162	A2	N-U---Z-	226	E2	NVU---Z-
35	23	--U---ZC	99	63	-VU---ZC	163	A3	N-U---ZC	227	E3	NVU---ZC
36	24	--U--I--	100	64	-VU--I--	164	A4	N-U--I--	228	E4	NVU--I--
37	25	--U--I-C	101	65	-VU--I-C	165	A5	N-U--I-C	229	E5	NVU--I-C
38	26	--U--IZ-	102	66	-VU--IZ-	166	A6	N-U--IZ-	230	E6	NVU--IZ-
39	27	--U--IZC	103	67	-VU--IZC	167	A7	N-U--IZC	231	E7	NVU--IZC
40	28	--U-D---	104	68	-VU-D---	168	A8	N-U-D---	232	E8	NVU-D---
41	29	--U-D--C	105	69	-VU-D--C	169	A9	N-U-D--C	233	E9	NVU-D--C
42	2A	--U-D-Z-	106	6A	-VU-D-Z-	170	AA	N-U-D-Z-	234	EA	NVU-D-Z-
43	2B	--U-D-ZC	107	6B	-VU-D-ZC	171	AB	N-U-D-ZC	235	EB	NVU-D-ZC
44	2C	--U-DI--	108	6C	-VU-DI--	172	AC	N-U-DI--	236	EC	NVU-DI--
45	2D	--U-DI-C	109	6D	-VU-DI-C	173	AD	N-U-DI-C	237	ED	NVU-DI-C
46	2E	--U-DIZ-	110	6E	-VU-DIZ-	174	AE	N-U-DIZ-	238	EE	NVU-DIZ-
47	2F	--U-DIZC	111	6F	-VU-DIZC	175	AF	N-U-DIZC	239	EF	NVU-DIZC
48	30	--UB----	112	70	-VUB----	176	B0	N-UB----	240	F0	NVUB----
49	31	--UB---C	113	71	-VUB---C	177	B1	N-UB---C	241	F1	NVUB---C
50	32	--UB--Z-	114	72	-VUB--Z-	178	B2	N-UB--Z-	242	F2	NVUB--Z-
51	33	--UB--ZC	115	73	-VUB--ZC	179	B3	N-UB--ZC	243	F3	NVUB--ZC
52	34	--UB-I--	116	74	-VUB-I--	180	B4	N-UB-I--	244	F4	NVUB-I--
53	35	--UB-I-C	117	75	-VUB-I-C	181	B5	N-UB-I-C	245	F5	NVUB-I-C
54	36	--UB-IZ-	118	76	-VUB-IZ-	182	B6	N-UB-IZ-	246	F6	NVUB-IZ-
55	37	--UB-IZC	119	77	-VUB-IZC	183	B7	N-UB-IZC	247	F7	NVUB-IZC
56	38	--UBD---	120	78	-VUBD---	184	B8	N-UBD---	248	F8	NVUBD---
57	39	--UBD--C	121	79	-VUBD--C	185	B9	N-UBD--C	249	F9	NVUBD--C
58	3A	--UBD-Z-	122	7A	-VUBD-Z-	186	BA	N-UBD-Z-	250	FA	NVUBD-Z-
59	3B	--UBD-ZC	123	7B	-VUBD-ZC	187	BB	N-UBD-ZC	251	FB	NVUBD-ZC
60	3C	--UBDI--	124	7C	-VUBDI--	188	BC	N-UBDI--	252	FC	NVUBDI--
61	3D	--UBDI-C	125	7D	-VUBDI-C	189	BD	N-UBDI-C	253	FD	NVUBDI-C
62	3E	--UBDIZ-	126	7E	-VUBDIZ-	190	BE	N-UBDIZ-	254	FE	NVUBDIZ-
63	3F	--UBDIZC	127	7F	-VUBDIZC	191	BF	N-UBDIZC	255	FF	NVUBDIZC

MICRO

Formatted Output with Atari BASIC

by Frank Roberts

This routine gives ATARI users the benefits of formatted output from ATARI BASIC. Extremely versatile, this routine is just as effective as "print using" commands found in more complex BASICs.

Formatted Output
requires:
Atari 400/800

Whether you are outputting stock and bond prices, profit/loss statements, or baseball batting averages, it is always a plus to format the report in neat, justified columns — especially if the columns have totals. True, the computer doesn't care whether decimal places line up, but it does make a difference to the human who reads the output. "Pretty-print" is not just another aesthetic, it is computer courtesy. A well-formatted report is much easier to read and comprehend.

One particular frustration for me when I first started programming my Atari for business reports, was the absence of a print using command in Atari BASIC. All those decimal points scattered over the paper just didn't look professional. Particularly annoying are figures such as $0.5 instead of $ .50. The following program contains a relocatable subroutine for formatting numbers to a user's or programmer's specifications.

Options presented are for length and type of character format. For example, the program (listing 1) offers the user a choice of four formats, and prompts user selection. Inputting "*****" will choose a field length of five integer positions with asterisks in place of blanks or leading zeros, and two decimal places, rounded (i.e., 1.555 will format

Listing 1

```
100 REM ***** FORMATTED OUTPUT BY SUBROUTINE AND $ MANIPULATION *****

110 REM ***** FRANK ROBERTS
120 REM ***** (C) APRIL 1982
130 REM
140 OPEN #1,4,0,'K:'
150 OPEN #2,8,0,'S:'
160 POKE 752,1
170 DIM FRM$(20),NUM$(20),PIC$(20),BLANK$(20)
180 BLANK$='                    '
190 PRINT CHR$(125):PRINT :PRINT
200 PRINT 'ENTER FORMAT PICTURE:':PRINT
210 PRINT 'TYPE ONE CHARACTER FOR EACH':? 'INTEGER POSITION
(DEFAULT=4)':PRINT · PRINT
220 PRINT '       0000...        BBBB...':PRINT
230 PRINT '       $$$$...        ****...':PRINT
240 POSITION 15,16:PRINT '.......'
250 POSITION 15,16
260 GET #1,A:IF A=155 THEN 290
270 PIC$(LEN(PIC$)+1)=CHR$(A):PRINT #2;CHR$(A+128);:GOTO 260
280 FOR J=1 TO LEN(PIC$):IF PIC$(J,J)='.' THEN PIC$(J,J)=' ':NEXT J
290 IF LEN(PIC$)=0 THEN PIC$=BLANK$(1,4)
300 IF LEN(PIC$)<4 THEN PIC$(2)=PIC$(1):PIC$(3)=PIC$(1,2)
310 IF PIC$(1,1)='B' THEN PIC$(1,LEN(PIC$))=BLANK$
320 IF LEN(PIC$)>1 THEN IF PIC$(1,1)='$' THEN PIC$(2,LEN(PIC$))=BLANK$
330 IF PIC$(LEN(PIC$))<>'.' THEN PIC$(LEN(PIC$)+1)='.'
340 LPRINT :LPRINT 'YOUR CHOICE.......',CHR$(34);PIC$;CHR$(34):LPRINT
350 LPRINT ,'UNFORMATTED','FORMATTED':LPRINT
360 FOR J=1 TO 20
370 R1=INT(RND(0)*10)
380 R2=RND(0)*100
390 IF R1<4 THEN R2=R2*(-1)
400 R=R1*R2
410 R=INT(R*1000)/1000
420 NUM=R
430 GOSUB 510
440 LPRINT ,R,,FRM$
450 NEXT J
460 PRINT CHR$(125);:? :? :? 'DO YOU WANT ANOTHER TEST? ';:INPUT
PIC$:IF PIC$(1,1)='Y' THEN RUN
470 POKE 752,0
480 END
490 REM
500 REM ***** SUBROUTINE TO FORMAT THE NUMBER R *****
510 FRM$=PIC$:PT=LEN(FRM$):SN=0
520 IF NUM=0 THEN FRM$(PT+1)='00 ':RETURN
530 IF NUM<0 THEN NUM=NUM*(-1):SN=1
540 NUM=INT(NUM*100+0.5):N=INT(NUM/100)
550 NUM$=STR$(N):LN=LEN(NUM$)
560 IF N=0 THEN NUM$=FRM$(PT-1,PT-1)
570 FRM$(PT-LN,PT-1)=NUM$
580 NUM$=STR$(NUM):LN=LEN(NUM$)
590 IF LN=1 THEN NUM$(2)=NUM$:NUM$(1,1)='0':LN=2
600 NUM$=NUM$(LN-1)
610 FRM$(PT+1)=NUM$:FRM$(PT+3)=CHR$(32)
620 IF SN THEN FRM$(PT+3)=CHR$(45)
630 RETURN
```

to ****1.56). The default value is four integer positions, so that inputting a single character, such as "*" or "0", will produce the following formats:

Input	Format
*	****.00
0	0000.00

Inputting a "B" or just a carriage return will also default to four positions, but with all leading zeros suppressed by-empty space (blanks) as in the following:

Unformatted	Formatted
0	.00
0.1	.10
12.356	12.36
1234	1234.00

Inputting six "$'s" ($$$$$$) will provide only a five-integer-positions format, as the "$" will only appear in the first position and suppress all zeros before the first whole integer. For example:

Unformatted	Formatted
0	$ .00
0.1	$ .10
12.356	$ 12.36
12345	$12345.00

The upper limits of the formatted number is 20-integer-positions. This limit, however, is controlled only by the DIMensions of PIC$ and FRM$ (line 170).

This particular program was written mainly for demonstrating user-oriented versatility for the formatting subroutine that begins in line 510. In a business program, the programmer would usually take the option of presetting the length and character format by initializing PIC$ to the desired format before the program begins processing data. A simple line such as PIC$ = "00000." will do the job for an entire report. However, I have in my file two programs that allow the user to choose the formatting, and they work quite well. Of course, care must be taken to safely estimate the maximum number of integer positions required by the report before choosing the length of PIC$; otherwise the program would terminate on an error if the number generated was larger than the format. For this reason it is safer and easier to preset PIC$ — otherwise an elaborate user-error trapping routine would be necessary (and beyond the purpose of this article).

Listing 1 contains a random number generator and demonstration print routine. Normally this section would be replaced by the main processing of data required by the particular program. The subroutine (lines 510-630) do the actual formatting work. These

Listing 2

```
Add the following to Listing 1 for user input of numbers:

175 DIM LINE$(80),TAB$(4)
185 TAB$=BLANK$(1,4)
340 PRINT :PRINT
350 PRINT "ENTER 4 NUMBERS NOT LARGER THAN PIC$":PRINT
360 LINE$=""
370 FOR J=1 TO 4
380 INPUT R
390 NUM=R
400 GOSUB 510
410 NEXT J
420 LPRINT LINE$
430 GOTO 340
440 REM
450 REM
625 LINE$(LEN(LINE$)+1)=FRM$:LINE$(LEN(LINE$)+1)=TAB$
```

Sample Run

YOUR CHOICE....... "00000."

UNFORMATTED	FORMATTED
470.283	00470.28
273.661	00273.66
-43.424	00043.42-
540.513	00540.51
299.816	00299.82
212.469	00212.47
-174.376	00174.38-
-14.49	00014.49-
-152.463	00152.46-
-148.249	00148.25-
-4.535	00004.54-
-133.429	00133.43-
644.65	00644.65
651.336	00651.34
210.9	00210.90
23.628	00023.63
221.429	00221.43
0	00000.00
-210.718	00210.72-
342.251	00342.25

YOUR CHOICE....... "   ."

UNFORMATTED	FORMATTED
656.314	656.31
0	.00
46.032	46.03
108.695	108.70
605.236	605.24
87.304	87.30
-256.22	256.22-
492.395	492.40
345.602	345.60
388.687	388.69
221.035	221.04
308.248	308.25
210.227	210.23
0	.00
-20.092	20.09-
164.111	164.11
378.997	379.00
21.063	21.06
108.221	108.22
346.295	346.30

YOUR CHOICE....... "$."

UNFORMATTED	FORMATTED
0.43	$ .43
367.208	$367.21
-81.876	$ 81.88-
855.683	$855.68
0	$ .00
-194.242	$194.24-
595.303	$595.30
-94.736	$ 94.74-
-280.477	$280.48-
595.143	$595.14
641.189	$641.19
440.605	$440.61
92.178	$ 92.18
144.006	$144.01
740.551	$740.55
379.896	$379.90
582.618	$582.62
-72.245	$ 72.25-
1.538	$ 1.54
-159.16	$159.16-

YOUR CHOICE....... "*******."

UNFORMATTED	FORMATTED
-46.761	*****46.76-
188.594	****188.59
349.267	****349.27
542.314	****542.31
303.863	****303.86
0	*******.00
0	*******.00
287.113	****287.11
287.185	****287.19
246.569	****246.57
480.201	****480.20
324.044	****324.04
269.66	****269.66
292.004	****292.00
132.221	****132.22
572.525	****572.53
355.319	****355.32
-13.355	*****13.36-
-21.958	*****21.96-
455.191	****455.19

line numbers can be changed to suit the programmer's needs. I generally place such routines near the top (beginning) of a program because they work much faster. With interpretive BASIC, the computer must start at the beginning of a program and search each line in sequence until it finds the subroutine. It does this each time a GOSUB statement is encountered, so that, on long programs, subroutines placed near the end take up unnecessary time to process.

After returning to the main program from the subroutine, the formatted number is ready for printing. In this demonstration only one column of formatted numbers is printed. Business reports and forms, however, often require several columns of formatted numbers. This can be easily accomplished in Atari BASIC by concatenating the formatted numbers into a long string of, say, 80- or 132-character length and dumping the entire batch at one time. Listing 2, an illustration of such a method, is an input routine that allows users to enter numbers until they get tired (at which time pressing any letter will terminate the program). It also formats the numbers into four columns by hooking FRM$ to LINE$ (line 625).

The formatting subroutine presented here has proven itself a real work horse in every report I've run. With a few modifications, it can be tailored to any kind of report or form. Credit (CR) and Debit (DB) can be added in place of the negative sign by changing line 620 as follows:

```
620 IF SN THEN FRM$(PT+3)=
    "CR" (or "DB", depending on
    what kind of account is being
    formatted)
```

Changing line 610 to:

```
610 FRM$(PT+1)=NUM$:FRM$
    (PT+3)=CHR$(43)
```

will tag a "+" onto every positive number, should the programmer want all signed numbers. With a little tricky string manipulation (which I will leave to the reader to solve), the negative or positive sign can be placed at the beginning of the format instead of the end.

The subroutine is extremely versatile and just as effective as a print using command found on more complex and expensive BASICS. Try it on your next report.

Contact the author at 3736 Ferndale Dr., Ft. Wayne, IN 46815.

MICRO

MICRO™

New Publications

BASIC Exercises for the Apple, by J.P. Lamoitier. Sybex, Inc. (2344 Sixth St., Berkeley, CA 94710), 1982, 251 pages, 7 × 9 inches, paperback.
ISBN: 0-89588-084-9 $12.95

Designed to rapidly teach Applesoft BASIC to Apple users, this book covers exercises related to data processing and finance, statistics, operations research, games, and more. Each exercise includes a statement and analysis of the problem, flow chart, and a corresponding program, as well as an actual run.

CONTENTS: Your First Program in BASIC; Flowcharts; Exercises Using Integers; Elementary Exercises in Geometry; Exercises Involving Data Processing; Mathematical Computations; Financial Computations; Games; Operations Research; Statistics; Miscellaneous; Apendices; Index.

Logo for the Apple II, by Harold Abelson. BYTE/McGraw-Hill Publications, Inc. (70 Main Street, Peterborough, NH 03458), 1982, 228 pages, 7 × 9¼ inches, paperback.
ISBN: 0-07-000426-9 $14.95

Logo for the Apple II teaches programming techniques through Turtle Geometry — a series of exercises involving both Logo programming and geometric concepts. It includes more advanced projects, such as the famous DOCTOR program with its simulated psychotherapist, and an INSTANT program that enables parents and teachers to create a programming environment for preschool children.

CONTENTS: A First Look at Logo; Programming with Procedures; Projects in Turtle Geometry; Workspace, Filing, and Debugging; Numbers, Words, and Lists; Projects Using Numbers, Words, and Lists; Writing Interactive Programs; Inputs, Outputs, and Recursion; Advanced Use of Lists; Glossary of Logo Primitive Commands; Appendix 1; Appendix 2; References; Index.

Kids Can Touch, by Patricia Shillingsburg. Patricia Shillingsburg (6 Magnolia Place, Summit, NJ 07901), 1981, 43 pages, 8½ × 11 inches, paperback.
$4.95 plus $1 shipping.

Kids Can Touch is a lively self-teaching guide to the Apple computer. It starts at the beginning, identifying the parts of the computer, turning it on and off, booting a disk. The book introduces computing with simple programs. The final chapters describe how computers work, their history, and what their presence may mean to our future.

CONTENTS: It's For Kids; Your Name's the Game; More Programming; Starting a Conversation; The Big Switch; Why Computers?; Computers Have Come and Are Here to Stay.

Swift's 1982 Educational Software Directory, Apple II Edition. Sterling Swift Publishing Co. (1600 Fortview Rd., Austin, TX 78704), 1982, 358 pages. 5½ × 8½ inches, paperback.
ISBN: 0-88408-150-8 $14.95

Two comprehensive indexes (alphabetical and by grade level) list thousands of educational software titles from 128 publishers. Each program is described, giving its price, publisher, grade level, memory and language requirements, backup policies, and cites reviews.

CONTENTS: Introduction; Software from Apple; Traditional Education Publishers; Other Education Publishers; Education Software Houses; Non-Commercial Software; Other Education Software; Education Software Distributors; Review Sources.

Instant BASIC, 2nd Edition, by Jerald R. Brown, dilithium Press (P.O. Box 606, Beaverton, OR 97075), 1982, 196 pages, 8¼ × 11 inches, paperback.
ISBN: 0-918398-57-6 $12.95

This is an active-participation and well-tested instructional workbook for the student, absolute beginner, and complete novice. It can be used with any brand of computer using Microsoft® BASIC.

CONTENTS: Ready, Set, RUN; Little Boxes: LET, the INPUT Cousins, and the READ-DATA Team; Loop de Loop (in other words, GOTO); Variables, Floating Point, and Work Savers; Compare and Decide: The IF ... THEN Family; Function Junction #1: SQR, INT, RND; Automatic Loops: FOR-NEXT; Function Junction #2: LEN, RIGHT$, LEFT$, MID$, STR$, VAL, ASC, CHR$, TAB, DEF FN, SIN, SGN, ABS; The Mysterious Realm of Subscripted Variables: One Dimensional Arrays; Two Subscripts & Subroutines Too: Two Dimensional Arrays, GOSUB-RETURN, ON ... GOTO...; ASCII Code Chart; Function Reference List; Index; Answers to End of Chapter Problems.

MICRO

APPLE Pascal P-Code Interpreter and the 6809

by Tom Whiteside

The author rewrites pieces of the APPLE UCSD Pascal P-code interpreter for the MC6809 and shows code size and speed improvements. P-code to native MC6809 code translation is also discussed.

Many 6502 users have switched over to UCSD Pascal from BASIC to take advantage of Pascal's increased reliability, easier modification, and greater portability between machines. Faster program execution is another plus for Pascal over BASIC, but the advantage in a stock Apple is not dramatic since the 6502 was really not designed to emulate the multiple 16-bit register, stack oriented, UCSD Pascal interpreter. However, with MC6809 cards becoming available for the Apple, we have a chance to switch over to a processor that will run Pascal much faster. The byte efficiency of the MC6809 also opens the door to translating the P-code directly into machine code for ultra fast operation!

This article will contrast pieces of the 6502 P-code interpreter's operation with a MC6809 version to show just how much faster, more byte efficient, and easier to program the MC6809 really is. We will then explore the possibility of using the MC6809's greater byte efficiency to translate the P-code to MC6809 native code. The P-code examples chosen for this article can be considered typical. Since the entire Pascal P-code interpreter is about 8K bytes long, it is only possible to cover a small part of it. The primary goal of this article is to use the P-machine to illustrate real-life MC6809 programming techniques and to demonstrate its superiority over the 6502.

The Pascal P-Machine

The University of California at San Diego (UCSD) Pascal "Pseudo-machine" or "P-machine" is designed to execute code created by their Pascal compiler. This P-machine concept has made it very easy to adapt UCSD Pascal to a large number of host processors. To use UCSD Pascal on a new machine, it is only necessary to rewrite the P-machine in the native code of the processor, which is far easier than rewriting the compiler.

The P-machine consists of eight, 16-bit registers and an instruction set of P-codes. These registers and instructions are documented in the Apple Pascal Operating System documentation. Briefly, the P-machine registers are:

SP:	The evaluation stack pointer. It is used to pass parameters and return function values.
IPC:	The interpreter program counter. A pointer to the next P-code instruction.
SEG:	The pointer to the procedure dictionary of the segment of the currently executing procedure.
JTAB:	A pointer referencing the code portion of the currently executing procedure.
KP:	A pointer to the current top of the program stack.
MP:	A pointer to the topmost markstack in the currently executing procedure. Used to reference local variables.
NP:	A pointer to the current top of the dynamic heap. All uses of NEW in Pascal reserve space using the NP pointer.
BASE:	A pointer to the main program code. All global variables are referenced relative to BASE.

The documentation on the P-machine in the Apple Pascal manual describes the operation of the P-machine in detail with examples. The real beauty of modifying the P-machine is that the individual tasks done by each P-instruction are quite simple. The compiler has the complicated job of selecting the sequence of P-codes — we do not have to worry about that!

The Apple P-Code Interpreter

Figure 1 illustrates the register variable usage for the Apple P-machine. The only 6502 register, which directly replaces a P-register, is the stack pointer. Since the 6502 stack register has only 8 bits, the stack is limited to 256 bytes.

The seven other P-machine registers are 16-bit and mapped into zero page, since the 6502 has no 16-bit registers other than the program counter.

```
SP    <-----> Stack pointer
IPC   <-----> <-------
SEG   <----->         |
JTAB  <----->         |
KP    <----->         |- Zero page
MP    <----->         |
NP    <----->         |
BASE  <-----> <-------

A, X, Y, and several zero page variables are
temporaries
```

***Figure 1:* Apple P-Machine Emulation**

```
SP    <-----> S register
IPC   <-----> X register
SEG   <-----> <--
JTAB  <----->    |
KP    <----->    |- Indexed off U register
MP    <----->    |
NP    <----->    |
BASE  <-----> <--

D (A, B) and Y registers are temporaries
```

***Figure 2:* MC6809 P-Machine Emulation**

The 6502's A, X, and Y registers, and several zero page locations, are used as pure temporaries in the Apple implementation of the P-machine. The P-machine makes extensive use of 16-bit pointers and almost all quantities are words forcing extensive use of indirect addressing.

The MC6809 Pascal P-Machine

Figure 2 illustrates the register variable usage for the MC6809 P-machine. Two registers in the MC6809 will be used to directly replace P-machine registers. The S register will act as the evaluation stack register just as the 6502 stack pointer did. However, the MC6809 stack will not be limited to a fixed page and is position independent. The 16-bit X register will act as the Interpreter Program Counter (IPC).

The six other 16-bit P-machine registers will be referenced by the MC6809 U register. This register storage is reserved on the system stack as shown in figure 3.

The remaining MC6809 registers, the D (the combination of the A and B registers) and Y registers, will act as temporaries for manipulating data. We might have chosen to replace another P-machine register with the U register and used direct page storage for the other registers instead of the approach taken here. The interesting thing about this method is that there is no need to worry about which locations are tied up in fixed locations. If more system variables need to be added later, only the stack initialization code must be changed. One of my big complaints about Apple Pascal is that system variables are scattered all over the memory map. Using the stack storage philosophy eliminates this problem. If all buffers and system variables are stack references, it becomes much simpler to extend features without worrying about stepping on another piece of code. All MC6809 code shown here will utilize this concept. We will not use temporary variables and all code will be position-independent.

Example 1. The NOP Instruction

The easiest place to start our comparison of 6502 and MC6809 code is the NOP P-instruction. NOP is a universal term for No OPeration and simply increments the program counter to the next instruction without doing anything else. Actually, almost every P-instruction uses NOP to increment the IPC, fetch the next instruction, and jump to it.

Figure 4 shows the 6502 code used to perform the NOP instruction. The best place to begin reading the code is the NOP entry point. The PUSHIM code is called only if a fetched P-code is positive. (All positive P-codes are data — NOT instructions and are pushed on the stack as the LSB of a word). Assuming a P-code byte is negative (and hence an instruction), it is multiplied by two and stored into the destination address of an indirect jump (Self-modifying code!). This jump table contains pointers to the actual 6502 code to execute the instruction. For example, if the P-code instruction was a $83, the jump offset would be $83 * 2 = $06. The jump table begins at $D000 in bank 2 of the language card. At location $D006 there is a pointer to $EAC2, which is the location of the code for this instruction. If we look in the documentation for instruction 131 ($83 decimal), we find that this is the

```
*
*  Set up labels for the P-machine registers
*
SEG      EQU $00         Label for SEG register
JTAB     EQU SEG+2         "    "  JTAB    "
KP       EQU JTAB+2        "    "  KP      "
MP       EQU KP+2          "    "  MP      "
NP       EQU MP+2          "    "  NP      "
BASE     EQU NP+2          "    "  BASE    "
*
         LEAS -(BASE+2),S  Carve off stack space on
*                            the System stack for
*                            the P-registers
*
         TFR S,U         Init User stack to the
*                          beginning of the P-register
*                          storage
              .
              .
              .
         STD JTAB,U      EXAMPLE USE OF "JTAB"
```

Figure 3: **Example showing use of the U register for system variable storage.**

```
*
* PUSHIM --- PUSH IMMEDIATE DATA ON THE STACK
*
*    FUNCTION:
*        Push "0" and A on the stack
*        Fall back into NOP
*
*    USAGE:
*        NOP branches here when P-code is >= 0
*
*    SIDE EFFECTS:
*        X is trashed
*
PUSHIM  EQU *          Lng Tim
        TAX             1   2    SAVE THE IMMEDIATE DATA IN X
        TYA             1   2    CLEAR THE ACCUMULATOR
        PHA             1   3    PUSH A ZERO ON THE STACK
        TXA             1   2    PUSH IMMEDIATE DATA ON STACK
        PHA             1   3
*
*  NOP -- FETCH THE NEXT INSTRUCTION AND JUMP THERE
*
*    FUNCTION:
*        repeat { loop on data bytes }
*          bump the IPC ( program counter )
*          get the next P-code byte
*          if byte >= 0 then push word on stack
*        until byte < 0
*        multiply byte * 2
*        stuff result into the operand of a jmp ()
*        jump to the code for that instruction
*
*    USAGE:
*        almost all P-instructions call NOP
*
*    ENTRY POINTS:
*        NOP  - normal entry
*        NOP1 - no IPC increment
*
*    SIDE EFFECTS:
*        Instruction * 2 is returned in A.
*        X is trashed. Y points at last P-code
*          byte loaded.
*
NOP     EQU *          Lng Tim
        INC IPC         2   5    BUMP 16-BIT PROGRAM COUNTER
        BNE NOP1        2  2/3
        INC IPC+1       2   5
NOP1    LDY #$00        2   2
        LDA (IPC),Y     2   5    FETCH THE NEXT INSTRUCTION
        BPL PUSHIM      2  2/3   IF  DATA > 0 THEN PUSH IT
        ASL A           1   2    ELSE MULTIPLY PCODE BY TWO
        STA NEXTOP+1    2   3      INIT JUMP DESTINATION
        JMP NEXTOP      3   3      GO THERE     JMP($DOXX)
              .
NEXTOP  JMP ($DOXX)     3   5
                           --
                           26
```

Figure 4: **6502 NOP P-Instruction**

"ADI" instruction. Thus, by reading this table, we can find the location of any P-instruction!

Many of the P-code instructions call other code to increment the IPC by more than one byte and then jump to NOP at entry point NOP1. For brevity, we will not show the code used to increment the IPC outside of NOP. All MC6809 examples will need only one entry point to NOP.

Figure 5 shows the MC6809 NOP code. For those unfamiliar with MC6809 assembler notation, the "+" beside an index register is a post-increment directive. This means after finishing an instruction, the index register is to be incremented. "++" means increment it twice. Similarly, "–" and "– –" are pre-decrement directives that decrement the index register before executing the instruction. In all the MC6809 interpreter code, we will always update the IPC (X reg) as each P-code byte is used, since this is straightforward and efficient. This is different than the 6502 code, which tries to minimize the 16-bit IPC updates.

Notice the technique used to reference the instruction lookup table (INSTAB). The LEAY INSTBL,PCR instruction loads the effective address of the table relative to the program counter and is thus position-independent.

The JMP operand in brackets means the operand is indirect. Unlike the 6502, the MC6809 can do indexed and indexed-indirect jumps. This is certainly cleaner than the 6502 NOP's self-modifying code.

Table 1 summarizes the bytes and machine cycles required for both versins of NOP. As the table indicates, the MC6809 code took ten bytes less and was slightly faster on data stores with GETIMM and two cycles slower for a simple NOP. While the MC6809 was slightly slower than the 6502 on the NOP instruction, the MC6809 code is position-independent and not self-modifying. The MC6809's real speed advantage will appear as we begin manipulating 16-bit pointers. We will use these timing results later to show the overhead of the interpreter for both processors.

Example 2. The LDCI P-Instruction

The next, slightly more complex P-code example is a 16-bit load immediate instruction. This instruction fetches the next two words following the instruction byte and pushes them on the stack. The 6502 code for LDCI is shown in figure 6. The 6502 must load the immediate data backwards, since it handles all pointers in LSB-MSB order. The 6502 "JMP SKPLWD" (SKiP Long WorD) adds three to the IPC and jumps to NOP1 (At a cost of 5 extra machine cycles over a NOP jump). Figure 7 is the LDCI code for the MC6809. Since the MC6809 loads words in MSB-LSB order like the P-machine, the "immediate" load is a simple LDD instruction. The "++" bumps the IPC past the data word to point at the next P-byte consistent with our convention. Table 2 shows the byte and machine cycle statistics for LDCI. The increase in complexity for LDCI over LDCN has little effect on the MC6809 code, but the difference really shows on the 6502.

	MC6809	6502
Bytes:	16	26
Cycles:		
PUSHIMM	19	30*
Simple NOP	32	30*

* Assuming the MSB of the IPC is not incremented

Table 1: **Statistics for the NOP P-Instruction**

	MC6809	6502
Bytes:	7	12
Cycles:	52	58

Table 2: **Statistics for the LDCI P-Instruction**

	MC6809	6502
Bytes:	14	30
Machine Cycles: (including GETIMM)		
Byte	92	107
Word	104	122*

* Assuming no update of IPC msb is required

Table 3: **Statistics for the LDL P-code Instruction**

Instruction	Byte Ratio *		Cycle Ratio *	
PUSHIMM	16/26	(.62)	19/30	(.63)
LDCN	8/7	(1.1)	47/41	(1.1)
LDCI	7/12	(.58)	52/58	(.90)
FJB				
FJP not taken	27/55	(.49)	52/52	(1.0)
FJB positive	27/55	(.49)	59/71	(.83)
FJB negative	27/55	(.49)	83/95	(.87)
LDL				
LDL byte	14/30	(.47)	92/107	(.86)
LDL word	14/30	(.47)	104/122	(.85)
average	17.5/33.4	(.52)	63.5/72.5	(.88)

* ratios show: MC6809 / 6502

Table 4: **Statistics showing byte and machine-cycle ratios of the MC6809 over the 6502 (MC6809 code is position-independent).**

```
*   NOP --- No OPeration
*
*    FUNCTION:
*        repeat
*          Fetch the next P-code byte
*          bump IPC (program counter)
*          If byte >= 0 then push byte on stack
*        until byte < 0
*        multiply the byte by two
*        use byte as an offset in P-code jump table
*        jump to the instruction
*
*    USAGE:
*        Most P-instructions call NOP
*
*    SIDE-EFFECTS:
*        The P-code byte * 2 is returned in B
*        The A, Y registers are temporary
*
NOP      EQU *              Lng Tim
         CLRA                1   2
NOP1     EQU *                        REPEAT
         LDB ,X+             2   6      GET THE NEXT PCODE BYTE
*                                          AND BUMP THE IPC
         BLT NOPEXT          2   3      IF PCODE < 0 THEN LEAVE
         PSHS D              2   7        ELSE SAVE WORD ON STACK
         BRA NOP1            2   3    UNTIL PCODE < 0
*
NOPEXT   EQU *                        GO DO THE INSTRUCTION
         ASLB                1   2      P-CODE:=PCODE * 2
         LEAY INSTBL,PCR 4       9      GET EFFECTIVE ADDRESS OF
*                                           INSTRUCTION TABLE
         JMP [D,Y]           2   10     GO THERE
                            ---
                            16
```

***Figure 5:* 6809 NOP P-Instruction**

```
*
*   LDCI --- LOaD one word Constant
*
*     FUNCTION:
*        Transfer the word immediately after
*        the P-instruction to the stack
*        Hop to NOP via SKPLWD (5 cycles extra)
*
*     USAGE:
*        P-code instruction
*
*     SIDE-EFFECTS
*        A and Y are temporary
*
*
LDCI     EQU *            Lng Tim
         LDY #$02          2   2
         LDA (IPC),Y       2   5   GET LSB FIRST
         PHA               1   3
         DEY               1   2
         LDA (IPC),Y       2   5   NEXT GET MSB
         PHA               1   3
         JMP SKPLWD        3   3   BUMP IPC BY 3 BYTES
*                                    AND JUMP TO NOP1.
                              35 (SKPLWD)
                          ---
                          12
```

***Figure 6:* 6502 LDCI P-Instruction**

```
*
*   LDCI --- LOaD one word Constant
*
*     FUNCTION:
*        Transfer the word immediately after
*        the P-instruction to the stack
*
*     USAGE:
*        P-code instruction
*
*     SIDE-EFFECTS
*        D is temporary
*
LDCI     EQU *      Lng Tim
         LDD ,X++    2   8      GRAB IMMEDIATE WORD AND
*                                 BUMP IPC PAST WORD
         PSHS D      2   7      SAVE IT ON THE STACK
         LBRA NOP    3   5      GO DO NEXT INSTRUCTION
*                       32      (NOP)
                    ---
```

***Figure 7:* MC6809 LDCI P-Instruction**

Example 3. The LDL P-Instruction

As a final interpreter example, let's examine a typical P-machine instruction that uses the GETIMM subroutine. The LDL instruction (LoaD Local word) is identical to several other instructions (LLA, LDO, and LAO) except for the P-machine register used to form the load address. The store equivalents of LDL (STL and SRO) are also very similar. LDL forms an address pointer based on immediate data and the MP register. This points to a local variable in the currently executing procedure. The word at this address is loaded and pushed on the stack in MSB LSB format. Figures 8 and 9 are the LDL code for the 6502 and MC6809 respectively. Table 3 is the byte and cycle statistics for LDL.

Table 4 summarizes the P-code interpreter for the MC6809 and the 6502. The table shows that the MC6809 required half the bytes of the 6502 and runs about 12% faster based on averaging our examples. From table 4, it is apparent that the 6502 did better on simple tasks, such as the LDCN instruction, in which the 6502 was faster and more byte efficient. For more complex tasks, such as the LDL instruction, the MC6809 was 10% faster than the 6502. While it is impossible to truly compare the efficiency of both processors in this application without analyzing all the P-instructions and their frequency of occurrence in the compiled P-code, the instructions covered in this article should be representative.

P-Code To Native Code Translation

We have seen so far that the MC6809 is faster than the 6502 at interpreting P-code, but a large percentage of the instruction execution time has been interpreter overhead. For example, the LDL instruction took 72 cycles to load a word and then 32 cycles to execute a NOP just to get the next instruction. That means 31% of the time to do an LDL is directly burned by the interpreter. The interpreter overhead is actually even worse since most P-code instructions are fairly general while the native code need not be. For example, the native GETIMM code can be different depending on whether a byte or word is to be loaded. The overhead of deciding which form of load is needed will be handled by the native code translator, NOT by the actual native code.

We will now redo the MC6809 interpreter P-instructions in native code and see just how much faster it will be. We will not do corresponding 6502 examples as the code growth would be too

```
*
*  LDL --- Load Local word
*
*   FUNCTION:
*       Form an address pointer using P-register MP and
*       the immediate data word.  Push the word pointed
*       at by the sum of this pointer and "MPOFF".
*
*   USAGE:
*       P-code instruction.  Very similar to others
*
*   SIDE-EFFECTS:
*       A,Y,TEMP2,TEMP2+1,TEMP3,TEMP3+1 are temporaries
*
LDL     EQU *          Lng Tim
        LDY #$01        2   2
        JSR GETIMM      3   3  GET IMMEDIATE WORD --> TEMP2
        LDA MP          2   3  ADDRESS OF MP ACTIVATION RECORD
        CLC             1   2
        ADC TEMP2       2   3  ADD OFFSET INTO MP ACTIVATION RECORD
        STA TEMP3       2   3
        LDA MP+1        2   3
        ADC TEMP2+1     2   3
        STA TEMP3+1     2   3
        LDY #MPOFF      2   2
        LDA (TEMP3),Y   2   5  GET WORD POINTED AT AND PUSH IT
        PHA             1   3
        DEY             1   2
        LDA (TEMP3),Y   2   5
        PHA             1   3
        JMP SKIPWD      3   3  HOP TO NOP (ADD 5 CYCLES)
*                          35    (SKIPWD)
                       ---
                        30
```

Figure 8: **The 6502 LDL P-Instruction**

```
*
*  LDL --- Load Local word
*
*   FUNCTION:
*       Form an address pointer using P-register MP and
*       the immediate data word.  Push the word pointed
*       at by the sum of this pointer and "MPOFF".
*
*   USAGE:
*       P-code instruction.  Very similar to others
*
*   SIDE-EFFECTS:
*       D and Y are temporaries
*
LDL     EQU *          Lng Tim
        LBSR GETIMM     3   9  D:=OFFSET TO LOCAL VARIABLE
        ADDD MP,U       2   7  FORM ADDRESS OF LOCAL VARIABLE
        TFR D,Y         2   6  Y:=ADDRESS
        LDD MPOFF,Y     2   6  D:=(LOCAL VARIABLE)
        PSHS D          2   7  SAVE IT
        LBRA NOP        3   5  GO DO NEXT INSTRUCTION
*                          32  (NOP)
                       ---
                        14
```

Figure 9: **The MC6809 LDL P-Instruction**

```
               Lng Tim
LDX MP,U        2   6    GET "MP" REGISTER
LDD OFFSET,X    4   9    LOAD LOCAL WORD
PSHS D          2   7    AND SAVE IT
               --- ---
                8  22
```

Figure 10: **The Native Code LDCI Instruction**

```
               Lng Tim
LDX MP,U        2   6    GET "MP" REGISTER
LDB OFFSET,X    4   8    LOAD LOCAL WORD
PSHS D          2   7    AND SAVE IT
               --- ---
                8  21
```

Figure 11: **The Native Code LDL Instruction for Word Load**

expensive. (Remember that the MC6809 interpreter took only half the bytes of the 6502!) Figures 10 and 11 are the native code versions of our P-instructions. In all examples, I rely on a "smart" translator to compute branch offsets and operand formats (word *versus* bytes).

Table 5 is a comparison of byte and machine cycle comparisons for the native code MC6809 versus the MC6809 interpreter code. As you can see, the native code grew by a factor of three over the very compact P-code. The native code will run faster than the P-code by a factor of 4.5! Perhaps 30% of the code growth can be reduced with a post translation optimizer since many long branches can be converted to short branches and many PSHS D / PULS D pairs are unnecessary (we can leave the result in D). Even with optimization, without some form of memory management, it will not be possible to convert large programs like the Pascal compiler or editor entirely to native code even if the interpreter is removed from memory. However, if the old adage "20% of the code uses 80% of the time" is true, a native code translator would achieve some tremendous speed inprovements even in large programs, by converting time consuming procedures to native code. Partial native code translation could use an unused P-opcode to inform the P-interpreter to switch to native code (the Warp 1 instruction!).

Instruction	Byte Ratio *	Cycle Ratio *
PUSHIM	4/1 (4)	8/19 (.42)
LDCN	4/1 (4)	11/47 (.23)
LDCI	5/3 (1.7)	10/52 (.19)
FJB positive	7/2 (3.5)	15/59 (.25)
FJB negative	" "	15/83 (.18)
LDL byte	8/3 (2.7)	21/92 (.23)
LDL word	8/3 (2.7)	22/104 (.22)
average	6.1/2.1 (2.9)	14.6/65.1 (.22)

* ratios show: native code / P-code

Table 5: **Statistics showing byte cost and speed gain of native code over P-code.**

Conclusion

This article has contrasted MC6809 and 6502 implementations of the UCSD Apple Pascal P-machine. My primary goals were to demonstrate just how easy the MC6809 is to program, and its byte and time efficiency relative to the 6502. The secondary goal of the article was to inspire others to work on native code implementations of Pascal, C, and other languages for MC6809-based systems. Finally, I hope to encourage all 6502 users to take advantage of the added power available with MC6809 boards and systems.

Tom Whiteside is an engineer in the processor design group at Motorola. He may be contacted at Motorola, Inc., MD M2880, 3501 Ed Bluestein Blvd., Austin, TX 78721

MICRO

AARDVARK — THE ADVENTURE PLACE

ADVENTURES FOR OSI, TRS-80, TRS-80 COLOR, SINCLAIR, PET, VIC-20

ADVENTURES — Adventures are a unique form of computer game. They let you spend 30 to 70 hours exploring and conquering a world you have never seen before. There is little or no luck in Adventuring. The rewards are for creative thinking, courage, and wise gambling — not fast reflexes.

In Adventuring, the computer speaks and listens to plain English. No prior knowledge of computers, special controls, or games is required so everyone enjoys them—even people who do not like computers.

Except for Quest, itself unique among Adventure games, Adventures are non-graphic. Adventures are more like a novel than a comic book or arcade game. It is like reading a particular exciting book where you are the main character.

All of the Adventures in this ad are in Basic. They are full featured, fully plotted adventures that will take a minimum of thirty hours (in several sittings) to play.

Adventuring requires 16k on Sinclair, TRS-80, and TRS-80 Color. They require 8k on OSI and 13k on VIC-20. Sinclair requires extended BASIC.

TREK ADVENTURE by Bob Retelle — This one takes place aboard a familiar starship and is a must for trekkies. The problem is a familiar one — The ship is in a "decaying orbit" (the Captain never could learn to park!) and the engines are out (You would think that in all those years, they would have learned to build some that didn't die once a week). Your options are to start the engine, save the ship, get off the ship, or die. Good Luck.

Authors note to players — I wrote this one with a concordance in hand. It is very accurate — and a lot of fun. It was nice to wander around the ship instead of watching it on T.V.

CIRCLE WORLD by Bob Anderson — The Alien culture has built a huge world in the shape of a ring circling their sun. They left behind some strange creatures and a lot of advanced technology. Unfortunately, the world is headed for destruction and it is your job to save it before it plunges into the sun!

Editors note to players — In keeping with the large scale of Circle World, the author wrote a very large adventure. It has a lot of rooms and a lot of objects in them. It is a very convoluted, very complex adventure. One of our largest. Not available on OSI.

HAUNTED HOUSE by Bob Anderson — This one is for the kids. The house has ghosts, goblins, vampires and treasures — and problems designed for the 8 to 13 year old. This is a real adventure and does require some thinking and problem solving — but only for kids.

Authors note to players—This one was fun to write. The vocabulary and characters were designed for younger players and lots of things happen when they give the computer commands. This one teaches logical thought, mapping skills, and creativity while keeping their interest.

DERELICT by Rodger Olsen and Bob Anderson — For Wealth and Glory, you have to ransack a thousand year old space ship. You'll have to learn to speak their language and operate the machinery they left behind. The hardest problem of all is to live through it.

Authors note to players — This adventure is the new winner in the "Toughest Adventure at Aardvark Sweepstakes". Our most difficult problem in writing the adventure was to keep it logical and realistic. There are no irrational traps and sudden senseless deaths in Derelict. This ship was designed to be perfectly safe for its' builders. It just happens to be deadly to alien invaders like you.

NUCLEAR SUB by Bob Retelle — You start at the bottom of the ocean in a wrecked Nuclear Sub. There is literally no way to go but up. Save the ship, raise her, or get out of her before she blows or start WWIII.

Editors note to players—This was actually plotted by Rodger Olsen, Bob Retelle, and someone you don't know — Three of the nastiest minds in adventure writing. It is devious, wicked, and kills you often. The TRS-80 Color version has nice sound and special effects.

EARTHQUAKE by Bob Anderson and Rodger Olsen — A second kids adventure. You are trapped in a shopping center during an earthquake. There is a way out, but you need help. To save yourself, you have to be a hero and save others first.

Authors note to players — This one feels good. Not only is it designed for the younger set (see note on Haunted House), but it also plays nicely. Instead of killing, you have to save lives to win this one. The player must help others first if he/she is to survive — I like that.

PYRAMID by Rodger Olsen — This is one of our toughest Adventures. Average time through the Pyramid is 50 to 70 hours. The old boys who built this Pyramid did not mean for it to be ransacked by people like you.

Authors note to players — This is a very entertaining and very tough adventure. I left clues everywhere but came up with some ingenous problems. This one has captivated people so much that I get calls daily from as far away as New Zealand and France from bleary eyed people who are stuck in the Pyramid and desperate for more clues.

QUEST by Bob Retelle and Rodger Olsen — THIS IS DIFFERENT FROM ALL THE OTHER GAMES OF ADVENTURE!!!! It is played on a computer generated map of Alesia. You lead a small band of adventurers on a mission to conquer the Citadel of Moorlock. You have to build an army and then arm and feed them by combat, bargaining, exploration of ruins and temples, and outright banditry. The game takes 2 to 5 hours to play and is different each time. The TRS-80 Color version has nice visual effects and sound. Not available on OSI. This is the most popular game we have ever published.

MARS by Rodger Olsen — Your ship crashed on the Red Planet and you have to get home. You will have to explore a Martian city, repair your ship and deal with possibly hostile aliens to get home again.

Authors note to players — This is highly recommended as a first adventure. It is in no way simple—playing time normally runs from 30 to 50 hours — but it is constructed in a more "open" manner to let you try out adventuring and get used to the game before you hit the really tough problems.

ADVENTURE WRITING/DEATHSHIP by Rodger Olsen — This is a data sheet showing how we do it. It is about 14 pages of detailed instructions how to write your own adventures. It contains the entire text of Deathship. Data sheet - $3.95. NOTE: Owners of OSI, TRS-80, TRS-80 Color, and Vic 20 computers can also get Deathship on tape for an additional $5.00.

PRICE AND AVAILABILITY:

All adventures are $14.95 on tape except Earthquake and Haunted House which are $9.95. Disk versions are available on OSI and TRS-80 Color for $2.00 additional.

Please specify system on all orders

ALSO FROM AARDVARK — This is only a partial list of what we carry. We have a lot of other games (particularly for the TRS-80 Color and OSI), business programs, blank tapes and disks and hardware. Send $1.00 for our complete catalog.

VISA

AARDVARK - 80
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

master charge

Phone Orders Accepted 8:00 a.m. to 4:00 p.m. EST. Mon.-Fri.

TRS-80 COLOR **SINCLAIR** **OSI** **VIC-20**

Atari Character Graphics from BASIC, Part I

by Paul S. Swanson

If you can form your displays using character instead of map-mode graphics, there is a tremendous savings of available memory. Two of the three character modes available to BASIC on the Atari can be easily converted to character-graphics modes. The sample program shows how to define one character and demonstrates a simple character-graphic animation technique.

Character Graphics
requires:
Atari 400 or 800

You can produce fine resolution graphics by using a map mode like mode 7 or 8. But those two modes eat up 4K and 8K of your valuable free RAM. If you have only 16K, this could be a problem. Mode 5 requires only 1K of memory but is not as fine a mode. It would be convenient to get mode 7 resolution using only 770 bytes for the screen, or twice that resolution using only about 1K. And it would be real handy to have not four, but five colors on the screen.

A Low-Memory Alternative to Map Modes

Character graphics has all the advantages mentioned above, plus easier animation in some cases. You can easily see simple character graphics in mode 0. Just type in any letter while you hold down the CTRL key and you will get a shape on the screen instead of a letter. That is the basic idea of character graphics. If you program your own character set, you can form characters that you can assemble into almost any shape you want.

Once you have the character set, which occupies 512 bytes, a character graphics screen in mode 2 requires about 260 bytes. In other words, in the room required for a mode 7 screen, you can fit about seven mode 2 screens.

Modes 1 and 2 both support five colors. There is one background color, which is usually left black, and four foreground colors. As you will see in the accompanying listing, the idea is to move the character set down into RAM and modify some of the characters in it. When you print the character on the screen, the computer looks up the format in the character set and displays whatever it finds there. If you have altered the character set at the right place, it will display a special graphics character instead of the # or % that you PRINTed to the screen.

An Important Restriction

Listing 1 shows a simple way to implement character graphics on the Atari. The first few lines demonstrate how to cope with a restriction that is not simple to obey in BASIC. With this restriction a character set must start on a ½K boundary for modes 1 and 2, or on a 1K boundary in mode 0. When you add a line to a BASIC program or execute an immediate mode command, the BASIC interpreter moves everything around a little bit. Once the program is running, however, everything stays in one place until the program ends or you stop it. The first few lines in this program locate a fixed place on a ½K boundary to put the character set and make sure it will stay there until the progam is stopped.

Reserving Space for the New Character Set

The string S$ is dimensioned to 1024, which is twice what we need for the character set. This guarantees that there will be 512 bytes in it that start on a ½K boundary. The worst case is one where the string starts one byte after a ½K boundary. The first 511 bytes are wasted in order to get to the first byte on a ½K boundary. If you add a line or two to the program, this wasted space will be moved up, so it does actually function as free space in a program you are developing.

The ½K boundary is located with lines 20 and 30. The expression used in line 30 actually finds the page (256 bytes) of the boundary. If you wanted a 1K boundary instead, you would need 2048 bytes in S$ and line 30 would be B = INT(A/1024 + 1)*4. The computer will find the page number to locate the character set in the variable B.

Line 40 locates what byte in the string is to be the start of the character set. This will be different when the program is modified, so it must be calculated. It is the location in memory of the character set (page times number of bytes per page), minus the starting location of the string, plus one.

The next section of the program is really not required for this demonstration, but it is good programming practice to know the contents of the strings you are using. The three statements (lines 50 to 70) simply clear the entire string to ASCII code zeroes. If you are going to use S$ for other things in the program, you could alter the 1024 in line 60 to the length you made S$ so that the entire string is cleared.

Stealing the Character Set

Lines 80 through 100 move the existing character set down from the operating system into user RAM. There is a full 1024-byte character set at that location. If you want lower-case letters instead of upper-case, add 512 to the 57344. Then alter the FOR statement to start at byte 8 (FOR I = 8 TO 511) so that the heart-shaped character is not moved down into the character reserved for a space.

The disadvantage here is the time involved in moving the character set. If you want to put in an entirely customized set, you don't need to do this. However, if you are using text in the display, it is much easier to steal all of the letters, numbers, and common punctuation marks already in the set so that you don't have to reinvent them.

This demonstration program uses only one special character. The # sign is replaced by the new character. Once

Listing 1

```
1 REM *** Custom Character Set ***
2 REM *** Demonstration Program ***
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
```

Listing 1 *(continued)*

```
159 REM --- Fill in rows ---
160 FOR I=1 TO 2
170 ? #6;"# # # # # # # # # #"
180 ? #6;"                  "
190 ? #6;"■ ■ ■ ■ ■ ■ ■ ■ ■ ■"
200 ? #6;"                  "
210 NEXT I
217 REM
218 REM
219 REM --- POKE character base ---
220 POKE 756,B
227 REM
228 REM
229 REM --- Make birds "fly" ---
230 FOR I=1 TO 100:NEXT I
240 SOUND 0,10,0,6
247 REM
248 REM
249 REM --- Wings down ---
250 S$(CBASE+25,CBASE+25)=CHR$(0)
260 S$(CBASE+26,CBASE+26)=CHR$(231)
270 SOUND 0,0,0,0
272 POSITION 5,5
274 ? #6;"DOWN"
280 FOR I=1 TO 100:NEXT I
290 SOUND 0,10,0,6
297 REM
298 REM
299 REM --- Wings up ---
300 S$(CBASE+25,CBASE+25)=CHR$(195)
310 S$(CBASE+26,CBASE+26)=CHR$(36)
312 POSITION 5,5
314 ? #6;"UP  "
320 SOUND 0,0,0,0
330 GOTO 230
1000 DATA 0,195,36,24,24,36,0,0
```

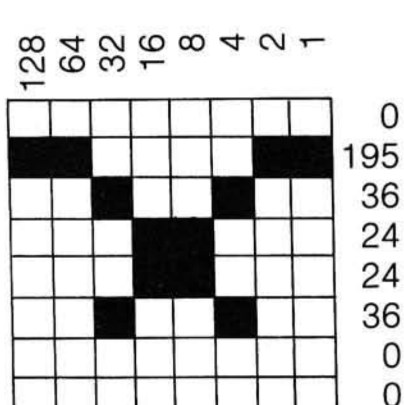
the modifications are done, the special character invented will be displayed when the program prints a #, a CTL-C or an inverse video of either of those two characters. Those four ways to print the character give it the four possible colors. All of the dot locations you define in the character become the color determined by the way you print it, while all of the no-dot locations are displayed in the background color.

There is one other restriction with this type of character graphics: each character can be only one color. You can get around this by using either one of two special modes not directly available to BASIC. To do this, you need to put together your own display list and adhere to other restrictions.

Creating a New Character

Figure 1 shows the 8 × 8 grid used to create the special character. All characters on the Atari in modes 0, 1, and 2 use 8 × 8 grids. When you form the character in the grid, each horizontal line in the grid must be converted to a value that you will be storing in one byte. The values for each column are shown (128, 64, 32, etc.) at the top of the column. For each row, add the numbers that are at the top of each column containing a one. For example, the 24 indicated for the fourth row is the result of adding 16 + 8. The resulting numbers may be placed in order in a DATA statement (see line 1000) so that you may use a READ statement and a CHR$ function to place the values into their proper places in the character set.

Figure 1



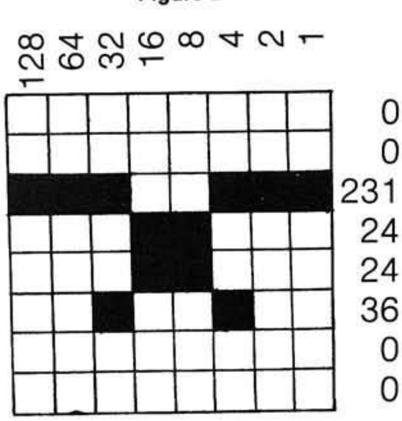
The proper place for a character must also be calculated. For 64-character sets, which is what we are using, the calculations are rather simple. First, subtract 32 from the ASCII value of the character. (You can look up these values in your BASIC Reference Manual.) The value for a space is 32, which gives you zero. No characters with ASCII values less than 32 or greater than ASCII value 95 may be used in these two modes. Now take your result and multiply it by eight. The character occupies eight consecutive bytes starting at that location. The space, for example, occupies bytes zero through 7. The # sign has an ASCII value of 35. Subtract 32 and multiply by eight to get 24. Therefore the character is in bytes 24 through 31. To get the values within the S$ string, just add CBASE, which was calculated when the ½K boundary was determined. As you can see in lines 110 through 140, the values are being READ into that area in S$.

Setting Up the Screen

Line 150 declares GRAPHICS 2 without a text window. To eliminate the text window, you add 16 to the mode number. This causes BASIC to figure out where to put the actual screen in memory and also leaves the work of forming the display list to BASIC. If you form your own screen area and display list, you cannot use PRINT to put the characters on the screen because BASIC doesn't know where the screen is. There are ways around that, too, but this program will not bother with them because it is usually not required.

As you can see in lines 160 through 210, line printers don't print all of the values required in this program. The first line of the display, line 170, is correct. Line 180 has spaces under each # and a CTRL-C under each space in line 170. Lines 190 and 200 are identical to lines 170 and 180, except inverse video is used for the # and CTRL-C characters. This loop will display two sets of these four rows, giving eight rows on the final screen.

Figure 2



At this point, the Atari is still referring to the standard character set in the operating system. To tell the computer where our new set is, we must POKE the page number of our modified set into location 756. We have been maintaining that value in variable B, so line 220 can do this easily.

Animation

Now that everything is set up, the screen will display the new characters. The rest of the program is a crude animation routine to make the figures fly for us.

The bird character starts with its wings up. Notice in figure 2 that the only difference between this "wings-up" character and the "wings-down" character is two bytes. By altering these two bytes in the character set (lines 250 and 260), every one of these figures on the screen will appear to move. Lines 300 and 310 reverse this so that the wings go back up. There are delays in several places in this loop as well as some SOUND statements. Lines 262 and 312 make the words "UP" and "DOWN" appear in the middle of the display at the proper times. Try that in a map mode display!

Beyond the BASICs

As you experiment with character graphics, you will soon discover that this is only a very crude example of what can be done. If you use your own display list instead of the one BASIC establishes for you, you can do some fancier things. For example, instead of having to move the objects one character-width at a time, there is a fine scrolling function, both vertical and horizontal, for movements less than a character-width.

There are also modes that are not directly accessible to BASIC. Two of these are character graphics modes that allow multicolored characters with up to three foreground colors plus the background color. You can get up to five colors on the display with these two modes. With a little extra work, my demonstration program can be modified to use player/missile graphics. The player/missile area must be fixed in relation to a 1K or 2K boundary, depending on the desired player resolution. Modifying line 30 will allow you to accommodate that. If you want to use only the four players and not the missiles, you can have the character set occupy the lower half of a 1K block by locating a 1K boundary at line 30. The upper half of the 1K can be used for two-line resolution players.

One of the best examples of character graphics can be seen in a game by Chris Crawford called "Eastern Front" (available through the Atari Program Exchange). The program has a map of Russia that is larger than the screen. You use the joystick to roam around on the map. The map moves smoothly across the screen using the fine scrolling function described above. There is a movable square indicator (a player) on top of the map, and text on the screen.

The easiest way to use character graphics is to set up a background screen for players and missiles. In games, you quite often have a case where the background is set and you use players in the foreground to play the game. You can also change screens fairly rapidly by storing your screens in strings. Just PRINT them to the screen when you want to change screens. This is much faster than redrawing a screen in one of the map modes.

The more serious applications for character graphics should also be noted. Schematics can easily be implemented — each item on the schematic may be one of the characters in the set. Maps can be constructed the same way, as in the "Eastern Front" example.

Using the sample program as a starting point, there are millions of displays you can create using character graphics. If you have a graphics program in mind, you can probably write it using character graphics and consume a lot less memory than you would with map modes.

Next month, Part II will show you how to move the bird around on the screen using fine scrolling. You will learn how to locate and modify the display list to do the fine scrolling.

Contributing Editor Paul Swanson is an independent software consultant and author of *Microcomputer Disk Techniques* — newly released from BYTE books. He has a wide range of experience with Atari and other microcomputers.

New Atari Column Begins Next Month

Paul Swanson's column covering Atari computers begins next month in MICRO. You may write to him at the address below:

Paul Swanson
c/o MICRO Magazine
P.O. Box 6502
Chelmsford, MA 01824

MICRO

A Homespun 32K Color Computer

by Ralph Tenny

Close-up photos detail a relatively simple memory expansion from 16K to 32K in the Color Computer. In addition, two memory diagnostic programs are furnished. The first tests normal memory and aids the user in verifying correct operation of the new memory ICs to be aded. A second program allows testing of the expanded memory bank if one of the new ICs does not work properly in the expanded memory areas.

Homespun
requires:
TRS-80C
with 16K memory and CBUG Monitor Program

You can either buy or build a 32K TRS-80C Color Computer. I purchased a 16K machine as soon as one was available, and planned to expand the memory when it became apparent how technically easy it would be. But note that the procedures I describe here not only expand the memory of the Color Computer, but leave it in a condition such that Radio Shack will likely refuse to repair your computer if something fails. So, if you feel, as I do, that you are capable of repairing your own computer, there is no reason for you to hesitate. Otherwise, you face a hard decision when planning to add memory to your Color Computer.

Before doing any work on your computer, you should get a copy of the Service Manual for the Color Computer (cat. # 26-3001/3002). After studying the Service Manual, review the following information. Also, to use the accompanying memory diagnostic programs, you need CBUG, the assembly-language monitor from MICRO WORKS, P.O. Box 1110, Del Mar, CA 92014, (714) 942-2400. The cost is $29.95, and worth it.

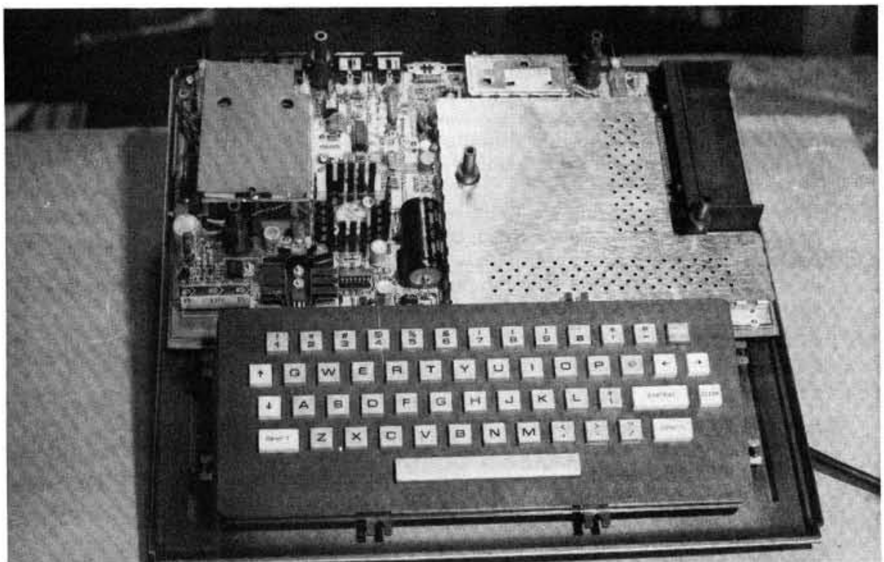
Obviously, you will need memory devices if you plan to add memory to your computer. Probably all the many sources of expansion memory that advertise in the hobby computer journals will have good parts. I recommend that you specify parts with 200 ns access time, just to have some margin. These will cost slightly more; remember, you will be saving the difference between approximately $20 and $100 if you do the work yourself, so don't skimp on memory specifications. Once you have the new parts, install them in your computer and use if for several days to "burn in" the expansion memory. Then you will have less chance of experiencing an early failure just after you install the new memory.

To open the computer, place it upside down on your workbench. Be sure to avoid static discharges while you work: use a grounding strap for your arm — just connect a wire with 100k ohms series resistance to a good ground and attach it to your wrist with a metal strap. Also, use a grounded metal plate to work on.

As you look at the computer, you will note six visible holes that have screws in them; one other screw is under the factory seal in the middle of the case. Loosen these screws and place tape over the holes so the screws won't fall out, then turn the computer upright. The top of the computer will now lift off; photo 1 shows what you will see. The memory is under the RF shield, so carefully remove the shield. It is a friction-fit via many spring clips along the edge; lift each edge of the shield a little, working around the perimeter. Eventually, the shield will lift straight off without damage.

With the shield off, you will see something like photo 2. The eight ICs

Photo 1. Front view of the Color Computer. The keyboard lifts off to clear the edge of the RF shield, which must be removed.



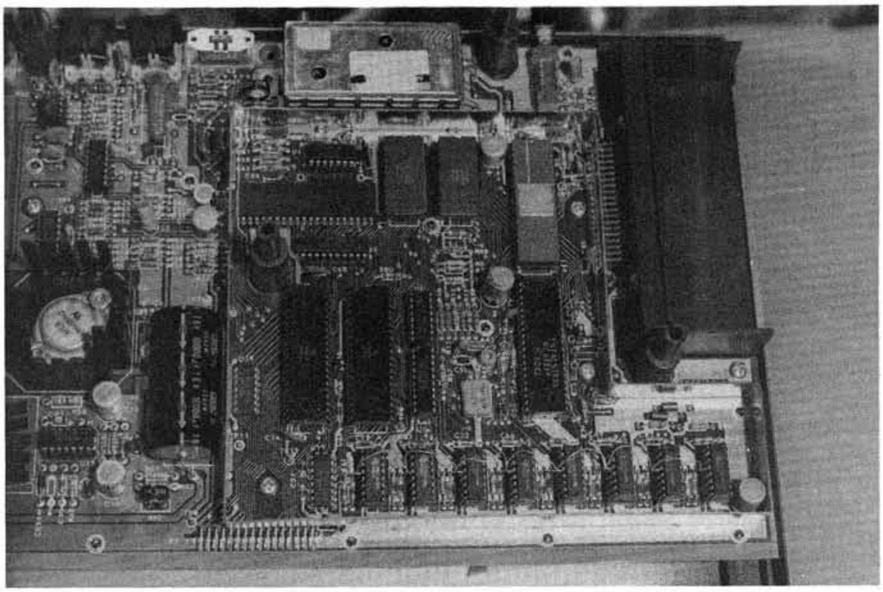


Photo 2. View of the CPU section with the RF shield removed. The memory devices are the right-hand eight chips along the front.

along the front edge of the RF can are the memory devices. Replace these eight parts with your new memory parts; note exactly how each chip is oriented as you pull it out, so the new part can be plugged in exactly the same way. Most likely the sockets will grip the IC pins tightly, so use an IC extractor and lift the parts straight out of the socket. Do not try to pull ICs with your fingers. Not only are you likely to bend the pins by pulling the chip out an angle, but about 50% of the time the IC will rotate and jam several pins into your thumb!

Plug each removed IC into the same non-conducting foam the new parts were shipped in. Install the new parts and apply power to the computer. If it boots and gives the normal prompt, all is well. If it does not, turn power off immediately. Check each part to see if it is hot; if none are hot, maybe no damage was done. Double-check the pins and the IC orientation; as a last resort, try the original memory devices. If it powers up properly with the original parts, and no bent pins, etc., were discovered while checking, replace one memory part at a time and try to power up. If one or more parts were marginal, they will be revealed when the working computer stops.

Once you verify that all the new parts work, allow the computer to run for several days to see if it will keep working. After this test period, look closely at photo 2 and then at photo 3. Photo 3 shows that each new part has been slipped over the IC beneath it, and all pins contact corresponding pins on

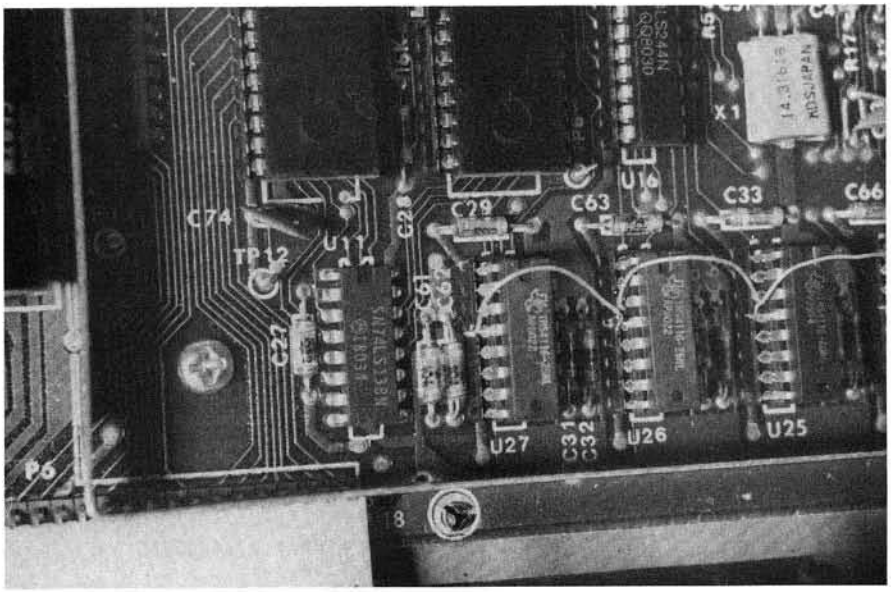


Photo 3. Close-up of the added memory devices, showing the piggyback details and the pin-4 connections.

the lower chip except pin 4, which has been bent out at an angle so it cannot touch. All these #4 pins must be connected together and to pin 35 of U10 (see Sheet 1 of the computer schematic in the Service Manual). I used #30 wire, wrapped lightly around each pin in turn and soldered to pin 35 of U10 with a low wattage iron. If you use #30 wire-wrap wire and work on the metal plate to avoid static discharge, it will be easy to tie the memory parts together. Simply leave a long pigtail to reach U10. Many people suggest soldering each new part to the IC it is riding piggyback, but a number of computers in this area use a friction fit instead. To accomplish this, bend each pin of the piggyback chip inward slightly so it drags heavily on the corresponding pin of the host chip. If you use care in mounting these parts, they seem to work well. At least two computers nearby have worked over a year with this mounting method.

Suppose that you do what I have described, and the computer doesn't boot up after the piggyback operation is finished? That happened to me, but I had purchased expansion devices from a friend who had verified their operation in a different computer. We swapped parts until I had ones which worked in the main bank; one of them still didn't work in the piggyback bank. To identify which one, I wrote a special memory test program that bypassed the response of one device at a time.

But you say, "How can you test the memory if the computer won't boot with the memory installed?" Well, the reason the computer won't boot is that it finds a bad memory location in the middle of a memory block and shuts down without finishing the boot operation. If you unplug any one of the piggyback devices, it will assume that the memory ends with its normal 16K complement of memory, and complete the boot.

So, load CBUG and CBUGMEM and make one tape containing both programs. Leave any one of the added ICs unplugged, and run the special diagnostic (J 0DAB) after inserting the correct bit pattern in memory location $0012. For example, assume that the left-hand memory device isn unplugged. Enter 7F in $0012, then enter J 0DAB. The test will either run or indicate an error; if it is OK, power down, insert the first part and disconnect the second. I recommend that you simply lift the IC off, and slip paper under it so it can't fall into the computer.

Once the entire 32K memory is running, you may want to run the first part of CBUGMEM to test the entire block. Remember to leave some room for the processor stack (test only from $0F00 to $7000, for example). Also remember that the display page and many program variables are located below $600, and the CBUG and CBUGMEM are located between $0600 and $0E25. After this test, enjoy!

Contact the author at P.O. Box 545, Richardson, TX 75080.

Listing 1

```
CBUGMEM                        4-10-82 TSC ASSEMBLER

            *REFERENCE: CBUG, COPYRIGHT 1981 BY
            *THE MICRO WORKS, INC. AND RAMTST2,
            *COPYRIGHT 1981 BY RALPH TENNY.
            *REVISED 4/82 BY RALPH TENNY
            *
            * THIS PROGRAM SEGMENT SUPPLEMENTS CBUG, GIVING IT
            * TWO MEMORY DIAGNOSTIC TESTS.

            *****REFERENCES FROM CBUG*****

     0088   CURPTR  EQU    $88        CURSOR POINTER
     00FB   PARAM   EQU    $FB        BUFFER FOR OUTHEX
     05FF   SCREND  EQU    $05FF      END OF DISPLAY BUFFER
     0618   ENTRY   EQU    $618       IN TO CBUG INNEY
     0627   HEX     EQU    $627       IN TO CBUG OUTHEX
     0651   BADDR   EQU    $651       READ BINARY ADDRESS
     0661   BYTE    EQU    $661       GET A BYTE FROM KEYBOARD
     067F   PCRLF   EQU    $67F       PRINT CRLF
     06AE   OUTHEX  EQU    $6AE       PRINT A BYTE
     06BD   OUTS    EQU    $6BD       PRINT A SPACE
     07A5   INT     EQU    $7A5       INTERRUPT ENTRY
     07A9   WARMS   EQU    $7A9       WARM START ENTRY
     07C9   GETADR  EQU    $7C9       GET TWO ADDRESSES
     07D9   REG     EQU    $7D9       PRINT STACK
     05E0   LSTLIN  EQU    $05E0      SCREEN+$200-32
     A000   POLCAT  EQU    $A000      (IND) POLL KEYBOARD
     A1B1   KBDIN   EQU    $A1B1      GET CHARACTER FROM KEYBOARD
     A30A   ROUTEY  EQU    $A30A      OUTPUT CHARACTER TO SCREEN
     A390   GETLIN  EQU    $A390      GET LINE INPUT
     A46C   CASOUT  EQU    $A46C      CASSETTE OUTPUT
     AD19   NEWCMD  EQU    $AD19      "NEW" COMMAND
     FF00   KBD     EQU    $FF00      KEYBOARD PIA ADDRESS
     FF20   PIA     EQU    $FF20      RS232 PIA ADDRESS
     FFC6   A6883   EQU    $FFC6      DISPLAY CONTROL
            *
            *THIS PROGRAM SEGMENT SUPPLEMENTS CBUG, GIVING IT
            *A MEMORY DIAGNOSTIC CAPABILITY.
```

(Continued)

Listing 1 *(continued)*

```
                          *****REFERENCES FROM CBUG*****
                          *
                          * NOTE: CHECK THESE VALUES AGAINST YOUR VERSION
                          * OF CBUG AND CHANGE IF NECESSARY.
                          *
                          *****EQUATES AND BUFFER DEFINITIONS FOR CBUMEM*****

0010                               ORG   $0010
0010 00                   DPLCH    FCB   0        STORAGE FOR DISPLAY CHARACTER
0011 00                   LINUM    FCB   0        RESERVE BUFFER
0012 00                   MEMLIN   FCB   0        COUNTER FOR MEMORY CHARACTERS
0013 00                   TXTCHR   FCB   0        BUFFER FOR INPUT CHARACTERS
0014 0000                 TXTEND   FDB   0        END OF TEXT BUFFER
0016 0000                 MEMPTR   FDB   0        NEXT MEMORY LOCATION
0018 0000                 DXBFR    FDB   0        TWO-BYTE SPARES
001A 0000                 DDBFR    FDB   0
001C 0000                 DYBFR    FDB   0
001E 0000                 DUBFR    FDB   0
0020 0000                 DSBFR    FDB   0
0022 0000                 LIST     FDB   0

0D30                               ORG   $D30
                          *THIS SEGMENT IMPLEMENTS A MEMORY DIAGNOSTIC
                          *FOR THE TRS-80C COLOR COMPUTER.

0D30 DE    18             MTST     LDU   DXBFR    GET # OF REPEATS
0D32 9E    20             BEGIN    LDX   DSBFR    GET START ADDRESS
0D34 4F                            CLRA           MAKE "ZERO" PATTERN
0D35 A7    84             OUT1     STA   0,X      WRITE PATTERN TO MEMORY
0D37 9C    1E                      CMPX  DUBFR    TEST FOR LAST LOCATION
0D39 27    08                      BEQ   TEST2    IF LAST, GO TO NEXT TEST
0D3B A1    80                      CMPA  0,X+     RAM IMAGE OK?
0D3D 26    58                      BNE   ERROR    REPORT IF BAD
0D3F 9C    1E                      CMPX  DUBFR    TEXT FOR LAST LOCATION
0D41 25    F2                      BLO   OUT1     OR GO UNTIL DONE
0D43 9E    20             TEST2    LDX   DSBFR    GET START ADDRESS AGAIN
0D45 86    80                      LDA   #$80     BEGIN WALKING ONES PATTERN
0D47 A7    84             OUT2     STA   0,X      AND WRITE IT OUT
0D49 27    11                      BEQ   TEST3    AND LEAVE IF DONE
0D4B A1    84                      CMPA  0,X      OTHERWISE, TEST THIS LOCATION
0D4D 26    48                      BNE   ERROR    REPORT ANY GOOFS
0D4F 30    01                      LEAX  1,X      BUMP THE POINTER
0D51 9C    1E                      CMPX  DUBFR    TEST FOR LAST
0D53 25    F2                      BLO   OUT2     LOOP UNTIL DONE
0D55 9E    20                      LDX   DSBFR    LOAD START POINTER
0D57 46                            RORA           THEN SHIFT THE PATTERN
0D58 81    80                      CMPA  #$80     TEST FOR LAST PATTERN
0D5A 26    EB                      BNE   OUT2     AND LOOP UNTIL DONE
0D5C 9E    20             TEST3    LDX   DSBFR    GET THE START AGAIN
0D5E 86    FF                      LDA   #$FF     AND A PATTERN
0D60 A7    84             OUT3     STA   0,X      WRITE ALL ONES
0D62 9F    14                      STX   TXTEND   SAVE THE POINTER
0D64 A1    84                      CMPA  0,X      TEST LOCATION FOR
                                                  CORRECT PATTTERN
0D66 26    2F                      BNE   ERROR    AND REPORT ANY ERROR
0D68 9E    20                      LDX   DSBFR    GET THE START POINTER AGAIN
0D6A C6    01                      LDB   #$01     GET PATTERN FROM LAST TEST
0D6C E1    80             READIT   CMPB  ,X+      AND TEST FOR CHANGE
0D6E 27    FC                      BEQ   READIT   IF OK, KEEP TESTING
0D70 30    1F                      LEAX  -1,X     OTHERWISE, BACK UP
0D72 9C    14                      CMPX  TXTEND   AND CHECK POINTER
0D74 27    10                      BEQ   TSTCEL   IF TEST LOCATION, SKIP
0D76 9C    1E                      CMPX  DUBFR    AT END?
0D78 27    02                      BEQ   GOMORE   BUMP THE "REPEAT" COUNTER
0D7A 26    1B                      BNE   ERROR    IF NOT, OOPS!
0D7C 33    41             GOMORE   LEAU  1,U      COUNT REPEATS
0D7E 1183  0000                    CMPU  #0       AND TEST FOR END
0D82 27    08                      BEQ   EXIT3    STOP IF TIRED
0D84 20    14                      BRA   REPORT   OTHERWISE, REPORT PROGRESS
0D86 E7    84             TSTCEL   STB   0,X      REWRITE ORIGINAL PATTERN
0D88 30    01                      LEAX  1,X      BUMP POINTER
0D8A 20    E0                      BRA   READIT   AND KEEP TESTING
0D8C 7E    07AD           EXIT3    JMP   WARMS+4  IF DONE, EXIT
0D8F E7    84             NEXT     STB   0,X      RESTORE NORMAL PATTERN
0D91 9E    14                      LDX   TXTEND   GET WRITE POINTER
0D93 30    01                      LEAX  1,X      AND POINT TO NEXT LOCATION
0D95 20    C9                      BRA   OUT3     THEN RESUME TESTING
0D97 7E    07A5           ERROR    JMP   INT      PRINT REGISTERS AND EXIT
0D9A 1F    30             REPORT   TFR   U,D      EXTRACT COUNT VALUE
0D9C 97    FB                      STA   PARAM    GET HI BYTE
0D9E BD    06AE                    JSR   OUTHEX   AND PRINT IT
0DA1 D7    FB                      STB   PARAM    REPEAT FOR THE
0DA3 BD    06AE                    JSR   OUTHEX   LO ORDER BYTE
0DA6 BD    06BD                    JSR   OUTS     PRINT A SPACE
0DA9 20    87                      BRA   BEGIN    BACK TO WORK!
```

Listing 1 *(continued)*

```
                    *
                    * THIS PROGRAM IMPLEMENTS A SPECIAL MEMORY DIAGNOSTIC
                    * FOR THE TRS-80C COLOR COMPUTER, RUNNING UNDER CBUG.
                    * TO COMPLETELY TEST AN EXPANSION BLOCK OF MEMORY,
                    * REMOVE ONE RAM DEVICE FROM THE EXPANSION BLOCK, AND
                    * ENTER A MASK VALUE IN MEMLIN, WHERE THE BIT PATTERN
                    * 11111111 IS MODIFIED WITH A 0 CORRESPONDING TO THE
                    * MISSING RAM DEVICE.  RUN THE TEST ONCE FOR EACH CASE
                    * OF MISSING RAM AND REPLACE THE BAD DEVICE INDICATED
                    * BY THE TEST.

0DAB 34     7E      MTST2   PSHS   A,B,DP,X,Y,U
0DAD 9E     20              LDX    DSBFR        GET START ADDRESS
0DAF 1F     10      W1      TFR    X,D          COPY TO D
0DB1 53                     COMB                INVERT LS BYTE
0DB2 E7     80              STB    ,X+          AND WRITE IT TO MEMORY
0DB4 9C     1E              CMPX   DUBFR        TEST FOR LAST ADDRESS
0DB6 25     F7              BLO    W1           AND LOOP UNTIL TRUE
0DB8 8D     61              BSR    DELAY        WAIT FOR BIT DECAY
0DBA 9E     20              LDX    DSBFR        AND START NEW PASS
0DBC 1F     10      R1      TFR    X,D          GET COPY OF START AGAIN
0DBE 53                     COMB                AND INVERT TEST PATTERN
0DBF D4     11              ANDB   LINUM        DELETE MISSING BIT
0DC1 D7     10              STB    DPLCH        SAVE A COPY OF THIS
0DC3 A6     80              LDA    ,X+          READ THE STORED PATTERN
0DC5 94     11              ANDA   LINUM        ALLOW FOR MISSING BIT
0DC7 91     10              CMPA   DPLCH        CHECK AGAINST MASTER
0DC9 26     26              BNE    ERROR2       EXIT IF WRONG
0DCB 9C     1E              CMPX   DUBFR        OTHERWISE, TEST FOR
                                                LAST LOCATION
0DCD 25     ED              BLO    R1           AND SPIN UNTIL DONE
0DCF 9E     20              LDX    DSBFR        START OVER AGAIN
0DD1 1F     10      W2      TFR    X,D          DO THE WHOLE BIT
0DD3 E7     80              STB    ,X+          EXCEPT WRITE IT STRAIGHT
0DD5 9C     1E              CMPX   DUBFR        AND TEST FOR LAST
0DD7 25     F8              BLO    W2           SPIN UNTIL DONE
0DD9 8D     40              BSR    DELAY        AND WAIT AGAIN
0DDB 9E     20              LDX    DSBFR        CHECK AGAIN, FROM THE TOP
0DDD 1F     10      R2      TFR    X,D          SORT OUT THE PATTERN
0DDF D4     11              ANDB   LINUM        JUST LIKE BEFORE
0DE1 D7     10              STB    DPLCH        AND REMEMBER IT
0DE3 A6     80              LDA    ,X+          READ THE MEMORY
0DE5 94     11              ANDA   LINUM        AND BLANK ONE BIT
0DE7 91     10              CMPA   DPLCH        CHECK THE ORIGINAL
0DE9 26     06              BNE    ERROR2       SPLIT IF BAD
0DEB 9C     1E              CMPX   DUBFR        IS IT LAST ADDRESS?
0DED 25     EE              BLO    R2           NOPE, BUZZ OFF!
0DEF 35     FE      EXIT4   PULS   A,B,DP,X,Y,U,PC
0DF1 34     40      ERROR2  PSHS   U            SAVE IT
0DF3 CE     0022            LDU    #LIST        POINT TO A BUFFER
0DF6 ED     C1              STD    ,U++         SAVE A & B
0DF8 AF     C1              STX    ,U++         SAVE X
0DFA 10AF   C1              STY    ,U++         SAVE Y
0DFD 35     40              PULS   U            GET IT BACK
0DFF 20     EE              BRA    EXIT4        AND EXIT NORMALLY
0E01 34     46      COPY    PSHS   A,B,U
0E03 CE     0022            LDU    #LIST        GET DATA ADDRESS
0E06 C6     06              LDB    #06          SET A COUNT
0E08 A6     C0      C2      LDA    ,U+          AND READ IT
0E0A 97     FB              STA    PARAM        THEN SEND IT OUT
0E0C BD     06AE            JSR    OUTHEX       TO THE SCREEN
0E0F BD     06BD            JSR    OUTS         PRINT SPACES BETWEEN
0E12 5A                     DECB                COUNT DOWN
0E13 26     F3              BNE    C2           IN A LOOP
0E15 BD     067F            JSR    PCRLF        AND RESET THE DISPLAY
0E18 35     46              PULS   A,B,U        GET'EM BACK
0E1A 39                     RTS                 RETURN TO CBUG
0E1B 36     06      DELAY   PSHU   A,B          PROTECT THESE
0E1D 5F                     CLRB                SET THE COUNT
0E1E 5C             SPIN    INCB                AND BUZZ
0E1F 26     FD              BNE    SPIN         UNTIL DONE
0E21 37     06              PULU   A,B          TAKE THESE AND RUN
0E23 39                     RTS                 BACK TO MAIN ROUTINE
0E24 00             LAST    FCB    0
                            END
```


MICRO

Reviews in Brief

Product Name:	**Soft-Step**
Equip. req'd:	Apple II with Applesoft or Apple II Plus and 6502 machine language.
Price:	$49.95 plus shipping and handling
Manufacturer:	Accent Software, Inc. 3750 Wright Place Palo Alto, CA 94306
Author:	Lee Stephens
Copy Protection:	Yes

Description: An Applesoft BASIC program debugger.

Pluses: *Soft-Step* fills an inportant void in aiding the Applesoft programmer with a powerful set of debugging functions that allow the programmer to single step through the program, set breakpoints, perform a command (print variables or memory contents) at the breakpoint, display the next line to be executed, or trace program action. The brief 11-page manual describes all of the debug features and includes a command summary. Tables can be created to identify lines for tracing, for breakpoints, or for performing PRINT, PEEK, POKE, or LET operations (called a DO command). The user can display or change these line number tables easily and can create a 96-byte DO table of commands.

Minuses: The TRACE and BREAK tables can only hold ten line numbers each. The manual provides very little information on how the program operates, what areas of memory it uses, etc., although it does indicate the starting and ending point of the program. Interestingly, no zero page or page three locations are used other than those used by Applesoft. Some problems arise in the use of DIMensioned arrays since *Soft-Step* does not clear variables when it begins debugging. A few paragraphs in the manual explain the problems to the user.

Skill level required: Can be used by virtually any Applesoft programmer.

Reviewer: David Morganstein

Product Name:	**LCA-47 Lower-Case Adapter**
Equip. req'd:	TRS-80 Color Computer
Price:	$75.00
Manufacturer:	Micro Technical Products Inc. 814 W. Keating Ave. Mesa, AZ 85202

Description: A hardware adapter that allows the TRS-80C to have a true lower-case character set. It is installed by removing the 6847 video display generator (VDG) in the TRS-80C. The adapter, approximately 2 by 3½ inches plugs into the VDG socket, and the 6847 plugs into the board. It puts the VDG into a mode that allows it to read an external character set. A PROM provides the new characters. Optional character sets are available, as are blank PROMs for designing your own set. The lower case and reverse video sets are switch selectable, and can be changed using internal or external switches. The circuit draws 150 mA from the computer power supply.

Pluses: Provides a true lower-case set for users with word processing applications. Screen color is slightly different from the VDG internal set; letters are yellow on a dark green background. The hardware is fully compatible with Color Scripsit, and all other TRS-80C software. There is no effect upon any of the graphics modes. No system memory is needed, and it is completely transparent to the user.

Minuses: Programs that use the lower-case letters to provide reverse video have those characters changed to lower case *via* the hardware. If this is an important consideration, the board can be deselected *via* the internal switch.

Documentation: An excellent twelve-page installation and operation manual is included.

Skill level required: Installation is easy. Care must be used to keep from bending the pins on the IC.

Reviewer: John Steiner

Product Name:	**Lightpen**
Equip. req'd:	TRS-80 Color Computer
Price:	$40.00 assembled, $19.95 kit
Manufacturer:	Moses Engineering Rt. 7 Greenville, SC 29609

Description: The Moses Engineering light pen is a full-function light pen that interfaces with the TRS-80C joystick. A standard joystick, modified to accept the light pen is included. A kit option is also available that allows you to install the pen and control circuitry in your own joystick. Included in the package is a pen storage box and three program cassettes containing sample programs, and operating instructions.

Pluses: The pen and software provided give the Color Computer owner all of the advantages and applications for the light pen. The program sampler includes some games, an event counter, color detector, and other applications software. All programs are written in BASIC.

Minuses: Though BASIC is relatively fast, calibrating and using the pen is a slow operation. The programs use a machine-language routine to read the joystick ports. The manufacturers say they are considering a machine-language driver that could be accessed from any BASIC programs.

Documentation: Nearly all of the documentation is included in the sample programs. Printed information amounts to only basic instruction on hooking up the pen and starting the programs. There is little information provided for using the pen in your own programs, though the BASIC listings make good study material for those wishing to use the pen for their own applications programs.

Skill level required: None, except for users desiring to apply the pen to their own applications software.

Reviewer: John Steiner

(continued)

Reviews in Brief *(Continued)*

Product Name: **Solicube**
Equip. req'd: Commodore computer with 16K RAM, 40-column screen, and disk.
Price: $50.00, plus $3.00 shipping
Manufacturer: COMQUEST
221 E. Camelback, Suite 1
Phoenix, AZ 85012
(602) 264-0324

Description: This general-purpose Rubik's Cube solver program includes a Rubik's cube, too. Graphics and CB2 sound are used to display movements and speed solution.

Pluses: Surely the best such program now available to Commodore users and much easier to use than books about Rubik's Cube. Several modes of operation, from total helpfulness in graphic and sound displays to high speed expert and experimental modes are included. Background music and foreground musical clues play as the program works. *Solicube* aims to teach users to solve their own cube problems, and should succeed. Excellent motivator for problem student in computer literacy courses.

Minuses: It is not listable, which keeps users from learning the excellent programming techniques used in the programs.

Documentation: Adequate, except for the section on getting the cube into starting position, which is definitely not simple enough for a first grader. From that point on, the program is self-documenting on the screen.

Skill level required: Must be able to read elementary-level English. Usable by children, but not childish.

Reviewer: James Strasma

Product Name: **Global Program Line Editor**
Equip. req'd: 48K Apple II or Apple II Plus with DOS 3.3. (80-column card optional)
Price: $64.95
Manufacturer: Synergistic Software
830 N. Riverside Drive
Suite 201
Renton, WA 98055
(206) 226-3216
Author: Neil Konzen
Copy Protection: No
Language: Machine Language

Description: Extensive revision of Konzen's popular Program Line Editor (PLE) adding global editing capabilities including search and replace, search and display, and search and edit; a type-ahead buffer; and DOS mover to the powerful line editing features of the PLE. Escape key macros may be defined or a built-in library utilized to reduce strings of commands to two keystrokes. The end result is an expanded environment for writing and/or modifying Integer or Applesoft programs.

Pluses: Konzen has added to the capabilities of the PLE without modifying the command structure, thus making the upgrade fairly painless for those used to the earlier program. The DOS mover and memory management features take full advantage of a RAM card if present.

Minuses: None.

Skill level required: Intermediate to advanced BASIC programmer.

Reviewer: Christopher Wiley, M.D.

Product Name:	**The Slide Show**
Equip. req'd:	Apple II with Applesoft in ROM or RAM card. One or more disk drives. 48K of RAM memory.
Price:	$49.95
Distributor:	Huntington Computing P.O. Box 1235 Corcoran, CA 93212
Authors:	Bruce A. Cash and Robert W. Hench C&H Video 110 West Caracas Avenue Hershey, PA 17033 (717) 533-8480

Description: Creates and presents a slide show of Apple II hi-res screens. It provides transitions between slides (curtain, horizontal and vertical wipes, dissolves, etc., 20 in number); adjustable time per slide, variable for each slide, or manual change selected by keystroke or paddle button; automatic mode with automatic restart option.

Pluses: Two types of overlay transitions where you can lay one frame on top of the other for addition or comparison are given. A very tolerant input protocol makes unintentional mistakes impossible when creating the series. The catalog of slides (hi-res screen files) from source disks is available at any time during series creation. It can use any number of disk drives for up to 75-slide program size.

Minuses: Standard DOS loading is slow and limits the minimum show time to eight seconds. The number of slides is limited to 16 per disk because there is no picture compression scheme. You cannot gracefully interrupt a presentation and restart where you left off, or reverse the direction of the presentation. No provision is made for temporarily changing display parameters. Print drivers are not provided for graphic printing of slides to paper. It only assembles slides created by other means. There is slow and awkward transfer of slides from source disks to presentation disk.

Documentation: Excellent 50-page manual with tutorial insures expert use with no computer background. Demo program included.

Skill level required: No computer programming skills required.

Reviewer: David P. Allen

Product Name:	**Gutenberg**
Equip. req'd:	48K Apple II or Apple II Plus with shift key modification to pin 4 of Game I/O, one disk drive, DOS 3.3, and printer: Centronics 737/739, Epson MX82FT and 100, NEC 8023, C IOTOH, PRO WRITER, F10, or Qume Sprint 5 and 9.
Price:	$315.00
Manufacturer:	Micromation Ltd., 1 Yorkdale Road, Suite 406 Toronto, Ontario Canada M6A 3A1 (416) 781-6675
Author:	John Wagner
Copy Protection:	Yes
Language:	6502 Machine Language

Description: *Gutenberg* is a comprehensive word and graphics processing system allowing complex page layout. Features include: lower case without hardware, font editor (7 × 12 matrix), split screen editing with text transfers between screens, keyboard macro definition, word/ character count, programmable and maskable search/replace, embed and print hi-res graphics images and custom fonts, justification with true proportional spacing, hi-res drawing utility, built-in merge with (Gutenberg) data files, and on-line tutorial.

Pluses: The power and flexibility of this program are enormous, exceeding anything else now available on micros. The editor is fast and very easy to use, and the utilities included with the system could easily sell as a stand-alone package. Format codes can be embedded or stored in separate files and called by name which simplifies use.

Minuses: A custom DOS is used, thus preventing code editing or file transfer to/from other programs. Similarly, custom printer drivers are used allowing the use of only the printers listed above. Finally, learning the format codes is a formidable task made harder by the current documentation. A new documenation manual is expected soon.

Skill level required: Full exploitation of its features requires nearly a typesetter's knowledge of page layout, but straightforward word processing can be done by a beginner.

Reviewer: Christopher Wiley, M.D.

Product Name:	**Understand Yourself**
Equip. req'd:	Apple II or Apple II Plus
Price:	$24.99
Manufacturer:	Huntington Computer P.O. Box 1297 Corcoran, CA 93212
Author:	Tests written by Dr. Harry E. Gunn, programs and concept developed by Mike Taylor
Copy Protection:	Yes
Language:	Applesoft

Description: A series of nine, self-administered personality tests cover the areas of: assertiveness, conscience, manipulation, personal equation, marital and personal adjustment, scale of values, preferred activities, and sexual attitudes.

Pluses: The program attempts to bring the subject of self-evaluation into the computer user's home. It begins with a menu that allows the user to select a test. Each test is prefaced by words of explanation and ends with a summary of the results and their implications.

Minuses: To bring testing to the personal microcomputer, programmers will have to develop new techniques. The method of displaying a question and multiple answers, as is done here, adds little to the cheaper paper form approach. I felt that the choice of words were frequently biased, and the answers sought were predictable. Often the order of the answers was the same from question to question — answer A. was always the traditional one and answer D. the conservative one. The overall effect is fun for the user and not serious self-examination. This would be okay except the program continually comments on the importance of the results and their possible implications regarding change, while telling the user that tests produce "errors" and the results need not be taken seriously.

Skill level required: Easy for any Apple user.

Reviewer: David Morganstein

(continued)

Reviews In Brief *(continued)*

Product Name: **The Menu II**
Equip. req'd: 48K Apple II with one or two disk drives
Price: $39.95
Manufacturer: C&H Video
110 West Caracas Ave.
Hershey, PA 17033
(717) 533-8480

Description: An excellent program for storing and manipulating recipes. It provides features that allow the user to write recipes, categorize them into types (e.g., breakfasts, entrees, beverages, etc.) and then store them with cooking procedure comments. The program is an improvement of a previous version.

Pluses: The program has a printer feature for making hard copies of recipes in 3'' × 5''-card format. There is also a feature for varying servings; i.e., a recipe is automatically adjusted for the number of people the user wishes to serve.

Minuses: The program is not immediately easy to use. Initially, some users may become frustrated, or worse, bored. This deficiency could be eliminated by a less formal, more dynamically written tutorial that entertains while teaching.

Documentation: There is a tendency towards lecturing but this is a minor irritant and the reader will not be disappointed if he perseveres.

Skill level required: No programming knowledge needed. Some familiarity with computers and storage would be helpful.

Reviewer: Chris Williams

Product Name: **Computer Almanac**
Equip. req'd: Apple II or Apple II Plus
Price: $24.99
Manufacturer: Huntington Computing
P.O. Box 1297
Corcoran, CA 93212
1-800-344-5106
Author: David Carman
Copy Protection: Yes
Language: Applesoft

Description: A computerized version of a mini-almanac.

Pluses: A few of the fun features of an almanac have been incorporated into this modern magnetic version of the traditional paper almanac. The program begins with a menu offering the following data delicacies: birthstones and anniversary gifts, windchill factor chart, lightning safety rules (including an update to protect your Apple!), loan amortization tables, biorhythms, a health chapter containing a calorie counter and a pulse checker, a vacation planner, sunrise/sunset times, and weather prediction.

Minuses: The overall effect is entertaining but not terribly informative. Most of the displays are merely hi-res displays (the disk does not use any of the new fast DOS programs to reduce the loading time of the displays). Some of the programs can be exited in mid-stream using the escape key, while others lock you in to a long series of questions and answers. The entire disk can be investigated in about a half hour. No manual is included, or appears necessary.

Skill level required: Easy for any Apple user.

Reviewer: David Morganstein

MICRO

Software Catalog

Name: **Basic[1]**
System: Apple II, TRS-80 Models I and III, IBM PC
Memory: 48K
Language: BASIC
Hardware: Disk system required, printer optional
Description: *Basic*[1] is a powerful, completely-structured extension of the BASIC language. Blocks of code are indicated by a unique indentation convention, eliminating the need for both extraneous statements (such as begin and end) and statement numbers. Comments are right-justified for ease of reading, block indentation reveals the flow of control, procedures are separated by white spaces, and an extensive cross-reference is provided.
Price: $129.00
Includes full documentation and disk in a binder format.
Author: Dr. James L. Schmit
Available:
Delta Micro Systems
P.O. Box 15952
New Orleans, LA 70175

Name: **Critical Path Scheduling**
System: Apple III
Memory: 128K
Language: Business BASIC
Hardware: 132-column printer and second diskette drive or hard drive
Description: *Critical Path Scheduling* is an extremely flexible and easy to operate scheduling aid. Based on an arrow network, the system will assign early and late starts and finishes, float and free float to each task. Date conversion (if desired) is automatic. A bar chart report is available, and day-by-day manpower requirements can be generated.
Price: $495.00
Includes operator's manual, programs, and sample project data.
Author: Dan Sargent
Available:
Great Divide Software
8060 W. Woodard Dr.
Lakewood, CO 80227

Name: **Portrait Subsystem**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Disk drive, game paddles, one monitor, Epson MX-80 with graphics or Centronics 739, Station Master and Dithertizer II.
Description: The *Portrait Subsystem* allows Apple II and Apple II Plus owners to create heat-transferrable portraits onto customized T-shirts, posters, and novelty items. Adult size portrait 9½" × 8".
Price: $1,175.00
Includes Dithertizer II, video camera, Station Master interface card with cable, portrait software, and starter supply set.
Author: David K. Hudson
Available:
Computer Station
11610 Page Service Drive
St. Louis, MO 63141
(800) 325-4019
(orders only)
or from local dealer

Name: **Poor Man's Graphics Tablet**
System: Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Disk drive with DOS 3.3, color monitor or T.V.
Description: *Poor Man's Graphics Tablet* is a precision graphics package which has an almost unlimited palette of colors and 59 textures. It contains a unique marking feature which allows you to trace transparencies overlaid on the screen of your monitor or T.V. Full shape table functions included as well as full manipulation of shapes and pictures.
Price: $49.95
Includes one diskette and complete documentation.
Author: Vincent Arnold
Available:
RCI Marketing
19517 Business Center Dr.
Northridge, CA 91324

Name: **The Cube Solution**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft ROM
Hardware: Disk drive
Description: Master the cube with *The Cube Solution*, Muse's answer to the common cube headache! Display your cube on the screen and follow screen instructions to solve it. Or instruct the program to list each move on the printer and solve the cube in your spare time. The screen displays a cube for you to scramble and solve if you don't have one of your own. With colorful graphics and easy commands, *The Cube Solution* makes solving your cube a pleasurable experience.
Price: $24.95
Includes disk, documentation, lifetime replacement policy.
Author: Leonard Biggerstaff
Available:
Muse Software and computer stores everywhere

Name: **6502 MacroAssembler**
System: Apple II Plus
Memory: 48K
Language: Machine
Hardware: DOS 3.3, one or two drives
Description: The *6502 MacroAssembler* is a full scale assembler capable of handling ASCII code conversions, variables, labels, backward and forward branching/jumping, and can save source and object code. It can compile 50 lines of code in approximately nine seconds. This program also includes an editor and a program lister for formal listings. It will support all 6502 mnemonics.
Price: $40.00
Includes floppy disk, manual, and any future updates.
Author: Daniel A. Syrstad
Available:
Daniel Syrstad
Syntronics
Rte. 3, Box 95
Glenwood, MN 56334
(507) 387-5711

Name: **Home Money Minder**
System: TRS-80 Color Computer
Memory: 32K
Language: Extended BASIC
Hardware: Cassette
Description: *Home Money Minder* can tell you in a nutshell how much money you spend on what, and where your income came from. You do this by recording all of your checkbook's activities — deposits, checks, and charges. By assigning each to an account code the computer can summarize all of your expenses, income, and cash flow. It will help balance the checkbook, and provide such reports as: summary of expenses, summary of income sources, and all transactions for the month and year. Taxes are a snap!
Price: $19.95
Includes cassette and instruction manual.
Available:
Computerware
P.O. Box 668
4403 Manchester Ave.
Encinitas, CA 92024
(714) 436-3512

Name: **Zoom Grafix**
System: Apple II
Memory: 48K
Language: Applesoft and machine language
Hardware: Printer
Description: High-resolution graphics screen printing package with support for a wide range of printers. *Zoom Grafix* allows you to print positive/negative, upright/rotated, in any size or proportion. Any part of the screen can be selected and printed. Automatic centering or selectable margins — keyboard forms control.
Price: $39.95
Includes disk and manual.
Author: Dave Holle
Available:
Phoenix Software, Inc.
64 Lake Zurich Drive
Lake Zurich, IL 60047
(312) 438-4850

(continued)

Software Catalog *(continued)*

Name: **Turtlegraphics**
System: UCSD p-System™
Memory: 48K runtime environment; 64K development environment
Language: Written in UCSD Pascal™
Hardware: 8086, Z80, 8080, 8085, 6502, 9900, 6809, 68000, and LSI-11/PDP-11
Description: *Turtlegraphics* is a machine-independent, adaptable set of library subroutines that produce portable, high-resolution monochrome or color graphics. It provides two-dimensional graphics by programming the activities of a fast-moving "turtle" as it carries a pen about a graphics display.
Price: $75,00
Includes object code for *Turtlegraphics.*
Available:
SofTech Microsystems, Inc.
9494 Black Mountain Rd.
San Diego, CA 92126
(714) 578-6105

Name: **The "Cardboard"**
System: VIC 20 Series
Memory: Any (use the Cardboard to add memory and games)
Description: Expansion motherboard with six slots, switch selectable, system reset button, and Daisy chainable.
Price: $119.95
Includes instructions.
Available:
GOSUB International, Inc.
501 E. Pawnee, Suite 430
P.O. Box 275
Wichita, KS 67211

Name: **Busywork**
System: Apple II with Applesoft in ROM
Memory: 48K recommended
Language: Applesoft
Hardware: DOS 3.3, one disk drive minimum, printer optional
Description: *Busywork* is a collection of several programs and many routines that will aid in the development of business application programs. Routines include input routine, dollar format, data disk, master disk, menu routine, center print, printer/screen, valid date, term edit, right justify, and many more. Also included is a greeting program, main menu program, update system parameters program. All programs fully listable and copyable.
Price: $39.95
Includes copyable diskette in DOS 3.3 and 60-page manual.
Author: Dale Ludewig
Available:
Datam Consultants
P.O. Box 238
Dekalb, IL 60115

Name: **IEP/MS (Individual Education Program/ Management System)**
System: Apple II
Memory: 48KB
Language: Applesoft Compiled (all files are binary)
Hardware: DOS 3.3
Description: An easy to use flexible system designed to help special education administrators create, print, and save IEPs for their special education students. It allows them to construct a large data base of information which is then used to create better IEPs in less time.
Price: $475.00
Author: Donald W. Cahill
Available:
Creative Educational Services
36 River Avenue
Monmouth Beach, NJ 07750
(201) 870-6543

Name: **Type-Test**
System: VIC-20
Memory: 5K
Language: BASIC
Hardware: Cassette drive
Description: This program is designed to improve a student's typing skills. It displays text on the screen for the student to type back. Mistakes are indicated by a tone, and their number is displayed with typing speed at the end of the exercise.
Price: $9.85 - U.S.
$11.90 - Canada
Author: M. James
Available:
MFJ Electro-Enterprises
P.O. Box 13076
Kanata, Ontario
Canada K2M 1X3

(continued)

Name: **MIT's LOGO for Apple**
System: Apple II
Memory: 64K
Language: LOGO
Hardware: Disk and 16K RAM card
Description: This is a 20-program tutorial series with a technical manual for Apple II by Abelson and L. Klotz. Step-by-step introduction is given for LOGO.
Price: $149.95
Includes LOGO, utilities, disk, Alice In Logoland disk.
Available:
Krell Software Corp.
1320 Stoneybrook
Stoney Brook, NY 11790

Name: **States and Capitals Drill**
System: OSI C1P, Superboard
Memory: 8K
Language: BASIC
Hardware: Cassette
Description: More than a drill, this program is fun. After deciding whether you want to be given the capital or the state, you give the answer. Correct answers cause the U.S. flag to wave. Questions are given in a different order each time. Stop at any time and get your grade. What better way to use your C1P?
Price: $10.95 plus $1.50 shipping
Includes cassette, instructions.
Author: Doug Jenkins
Available:
Tripod Productions
Box 71, Rt. 11
Bowling Green, KY 42101

Name: **Brickaway**
System: TRS-80 Color Computer
Memory: 4K and up
Language: Assembler
Description: Using a joystick, the player controls the paddle which knocks a bouncing ball into the rows of bricks. It's very challenging, since the game difficulty increases as the player's skill level increases.
Price: $10.00 ea.
Includes documentation.
Author: Britt Monk CDP
Available:
Veico International
3870 W. 143rd Street
Cleveland, OH 44111

Name: **Gradisk**
System: Apple II, 3.3 DOS
Memory: 48K
Language: Applesoft
Description: *Gradisk* is a grade management program that allows an instructor to record, edit, calculate, and plot up to 25 scores for as many as 200 students. *Gradisk* also lets you compile numerical and letter grades, annotate an individual student's records, calculate weighted averages, class averages, and grade distribution — all with computer speed and accuracy.
Price: $100.00 (tentative)
Includes complete documentation and disk.
Author: Richard Cornelius
Available:
John Wiley & Sons, Inc.
Eastern Distribution Center
Order Processing Dept.
1 Wiley Drive
Somerset, NJ 08873

Name: **Mathemagic**
System: Apple II or Apple II Plus, CPM
Memory: Apple - 48K
CPM - 64K
Language: Apple - Applesoft in ROM
CPM - CPM 2.2 or later MP/M
Hardware: At least one floppy disk drive (DOS 3.2 or 3.3 for Apple; Z-80-based micro for CPM)
Description: *Mathemagic* is a calculating program. Three work areas are displayed on the screen: 1. a menu showing what commands can be given at each stage of the program; 2. a work area showing the formulas being used or created; and 3. an answer space that presents the progressive calculation, and then displays a final result after all the values have been plugged into the formula.
Price: $89.95 - Apple
$99.95 - CPM
Author: Int'l. Software Marketing (I.S.M.)
Available:
Software Distributors
10023 W. Jefferson Blvd.
Culver City, CA 90230

(Continued on next page)

Software Catalog *(continued)*

Name: **Pascal Speed-Up System**
System: Apple II
Language: Apple Pascal or FORTRAN
Hardware: Mill 6809 coprocessor board
Description: A *Pascal Speed-Up System* upgrade is available for the increasing number of business and data base applications that run under Pascal for the Apple II. This program speeds up Pascal by doing all the interpretation of P-codes, including the floating point operations associated with Pascal REAL variables, while the 6502 processor continues doing input and output.
Price: $295.00
Includes disk with software printer spooler, floating point capability, and speed-up software.
Available:
Stellation Two
The Lobero Bldg.
P.O. 2342
Santa Barbara, CA 93120
(805) 966-1140

Name: **PAL (Personal Aid to Learning)**
System: Apple II with Applesoft
Memory: 48K
Language: BASIC/Assembler
Hardware: One or two drives
Description: *PAL* is a diagnostic/remediation program that teaches reading for grades two through six. *PAL* actually diagnoses the cause of reading problems, and provides remediation exercises directly targeted at those problems.
Price: $99.95 - Master Disk Package; $99.95 - Reading Curriculum Package for each grade level; $29.95 - Reading Placement Test; $9.95 - Demo-Disk Package
Author: Dr. Dale Foreman
Stanley Crane
Daniel Myers
Available:
Universal Systems for Education, Inc.
2120 Academy Circle, Suite E
Colorado Springs, CO 80909
(303) 574-4575

Name: **SWiFT**
System: Commodore CBM BASIC 4.0
Memory: 32K
Language: BASIC (with some Assembler)
Hardware: Commodore CBM 8032 with 8050 disk drives
Description: *SWiFT* is an interface between the WordPro (TM by Professional Software) word-processing package and many other packages. Its uses include: 1. translating text files from other word processors to WordPro; 2. translating sequential files output by other packages (database packages, mail lists, etc.) into WordPro-readable text files.
Price: $30.00
Includes disk, manual.
Available:
INI Inc.
4013 Chestnut St.
Philadelphia, PA 19104

Name: **Dueling Digits**
System: Apple II, Apple II Plus
Memory: 48K
Language: Assembly
Description: Entertainment/Education game - The human race is in a dark age, and the art and science of math is but one of many which are lost, though not entirely forgotten. A place deep in the blasted sands of the San Francisco desert is said to hold those ancient secrets, a place called "The Temple of Numbers." People of this future age consider numbers sacred. Using the machines left to them by their ancestors, they struggle for the ultimate spiritual discipline... The balanced expression.
Price: $29.95
Includes software.
Author: Brian Crouch
Available:
Broderbund Software
1938 Fourth Street
San Rafael, CA 94901
(415) 456-6424
or dealers and distributors

Name: **Snake Byte**
System: Apple II, Apple II Plus
Memory: 48K
Language: Assembly
Hardware: One disk drive, keyboard, or Sirius Joyport with Atari Joystick.
Description: What has 48K bytes and is addictive? *Snake Byte*! A game that works like a charm. A tail of Perilous Purple Plums that's ahead of its time. A game you can sink your teeth into. An antidote for boredom. *Snake Byte*. Fangs alot, Sirius Software!
Price: $29.95
Author: Chuck Sommerville
Available:
Sirius Software, Inc.
10364 Rockingham Drive
Sacramento, CA 95827
(916) 366-1195

Name: **Moonbase IO**
System: Atari 800
Memory: 24K
Language: Machine Language
Hardware: Disk Drive (Atari 400 cassette version soon)
Description: *Moonbase IO* combines three exciting arcade adventures in one exciting game. This machine-language program uses advanced graphics and sound effects. It is a voice-activated program. It requires 24K to be used with a disk drive.
Price: $29.95
Includes disk and audio cassette.
Author: John Konopa
Available:
Program Design, Inc.
11 Idar Court
Greenwich, CT 06830

Name: **Lab Statistics Package™, Harmonic Motion Workshop™**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: Disk II
Description: *Lab Statistics Package* provides an easy method of introducing laboratory students to statistical procedures used in the science lab. The concepts presented are applicable to physics, chemistry, biology, engineering, and the behavioral sciences. *Harmonic Motion Workshop* visually presents the concepts associated with simple and damped harmonic motion by the use of high-resolution graphics.
Price: $50.00, $75.00
Includes program diskette and complete documentaion.
Author: Charles Yarger
High Technology Software Products, Inc.
Available:
High Technology Software Products, Inc.
P.O. Box 14665
2201 N.E. 63rd St.
Oklahoma City, OK 73113
(405) 478-2105

MICRO

Hardware Catalog

Name: **RGB Color Board for the Apple II**
Description: A video board for the Apple II, provides red, green, blue, and composite sync signals for use with an RGB monitor. Color graphics or text are superior in resolution and color-quality compared to composite video. It may be used with the VIDEX 80-column board. Modifications are not required of your computer.
Price: $179.00
Includes complete instructions.
Available:
Video Marketing, Inc.
P.O. Box 339
Warrington, PA 18976
(215) 343-3000

Name: **Microcomputer Work Center**
Description: This work center with 30"-deep, 1"-thick wood laminate top is available in widths of 30", 36," and 48." "C"-leg base is made out of heavy-guage tubular steel, and arc-welded for maximum strength. Available for immediate shipment. Accessories available include station return surface, power assembly, and supply carrier.
Price: $245, $255, $270
Includes disk drive and supply carrier.
Available:
Office furniture dealers nationally.
Manufactured by Lusor Corp.
2245 Delany Road
Waukegan, IL 60085

Name: **Escon Interface**
System: IBM Personal Computer
Description: A typewriter interface that allows excellent quality output from the IBM Selectric and Electronic machines.
Price: $599.00
Available:
Escon Products, Inc.
12912 Alcosta Blvd.
San Ramon, CA 94503

Name: **Olympia Portable Computer (OPC)**
System: 6502 microprocessor-based system
Memory: 64K ROM plus 52K RAM
Language: Microsoft BASIC, Extended BASIC, snapFORTH, assembler, and others shortly
Hardware: Supports RS-232C interface, modem, printers, color plotters, TV & I/O adaptors
Description: The *OPC* is a powerful system ideally suited for business applications, schools, or people on the go. The *OPC* comes standard with 4K RAM, 16K ROM, a 65-character fully programmable keyboard and a 26-character liquid display window. The system is totally portable and expandable. A user can start with the primary unit and add system peripherals, application packages and letter-quality printers.
Price: $380.00 retail for primary unit
Includes 4K RAM, 16K ROM, a 10-key calculator, clock/controller, and file system.
Available:
Olympia dealers and major account representatives

Name: **SSD Solid State Disk Emulator**
System: Apple DOS 3.3, Apple Pascal, CP/M operating systems
Memory: Model 2201 - 147K Bytes; Model 2202 - 294K Bytes
Language: Integer BASIC, Pascal, Applesoft, CP/M, FORTRAN
Description: The SSD board plugs into the Apple in any slot and emulates the disk drive. It requires no external power source and the Apple's power supply can maintain up to three of the boards with no additional source. The SSD board may increase the operating speeds up to 1000% depending on the type of program and operation.
Price: $550 for Model 2201
$950 for Model 2202
Includes CP/M, Pascal, and DOS software and documentation.
Available:
Synetix
15050 N.E. 95th
Redmond, WA 98052

Name: **Hayes Smartmodem 1200**
System: RS-232C-compatible
Language: Any using ASCII character strings
Description: The *Hayes Smartmodem 1200* is a 1200/300-baud direct connect modem compatible with RS-232C computers. It features auto-dial/auto-answer, auto-speed selection, Bell 212A compatibility, Touch-Tone or pulse dialing, and a built-in audio speaker for monitoring calls; it operates in full or half-duplex and can be programmed through ASCII character strings.
Price: $699.00 suggested retail
Includes Smartmodem 1200, owner's manual, power pack, modular telephone cable.
Available:
Retail computer stores nationwide.
For nearest dealer contact:
Marketing Services Dept.
Hayes Microcomputer Products, Inc.
5835 Peachtree Corners East
Norcross, GA 30092
(404) 449-8791

Name: **PROM Programmer**
System: Apple II, Apple II Plus
Memory: 48K RAM
Language: BASIC
Hardware: One disk drive
Description: Hardware/software combination for reading PROMs or ROMs, and writing 2K 2716 5V EPROMs. Input can be from keyboard, disk, PROM, or ROM, and in hex or decimal. It can be adapted to program 8K, 16K, and 32K EPROMs. Word-Power also offers some construction kits such as "Computer Security Lock" and "Dual DOS" that use progammed EPROMs. Hardware includes a pad for adding another socket so you can program two EPROMs simultaneously.
Price: $99.95
Includes hardware, software disk, and detailed instructions.
Available:
Word-Power
P.O. Box 736
El Toro, CA 92630

Name: **Hardwood for Hardware**
Description: Worktables and printer tables designed for today's microcomputers. These units are made from hardwood and hardwood veneers. Choice of woods: red oak, birch, cherry, and walnut. Oak and birch are available either stained or natural color. The largest worktable is 46" wide and 26" deep. Printer table will hold letter-quality printers and comes with casters, one paper shelf, and wide slot for paper feed. All units are typing height of 26". All components are shipped knocked down *via* UPS. Easily assembled with modern fittings. A variety of shelf units is available.
Price: $195 plus freight - oak or birch units (cherry and walnut are more).
Includes 46" table with shelf.
Available:
The Wood Works
11th and Haskell
Rt. 2, Box 407
Lawrence, KS 66044
(913) 842-7797

MICRO™

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

6809 Bibliography

72. '68' Micro Journal 4, Issue 6 (June, 1982)

Anderson, Ronald W., "Flex User Notes," pg. 9-13.
A C Dump program for 6809 systems. Suggestions for implementing the program.

Commo, Norm, " 'C' User Notes," pg. 13-16.
Discussion of terminal I/O for the 6809.

Nay, Robert L., "Color User Notes," pg. 16-18.
Quick looks at a number of routines for the 6809 systems.

Harkness, H.L., "Quicksort," pg. 21-24.
Sort routines for 6809 systems.

Jones, R., "Checkers," pg. 29-30.
The game of checkers for 6809 systems.

Nunn, Philip, "Month Number Converter," pg. 31.
For 6809 users using the FLEX Text Processor a routine to convert the month number in the system register to the abbreviation of the corresponding month.

Ousterhout, Roger, "Upsave/Download," pg. 34-36.
Utilities for the 6809.

Kron, Randy L., "Getnum," pg. 37.
A 6809 system for number conversions among binary, octal, decimal, and hexadecimal or other bases.

Hartman, William, "Datecopy," pg. 38-39.
This is an overlay to the FLEX 09 Copy Cmd Ver 1 to maintain the original file creation date during copy.

73. BYTE 7, No. 7 (July, 1982)

Stearman, Colin, "Color Computer Disk System," pg. 312-328.
A discussion of the disk system for the 6809-based Radio Shack Color Computer.

74. Compute! 4, No. 7, Issue 26 (July, 1982)

Givan, Richard and Betty, "IRA Planner," pg. 34-38.
See how IRA accounts compute with this short program for the Radio Shack 6809-based Color Computer.

75. '68' Micro Journal 4, Issue 7 (July, 1982)

Anderson, Ronald W., "Flex User Notes," pg. 9-12.
Assembler techniques for 6809 systems using FLEX, notes on Pascal, set theory, etc.

Cadmus, Ray, "OS-9 Notes," pg. 12-13.
A copy program to copy a directory's worth of files at one time using OS-9 on 6809 micros.

Commo, Norm, " 'C' User Notes," pg. 13-16.
Comparison of how four compilers stack up using a benchmark technique and a 1 MHz 6809.

Nay, Robert L., "Color User Notes," pg. 16-18.
"Spell 'N Fix," a TRS-80C Color Computer dictionary program.

Odneal, Steve, "FMATE Usr Rout.," pg. 18-20.
Information to allow usage of some of the special routines available on the Radio Shack 6809-based Color Computer.

Abrams, Clayton W., "CC-Amateur Radio," pg. 23-27.
Use of the TRS-80C Color Computer for radio amateur CW and RTTY as well as slow scan TV modes. Notes on transmitting and receiving hardware.

Watson, Ernest Steve and Brady, F. Dale, "Home Acct. Prog.," pg. 28-31.
A 6809-based home or small business accounting program running under FLEX2 or FLEX9.

76. 80 Micro, No. 31 (August, 1982)

Miller, Franklyn D., "The Colorful Computer — Part I," pg. 94-103.
Twenty-one graphics programs illustrating the capabilities of the 6809-based TRS-80C of Radio Shack.

Washington, Winifred, Jr., "Cube-80," pg. 106-110.
Let the TRS-80C Color Computer unscramble your Rubik's Cube, with program listing.

White, Richard A., "Color Maze," pg. 188-192.
A game for the color computer.

Becker, Larry D., "Four in One Plus Another," pg. 202-208.
Several games and a utility for the TRS-80 Color Computer.

Ednick, Dave, "Space Duel," pg. 260-262.
A game for the TRS-80C Color Computer.

Kanach, Chuck, "Square Game," pg. 264-266.
An adaptation of the game Merlin for the TRS-80C Color Computer.

Grossbach, Mitchel, "Color Breakaway," pg. 268-269.
A one-on-one hockey simulation using the TRS-80C Color Computer.

Kitsz, Dennis, "80 Applications," pg. 344-355.
A homebrew ROM pack project for the TRS-80C Color Computer.

77. MICRO, No. 50 (July, 1982)

Pass, Edgar, "Computer-Assisted Translation of Programs from 6502 to 6809," pg. 77-90.
Both the 6502 and 6809 are derivatives of the older 6800 microprocessor. Techniques of translation are discussed, along with tables, 6809 routines, and special problems.

Staff, "MICRO Hardware Catalog," pg. 103.
Includes description of GIMIX Multiuser, a 6809 system.

Staff, "MICRO Software Catalog," pg. 110-116.
Several software items for the 6809 are discussed.

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

Sias, Bill, "REMarks," pg. 3-7.
Discussion of FLEX and the Radio Shack Color Computer.

Ventling, J., "George C. Scott Portrait," pg. 15.
A graphics listing for the 6809-based Color Computer.

Howard, Gary, "William Tell Overture," pg. 16-28.
A music routine in machine language for the Radio Shack Color Computer.

Regina, "Cookie File," pg. 29-32.
A recipe program for the 6809-based Color Computer.

Bodnar, David, "Making Education More Colorful," pg. 34-37.
Several listings and procedures for using the Color Computer in education.

Bassen, Howard, "Optimizing High-Resolution Animated Games in Extended BASIC," pg. 39-43.
A tutorial with several listings for the 6809-based Color Computer.

Ostrom, Steven M., "Graphics and Animation for the Color Computer," pg. 49-53.
A four-part tutorial on graphics.

Sias, Bill, "OS9 Corner," pg. 55-57.
A discussion of the OS9 operating system for the 6809-based Color Computer.

MICRO™

Command	MIT[1]	APPLE[2]
TURTLE GRAPHICS:		
Set background color	BACKGROUND or BG *color#*	SETBG *color#*
Clear graphics screen	CLEARSCREEN or CS	CLEAN
Clear graphics screen, home and show turtle	DRAW	CLEARSCREEN or CS
Clear text screen	CLEARTEXT	CLEARTEXT
Show full graphics screen	FULLSCREEN	FULLSCREEN
Show mixed text/graphics	SPLITSCREEN	SPLITSCREEN
Show full text page temporarily	NODRAW or ND	...
Show full text page erasing graphics	...	TEXTSCREEN
Move turtle backwards specified distance	BACK or BK *distance*	BACK or BK *distance*
Move turtle forward specified distance	FORWARD or FD *distance*	FORWARD or FD *distance*
Set turtle direction in degrees	SETHEADING or SETH *degrees*	SETHEADING or SETH *degrees*
Output turtle direction	HEADING	HEADING
Rotate turtle clockwise in degrees	RIGHT or RT *degrees*	RIGHT or RT *degrees*
Output heading from turtle to point x,y	TOWARDS *x y*	TOWARDS [*x y*]
Move turtle to center pointing straight up	HOME	HOME
Move turtle to position	SETXY *x y*	SETPOS [*x y*]
Move turtle horizontally	SETX *x*	SETX *x*
Move turtle vertically	SETY *y*	SETY *y*
Output horizontal position	XCOR	XCOR
Output vertical positon	YCOR	YCOR
Output turtle position x y	...[3]	POS
Output pen up/down, color	...	PEN
Output turtle shown/hidden	...	SHOWNP
Output background color	...	BACKGROUND
Output pen color	...	PENCOLOR
Output pen u/d, turtle s/h, background color, pen color	TURTLESTATE or TS	...
Make turtle disappear	HIDETURTLE or HT	HIDETURTLE or HT
Make turtle appear	SHOWTURTLE or ST	SHOWTURTLE or ST
Set pen color	PENCOLOR or PC *color#*	SETPC *color#*
Set pen down	PENDOWN or PD	PENDOWN or PD
Pick pen up	PENUP or PU	PENUP or PU
Set pen eraser	PC background color	PENERASE or PE
Set pen reverser	PC 6	PENREVERSE or PX
Set pen mode and color	...	SETPEN [*mode color#*]
Turn on point on screen	...	DOT [*x y*]
Set screen boundaries	NOWRAP	FENCE
Set wrap-around screen	WRAP	WRAP
Set non-wrap-around screen with no boundaries	...	WINDOW
Set verticle aspect ratio	.ASPECT *a*	SETSCRUNCH *a*
Output verticle aspect ratio	...	SCRUNCH
ARITHMETIC OPERATIONS:		
Addition	*a + b*	*a + b*
Subtraction	*a − b*	*a − b*
Multiplication	*a*b*	*a*b*
Division	*a/b*	*a/b*
Equals	*a = b*	*a = b*
Greater than	*a*>b	*a*>b
Less than	*a*<b	*a*<b
Squareroot	SQRT *a*	SQRT *a*
Addition of several numbers	(a + b + c...)	SUM *a b c* ...
Multiplication of several numbers	(a*b*c...)	PRODUCT *a b c* ...
Output the integer of a number	INTEGER *a*	INT *a*
Output the integer quotient	QUOTIENT *a b*	QUOTIENT *a b*
Output the integer remainder of an integer quotient	REMAINDER *a b*	REMAINDER *a b*
Round to the nearest integer	ROUND *a*	ROUND *a*
Seed random number generator	RANDOMIZE	RERANDOM
Output random number between 0 and a-1	RANDOM *a*	RANDOM *a*
Output in degrees the angle whose tangent is a/b = n	ATAN *a b*	ARCTAN *n*
Output cosine of angle	COS *a*	COS *a*
Output sine of angle	SIN *a*	SIN *a*

Command	MIT[1]	APPLE[2]
WORD AND LIST OPERATIONS:		
Output character code	ASCII *char*	ASCII *char*
Output character from ASCII code	CHAR *a*	CHAR *a*
Output all but first element	BUTFIRST or BF *object*	BUTFIRST or BF *object*
Output all but last element	BUTLAST or BL *object*	BUTLAST or BL *object*
Output # items in list	...	COUNT *list*
Output t/f if empty element	...	EMPTYP *object*
Output t/f if equal elements	...	EQUALP *obj1 obj2*
Output t/f if item is list	LIST? *object*	LISTP *object*
Output t/f if item in list	...	MEMBERP *object list*
Output t/f if item is number	NUMBER? *object*	NUMBERP *object*
Output t/f if item is word	WORD? *object*	WORDP *object*
Output t/f if item has value	THING? *object*	NAMEP *object*
Output t/f if item is primative	:THING = TEXT:THING	PRIMATIVEP *object*
Output t/f if item is procedure	...	DEFINEDP *name*
Output only first element	FIRST *object*	FIRST *object*
Output only last element	LAST *object*	LAST *object*
Combine first input and list	FPUT *object list*	FPUT *object list*
Combine list and first input	LPUT *object list*	LPUT *object list*
Output nth item of list	...	ITEM *n list*
Output list of items	LIST *obj1 obj2* ...	LIST *obj1 obj2* ...
Combine items into list	SENTENCE or SE *obj1 obj2* ...	SENTENCE or SE *obj1 obj2* ...
Combine words into one word	WORD *word1 word2* ...	WORD *word1 word2* ...
DEFINING AND EDITING:		
Write procedure that defines other procedures	DEFINE *procname list*	DEFINE *procname list*
Define procedure	TO *procname*	TO *procname*
Edit procedure	EDIT or ED *procname*	EDIT or ED *"procname*
End definition	END	END
Output procedure as a list	TEXT *"procname*	TEXT *"procname*
Edit variables or package	EDIT NAMES EDIT PROCEDURES EDIT ALL	EDNS (*package*)[4]
Copy definition to new name	...	COPYDEF *newname name*
Define a label	*name:*	LABEL *"name*
Define a local variable	...	LOCAL *name*
Assign a value to a variable	MAKE *name obj*	MAKE *name obj* or NAME *obj name*
Output the value of a word	THING *word*	THING *word*
CONDITIONALS AND CONTROL FLOW:		
Output true if all are true	ALLOF *pred1 pred2* ...	AND *pred1 pred2* ...
Output true if any are true	ANYOF *pred1 pred2* ...	OR *pred1 pred2* ...
Output true if false, false if true	NOT *pred*	NOT *pred*
Conditional branching	IF *pred list1* (ELSE *list2*)	IF *pred list1* (*list2*)
Conditional testing	TEST *pred*	TEST *pred*
Conditional execution if true	IFTRUE or IFT *list*	IFTRUE or IFT *list*
Conditional execution if false	IFFALSE or IFF *list*	IFFALSE or IFF *list*
Temporary stop	PAUSE	PAUSE
Continue from stop	CONTINUE or CO	CONTINUE or CO
Trace execution	TRACE	...
Turn off tracing	NOTRACE	...
Transfer unconditional	GO *label*	GO *label*
Output result to calling procedure	OUTPUT or OP	OUTPUT or OP
Repeat a loop	REPEAT *a list*	REPEAT *a list*
Execute a list	RUN *list*	RUN *list*
Stop current procedure and return control to calling procedure	STOP	STOP
Abort procedures and return to top level control	TOPLEVEL	THROW "TOPLEVEL
Run a list conditionally	...	CATCH *name list*
Return from conditional running	...	THROW *name*
Output information about error	...	ERROR

LOGO

Next Month in MICRO

November: Games Feature

- **Number Shuffle on the Atari —** A computer version of the popular game Magic Square.
- **Atari Decorder Ring —** A polyalphabetic substitution for encoding and decoding text.
- **Shootdown —** A low-resolution graphics game for the Apple.
- **Solve the Pagoda Puzzle —** For the PET, Apple, and OSI OS-65D systems.
- **Space Invasion —** Fight aliens in this real-time arcade game for OSI computers.

Department Highlights

Data Sheet
Apple Slices
PET Vet
New Publications
Reviews in Brief
Software and Hardware Catalogs

Plus Graphics

Atari Character Graphics
Introduction to 3-D Graphics
Getting Around the Hi-Res Graphics Page on the Apple
Hi-Res Graphics and Memory Use for the Apple
OSI Programmable Character Generator
Extra Colors for the Atari