NO. 61 JUNE 1983 U.S. Edition: $2.50 International Edition: $3.00

MICRO™

Advancing Computer Knowledge

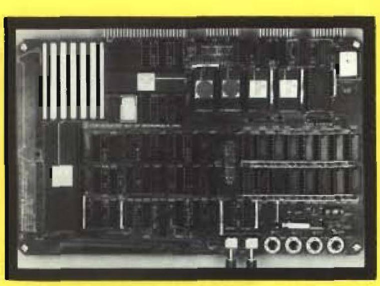
Operating Systems

OS-9
The Best OS for 6809 Microprocessors

PASCAL
Machine Level Read and Write

APPLE DOS
Improve Your Apple With A New OS

Color Disk OS
Restore A Crashed Disk
Recover Your Directory

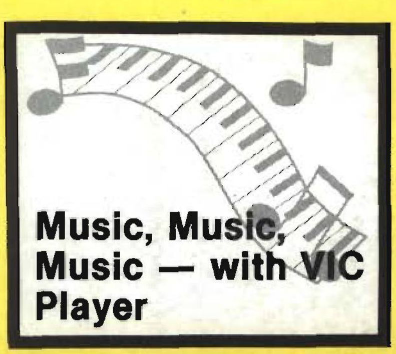


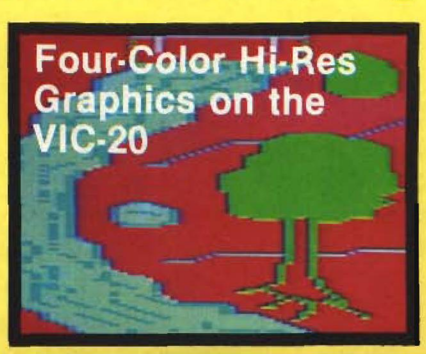
The 68000 Educational Board Reviewed

Redesign Your PET Calculator Keyboard

Atari Program Perfects Calibration

In this month's Learning Center:






BUY A BANANA.
SAVE A BUNCH.
MORE TO COME.
GORILLA · TOUGH · GUARANTEED
Leading Edge Products, Inc., 225 Turnpike Street, Canton, Massachusetts 02021.
Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.

MICRO™

Highlights

Your computer system is only as good as your operating system. New operating systems coming on the market provide the consumer with products that are more powerful and easier to use than most earlier versions. The June issue of MICRO takes a look at some of these operating systems including OS-9 and Motorola's MEX68KECB educational computer board. Our feature section will help you learn about many of these exciting new items and enable you to choose the right options for your particular needs and computer system.

We open our feature section with an article by Phil Daley entitled "Apple Operating Systems" (pg. 20). Phil discusses operating systems that can be used with both unmodified Apples and with those requiring additional hardware. All the systems use standard DOS 3.3 format disks. Steven Lesh, in "U.C.S.D. Directory" (pg. 26) examines the U.C.S.D. directory at the byte and bit level rather than in the usual terms of high-level language data structures. Included is a brute-force method of accessing U.C.S.D. directory blocks.

"OS-9, A Structured Operating System" by Mark Boyd (pg. 32) is a summary of OS-9, one of the most powerful systems for an 8-bit microprocessor. MICRO follows up on Mark's article with "A Unix-Like Operating System for the 6809 Microcomputer, Part 1" by Steve Childress (pg. 46). Steve discusses the "power-per-dollar" of hardware as yet untapped due to manufacturers' fears of software incompatibility. Steve claims OS-9 is a new way to view software architecture that is beneficial to the small-computer user.

William Clements reviews the Color Computer disk system, examines disk sectors, and explains how to repair a crashed directory in "Comments/Utilities on Color Disk BASIC" (pg. 34). And finally, to complete our feature section see Terry Jackson's "A Review of the 68000 Educational Board" (pg. 42), an overview of Motorola's new 68000 educational computer board.

In our applications section we have three articles designed to assist you in problem solving. First, if you are building remote sensors for a personal computer you may encounter trouble calibrating the homebrew sensors. See Mike Dougherty's program "Calibrating by Least Squares Polynomials" (pg. 54), which allows a set of calibration data points to be fitted with a least squares polynomial. "Pinewood Derby with Computer Timing" by Sidney Koegler (pg. 60) eliminates the arguments and frustrations of judging the Boy Scouts' Pinewood Derby! Here is an automated timing and judging program that uses photosensors installed at the finish line. For those of you who feel helpless when converting BASIC decimal results to fractions, Dr. LeRoy Moyer has written a program that automates such calculations in "Fractionated BASIC" (pg. 64).

About the Cover

This month's painting in our computer screen offers a new interpretation of the concept "operating system."

Painting by: Frank Wyman
Wyman Art Studio
Lowell, MA 01852
(617) 459-7819

This month the Learning Center provides you with three programs that educate and entertain. Brian Zupke teaches you how to use the VIC's joystick to draw high-resolution pictures in four different colors. Study "Four-Color Hi-Res Graphics" (pg. 70). Do you dream of becoming a *note-able* musician? Phil Daley's "VIC Player" (pg. 72) is a five-octave keyboard on which you can compose your own tunes or play old favorites while learning music programming. And David Bryson shows you how to construct a lightpen for approximately $15.00 and two hours of work. Read "An Inexpensive Lightpen for the VIC-20, C64, and Atari" (pg. 82).

Our machine-language utilities include the continuation of Joe Hootman's series on "68000 Instructions" (pg. 88) and Randy Hyde's "Parameter Passing, Part 2" (pg. 94). Also included is Bob Sullivan's "HEXPAD: Utility for M/L Key-ins" (pg. 90).

MICRO™

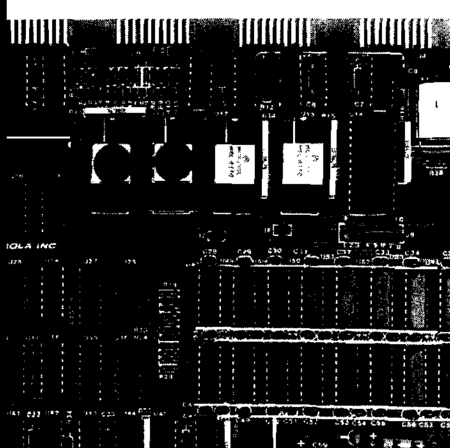
Advancing Computer Knowledge

MICRO
P.O. Box 6502
Amherst, NH 03031
(603) 889-4330

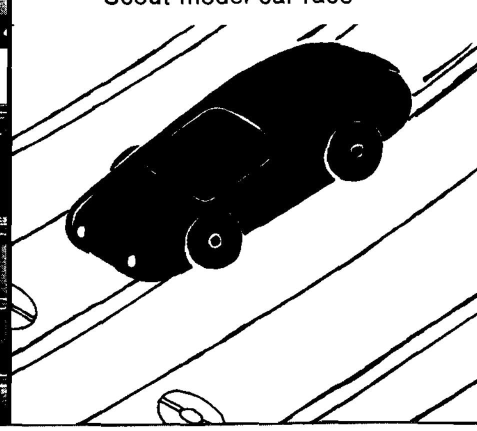
MICRO is published monthly by: MICRO, Amherst, NH 03031. Second Class postage paid at: Amherst, NH 03031 and additional mailing offices. USPS Publication Number: 483470. ISSN: 0271-9002. Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to MICRO, 10 Northern Blvd., P.O. Box 6502, Amherst, NH 03031, or call (603) 889-4330, Telex: 955329 TLX SRVC, 800-227-1617. *Subscription rates (per year):* U.S. $24.00, $42.00 / 2 yr. 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.

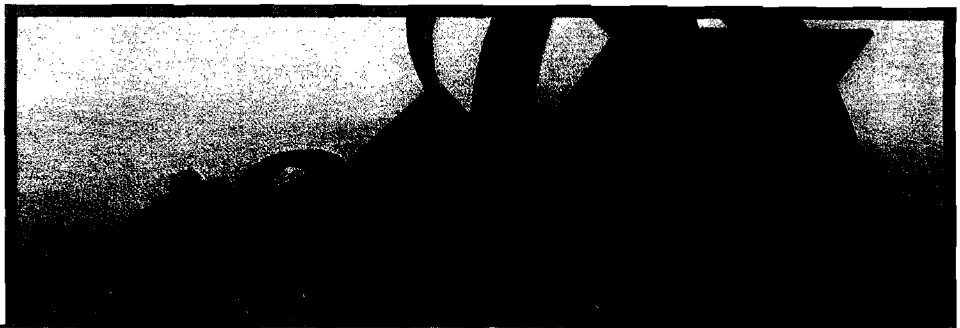
Operating Systems


```
BA00   7F  C1  1  r  1  SysGo
BA7F  193  E1  1  r  8  ACIA
BC12   38  F1  1  r  2  TERM
BC4A  651  C1  1  r  1  IOMAN
C29B  BBC  D1  1  r  1  RBF
CE57  41C  D1  1  r  8  SCF
D273  33A  E1  1  r     DC3
D5AD   2C  F1  1  r     D0
D5D9   2F  F1  1  r  1  H0
D608   36  F1  1  r  6  T1
D63E  216  E1  1  r  1  WD1000
D854   CA  C1  1  r  1  CLOCK
D91E  472  11  1  r  2  Shell
DD90   2E  11  1  r     Load
DDBE   48  11  1  r     Unlink
AC00  1DA  11  1  r  1  Mdir
```



Applications





Operating Systems Feature

286K per 3" disk cartridge!

AFTER YOUR FIRST DRIVE GO WITH AMDISK.

The AMDISK-I Micro-floppy disk system is an engineering breakthrough in disk size, storage capacity, media protection and user convenience. It's fully compatible with your Apple II* and Apple IIe and is furnished with an interconnect cable. Enjoy a full 286K storage† capability and the extra convenience of the new 3" hard plastic encased diskette. Packed 10 per box, the disk cartridge fits into a shirt pocket and is easy to mail.

Suggested Retail Price . . . . $299 (3" cartridge $6.99 ea.)

The Amdisk Single Drive system is ruggedly constructed for years of trouble-free operation and is backed with our 90 day warranty on parts and labor.

Just circle the reader service number for complete specifications.

* Apple II is a registered trademark of Apple Computer, inc.
† Requires recording on both sides

2201 Lively Blvd. • Elk Grove Village, IL 60007
(312) 364-1180 TLX: 25-4786

AMDEK CORP.

Amdek . . . your guide to innovative computing!

MICRO™

Editorial

My favorite computer dictionary offers this definition of an operating system: *1.* An organized collection of techniques and procedures for operating a computer. *2.* A part of a software package (program or routine) defined to simplify housekeeping as input/output procedures, sort-merge generators data-conversion routines, or tests.

Now you ask, "OK, but what *really* is an operating system?" Unfortunately the definition will not get much clearer. If you are technically oriented and know your computer inside as well as out, you don't need a better definition because you already have a good understanding of operating systems. But, if you're still in a fog, read the articles in our feature section beginning on page 20.

Of course, whatever your level of knowledge, we all know that the operating system is crucial to the running of any computer. The reason we have problems using different software on different computers is that the operating systems aren't compatible. The logical solution seems to be to provide a standard operating system that could run on many different computers. Some steps are being taken in this direction.

As you read through this issue you'll notice that operating systems are usually developed for a particular chip or by a company for its computers. The 6502 world, where there is no standardization, provides the most problems; you'll find Commodore BASIC, Atari BASIC, Atari DOS, Apple DOS, and more. For the most part, 6809-based computers are standardized to FLEX or OS-9, and UNIX is becoming a standard for 68000-based machines.

When you get involved with computers based on the Z80, 8086, etc., you'll find that CP/M is supported by virtually all of these computers. Because all the machines use one operating system, more software is available for these machines than those based on other microprocessors. As a result, many of the manufacturers of the new computers have chosen the Z80 or 8086. And, manufacturers of 6502 microprocessors are beginning to plan for CP/M compatibility on their machines. For instance, Commodore has made provisions for the Z80 or 8086 on their C 128, although neither option is yet available. There will be a Z80 cartridge for the Commodore 64, too. A Z80 card is available for the Apple, as well as a 6809 card, which will allow for CP/M, OS-9, and FLEX.

Marjorie J. Morse

Marjorie Morse
Managing Editor

MICRO Moves North!

Our new address is: 10 Northern Blvd., Northwood Executive Park
P.O. Box 6502
Amherst, NH 03031
Our new phone number is: (603) 889-4330

MICRO™

Letterbox

New FCC Ham Radio Proposal

Dear Editor:

Since the April issue of MICRO featured Communication, I thought you should be aware that the FCC is now proposing a new class of amateur radio license requiring NO morse code test. The class will be intended for people whose primary interest is in computers or experimentation rather than the traditional amateur goals of working all states or all countries. The license will probably be restricted in power and in frequency, and will be mainly for short-range high-quality communication such as data links.

The existing ham radio community is vigorously opposing this new class and has organized letter-writing programs to the FCC stating their opposition. I would like to suggest that the readers of MICRO would be among those to benefit from such a new class of license. The FCC should hear from more computer people since the ham radio voice will be well represented opposing the idea.

I believe connecting a radio link to a computer will open a new mode of communication. Comments should be addressed to the FCC referring to docket = 83-28 FCC, Washington DC 20554. Send your comments to MICRO too.

An anonymous radio ham/computerist

Updates and Microbes

Interface Fixes

Four misprints appeared in my article "Building a Parallel Printer Interface" (53:23). In the diagram on page 23, lines A0 and A8 are interchanged. In the same diagram, the STROBE line should *not* be connected to +5. On page 24 there are two errors in the second column. About halfway down, the text should read "...simply use the Q output from U8 rather than Q." Finally, on the next-to-last line of column 2, the STA command should be changed to LDA.

Rolf B. Johannesen
Rockville, MD

Line Correction

There is an error in my article, "A Binary Search Routine" (57:37). Line 10170 in both listing 1 and listing 2 should be:

```
10170 For J=J TO 0 STEP -1
```

instead of

```
10170 For J=J TO 1 STEP -1
```

Alfred J. Bruey
Jackson, MI

Send your comments, insights, ideas, or bugs to Letterbox, MICRO, P.O. Box 6502, Amherst, NH 03031.

Parts List

The following parts list was left out of John Steiner's article, "A Home-Built Communications Interface" (59:44):

PARTS LIST MODULATOR

REF. NUMBER	VALUE	NAME
C1	1 UF 16V	CAPACITOR
C2	.03 UF	"
C3	.03 UF	"
D1, D2, D3	1N914 OR EQUIV	DIODE
IC1	LM566	PLL IC
IC2	LM 339	
Q1, Q2	2N2222	TRANSISTOR
R1	10 K	RESISTOR
R2	1 K	"
R3	1 K	"
R4	10 K	"
R5	10 K	"
R6	1 K	"
R7	10 K	"
R8	5 K	"
R9	2.2 K	"
R10	250	TRIMPOT
R11	1 K	RESISTOR
R12	24 K	"
R13	10 K	"
R14	12 K	"
R15	20 K	"
R16	10 K	"
R17	1 K	"
	ALL RESISTORS ¼ WATT	
SW1, SW2	SPDT MINI TOGGLE SWITCH	

PARTS LIST REGULATOR

REF. NUMBER	VALUE	NAME
C1	.1 UF	CAPACITOR
C2	1 UF 16V	"
IC1	LM317	ADJ REG IC
R1	220 OHM ½ WATT	RESISTOR
R2	5 K	TRIMPOT

MICRO

SAVE 20%!!!! Subscribe to MICRO™

Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAVE $6.00 Off Newsstand Price!

MICRO™ SAVE 20%

☐ Yes! Enter my subscription to MICRO, and send me the next 12 issues for just **$24.00.**
I save **$6.00 off** the newsstand price!

Name ______________________
Address ______________________
City ______________________
State ______________ Zip __________
I'm paying by ☐ Check ☐ MO ☐ VISA ☐ MasterCard
Card # ______________ Exp.Date __________
Signature ______________________

I OWN A:
☐ Commodore 64 ☐ VIC-20
☐ Apple ☐ PET
☐ Atari 400 ☐ Atari 800
☐ Other: __________

For Faster Service Call:
1-800-345-8112
(In PA: 1-800-662-2444
VISA or MasterCard Only

A Feast Of Computing Ideas... Order These Books From MICRO™

MICRO™ Books

NEW for VIC-20 Users!
Mastering Your VIC-20 With 8 BASIC Projects

A book that makes learning to program your VIC-20 fun! Contains 8 projects and programs. Games, utilities — even a VIC-20 version of "VisiCalc." All 8 programs on cassette to help you learn faster.

☐ Mastering the VIC-20 @ $23.95

NEW for OSI Users!
MICRO on the OSI

Includes Machine-Language enhancements and BASIC Aids, hardware modifications for enhanced/reversed video, programs for control code and upper/lower case entry. A valuable programming tool.

☐ Micro on the OSI @ $19.95

Best Sellers for APPLE Users!
MICRO on the APPLE

Programming aids, utilities, games, enhancements. Together Volumes 1, 2, and 3 contain over 100 programs on diskette. Fully documented and explained.

☐ 3-Volume Gift-Boxed @ $59.95
☐ Vol.1 ☐ Vol.2 ☐ Vol.3 **$24.95 ea.**

Please rush the MICRO Books I've checked above to:
Name ______________________
Address ______________________
City ______________ State ______ Zip ______
(Allow 6-8 weeks for delivery)

I'm paying by: ☐ Check ☐ MO ☐ VISA ☐ MC
Total Enclosed: $ __________
(Add $2.00 s/h per book. MA res. add 5% tax)
Visa/MC # __________
Exp. Date: __________

MICRO™ Reader Service... For More Information On The Advertisers In This Issue!

MICRO™ Reader Service Card

The numbers below correspond to the numbers imprinted on the advertisements in this issue of MICRO. This card valid for 90 days only.

Mail information to:
Name ______________________
Address ______________________
City ______________ State ______ Zip ______

1	2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77	78
79	80	81	82	83	84	85	86	87	88	89	90	91
92	93	94	95	96	97	98	99	100	101	102	103	104
											105	106

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

SAVE 20%!!!!
Subscribe to MICRO™
Use This Postage Paid Card to Order the Next 12 Issues of MICRO and SAVE $6.00 Off Newsstand Price!

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

A Feast Of Computing Ideas...
Order These Books From MICRO™

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 20, AMHERST, NH

POSTAGE WILL BE PAID BY ADDRESSEE

MICRO™

10 Northern Blvd.
P.O. Box 6502
Amherst, NH 03031

MICRO™ Reader Service...
For More Information On The Advertisers In This Issue!

PET Vet

Loren Wright

When you start to think about serious applications for your Commodore 64, a word processor should be one of the first to come to mind. Word processors work well with the C64. The extra memory allows more text to be held in memory at one time, and the keyboard allows rapid and accurate entry of text. The 40-column display is not as much of a problem as you would think. Because of interference of certain colors on some TVs or monitors, being able to change character and screen colors is important.

I have reviewed two very good word processors — *Script 64* and *Word Pro 3 Plus/64*. There are others; they will be reviewed as time and space permit.

Script 64

Script 64 is a new word-processing program for the Commodore 64 by David Foster of Richvale Telecommunications (10610 Bayview Avenue, Richmond Hill, Ontario L4C 3N8, Canada). Richvale's products, including *Script 64* and the *C64-Link* cartridge reviewed in April, are now marketed in the U.S. by Computer Marketing Services (300 W. Marlton Pike, Cherry Hill, NJ 08002). The disk version sells for $139.95.

Design

Unlike other word processors that use either an actual-page or continuous-scroll organization, *Script 64* organizes text in 22-line screens. At first this seemed inconvenient, but the more I used the program, the more I could see advantages. Advancing to the next numbered screen is accomplished easily by pressing the 'F1' key. If you advance to the next screen well before you reach the end, then you avoid a lot of problems. This leaves room to insert text later in the editing process. If you keep only a paragraph or two on a screen, then cut-and-paste operations are easy. On printout, incomplete screens are handled routinely by ignoring the blank lines at the end of a screen. If you do have to fill a screen and continue immediately on the next one, then that can be accomplished too.

The disadvantage of this screen organization is that you always have to keep one eye on the screen to make sure you aren't about to fill it. There is an audible warning when you have filled the screen, but it comes too late to leave room conveniently. In these cases, it is possible to join the screen to the next one.

The disk holds 999 screens, of which 40 can be held in the C64's memory at once. When a load or save operation is performed for a particular screen, then the entire file, including that screen, is copied. This is a slow process with the serial disk drive, so there is a provision to save only half at a time.

Each disk has an index where you keep track of the locations of documents. You can load the file containing a document simply by hitting RETURN with the cursor at the appropriate index entry. Unfortunately, maintaining the index is entirely up to you. This is the only major deficiency in *Script 64*.

Entry of Text

The normal mode of text entry is called 'wordstream.' That is, everything typed in appears continuously on the screen. At the end of a line, the next character appears at the beginning of the next line, even if it is in the middle of a word. To indicate the end of a paragraph you just hit the RETURN key and begin the next paragraph on the next line.

Superscripts, subscripts, and underlined, bold, and italic printing are indicated by special characters within the text. All other formatting controls are handled by the control maps and on special 'structured' screens.

The large control map applies to the whole document and defines such things as the left margin, line length, line spacing, top and bottom borders, page length, page numbering, and placement of a title. *Script 64* has a default control map that is adequate for many applications, but you may change the parameters. You can also save a control map with these altered parameters.

In addition, each screen has its own small control map that overrides the large control map settings for left margin, line length, line spacing, justification, and a few other parameters. A small control map affects only its associated screen.

Structured screens are used for a number of special situations including tables, centered text, and columns of figures. Each line, up to 80 characters, that you want in the final output must end with a return character. There is, however, an automatic return feature that keeps whole words together and terminates the lines properly. Tabs may be set over the full 80 columns. Also available are a bell that sounds five characters before the end of each line, automatic tabulation, and automatic structured formatting.

Editing

Editing with *Script 64* is convenient. The cursor-control keys allow type-over editing anywhere on the screen. Deletions may be done on individual characters, words, sentences or lines, or on a block of text. You may also specify how you want character deletions to work. The default, or typing, delete removes characters to the left of the cursor, closing the space with characters from the right. The editing delete removes the character under the cursor, closing the space with characters from the right. Insertions are by individual character or line, or you can enter insert mode, where all characters typed are continuously inserted at the cursor position. Entering and exiting insert mode is more cumbersome than with WordPro.

It is easy to transfer or copy blocks of text from one position to another, even between widely separated screens. The block is defined using the same procedure as for block-deletes.

Whole screens may be deleted or moved, and blank ones may be inserted.

Search and Replace

The search and search-and-replace functions operate either locally (from current screen to the end of the 40-screen file in memory) or globally (from current screen to the end of the disk). You may also specify that the program pause to give you selective control over each replacement, or you may have it make replacements on all occurrences automatically.

Perhaps the handiest feature of *Script 64* is its ability to specify several search or search-and-replace operations on the same pass through the text. There is no wild-card character match or ignore-case option.

Printouts

The printout is controlled by the control map. To print a file you must specify the sequence of screens to be printed when you first enter print mode. When you enter '000' the output begins. You may output to the screen first to be sure everything appears as you intended. The '4' and '6' keys move the screen left and right as a window on the output, which usually is wider than the 40-column screen. You can begin printing from the next page during this output to video by pressing 'P'. Other print options available include automatic hyphenation, multiple copies, pauses, and printing of the screen numbers. A frequently used or complicated print sequence may be saved on a blank screen. Then, at printout time only this one screen needs to be listed in the print sequence.

Form Letters and Variable Data

Script 64 has a variable chart that allows you to define up to 20 different words or phrases of 30 or fewer characters. Whenever you want to use one of these, you enter an asterisk and the number of the variable. The word or phrase corresponding to the variable number will be inserted at that position at printout time.

These asterisk codes are also used to fill a document from fields of a sequential or CBM Scratchpad file. (CBM Scratchpad is a separate database program sold by Richvale Telecommunications.) Because the fill points are numbered in the document, you can reuse or skip fields or use them out of order.

There is no way to use *Script 64* itself to construct a list of names and [illegible] fer, as with WordPro. However, it is fairly easy to write a BASIC program to construct sequential files for this purpose. Also there is no provision for manual fill points within a document.

Manual

The manual is well written and easy to follow, particularly when you are learning the program. As a reference, however, the manual is not as good. There is no index and no summary chart, although there are three help screens available on the disk. Also there is a summary table at the end of each chapter. It comes in a three-ring binder so that it will lie flat on a table.

Equipment Compatibility

The initial menu lists eight different printer types, including ASCII. I tried *Script 64* with a CBM 4022 and with a Star Micronics Gemini-10 interfaced through a Micro World Electronics MW-302. Both boldface (double printing) and underlining worked on both, but subscripts, superscripts, and italics worked on neither. Using the redefinable key feature, these options can be supported if they are available on your printer.

Script 64 also will support either IEEE or serial disk drives and parallel, serial, or IEEE printers.

Special Features

Script 64 provides for entry of special French characters including grave, circumflex, aigu, and cedille marks. These appear correctly on the screen and on the printout, if the printer can handle it.

The dictionary mode is an interesting and potentially useful feature. It allows you to build up a dictionary disk of 20,000 words. You then com-

(Continued on next page)

PET VET *(continued)*

pare your text to the dictionary and it indicates words not found. You may add a word to the dictionary, replace it, skip it, or quit. The dictionary does not actually correct your spelling.

Additional features include 'help' screens, redefinition of keys, screen and character color change, and a numeric mode that lines up decimal points automatically.

The cassette version of this program was not tested and, although it is mentioned in the manual, it is not being sold by Computer Marketing. The index, help screens, dictionary, and global search features are not supported in the cassette version.

Text prepared using *Script 64* is compatible, to varying degrees, with other word processors sold by Richvale.

WordPro 3 Plus/64

WordPro 3 Plus/64 is a word processor program for the Commodore 64 available on disk from Professional Software (51 Fremont Street, Needham, MA 02194) and dealers for $89.95.

Design

Text in *WordPro 3 Plus* is a continuous scroll of 40-character lines. Up to 352 lines may be in memory at the same time. These are divided between *main* and *extra* text at start-up. Normally you would use 329 lines in the main text area and 23 lines in the extra text area, but the division is flexible. Extra text is used to hold frequently used phrases or stock paragraphs for convenient recall. Also it can be used to display the disk directory or to hold lists of names and addresses for insertion into a form letter.

Formatting, including margins, line spacing, centering, justification, and a great number of other parameters, is specified in special format lines. These lines begin with a check mark and do not print. The parameters may be changed at any point in the text simply by inserting a new format line.

Files are saved on disk by name, and they may be recalled conveniently from a directory listing or with an abbreviation.

Entry of Text

Text is entered in a continous 'wordstream' style. There is no division into paragraphs or screens; this occurs at printout time. When the cursor reaches the end of a screen line, it simply moves to the beginning of the next line, even if it means splitting a word in the middle. To force a new line, such as at the end of a paragraph, you just hit the RETURN key.

Another file on disk may be linked to the end of the current one by including the name of the next file in a special format line.

Superscripts, subscripts, and bold or underlined text are indicated with special control characters. Special characters, such as those found on certain letter-quality printers' print wheels, also can be defined and included.

Editing

The cursor-control keys can be used to position the cursor anywhere in the current file. Changes may be made by typing over, inserting, and deleting, etc. Individual characters, as well as words, sentences, and screen lines may be deleted. Individual characters may be inserted, and there is a convenient insert mode where all typed characters are inserted at the current cursor position.

There is a *range* function used to memorize any sequence of screen lines. Once defined, these lines may be erased, transferred, or copied. This is not as convenient as a true *block* type function where the beginning and ending points may be specified anywhere within a line.

Also you can insert a whole file from the disk at any point, assuming it will fit.

Search and Replace

There are separate search and search-and-replace functions, which can operate either locally or globally.

Global operations act both on the current file and upon subsequent linked files stored on the disk. There are two separate command codes, plus two more to restart the searches. Replacement is done automatically on all occurrences, without any selective option. The '?' character is used as a wildcard to match any character, but there is no ignore-case option.

Printouts

Printouts are done according to the codes contained in the format lines you enter in the text. Before printing, you may specify whether or not to pause between pages, where to get data to fill variable blocks, and the number of copies. There is also a global option where you specify the name of the first file in a sequence. Linked files are loaded automatically as they are required.

Form Letters and Variable Data

Fill points may be indicated within your text. These will be filled before printout either manually, with data you type in, or automatically, from extra memory or a sequential file. There is no differentiation among fill points in a document, so you must be sure your extra memory or sequential file contains the right items in the proper order.

Frequently used phrases and paragraphs may be defined in extra memory. A few keystrokes, including the appropriate abbreviation, will copy that phrase or paragraph at the current cursor position.

Equipment Compatibility

WordPro is especially designed to operate with the NEC Spinwriter series of printers, so it supports all the features of that line, including special characters on its interchangeable print wheels. Other printers, including Diablo, Qume, TEC, and various Commodore printers are supported too.

This evaluation was conducted using a CBM 4022 printer and a Star Micronics Gemini-10 printer through a Micro World Electronics [13333 S. Wadsworth Blvd. C105, Lakewood, CO 80227] interface. Overprinting was not supported on either printer using the standard codes, and underlining was done with individual backspaces for each letter. Subscripts and superscripts could not be printed using the standard codes, but *WordPro's* definable characters can be used to produce these and other special effects on the Gemini-10.

Manual

The manual is well organized with an index, table of contents, and reference charts. There is also a series of lessons, and the disk includes a linked letter with examples of most of the format commands.

The soft-bound manual is difficult to keep open on a table, but it is punched so you can put it in your own binder.

Special Features

The screen, character, and border colors may be changed. This is required to counteract interference and peculiarities of some monitors and TVs.

There are tabs, numeric mode, column add/subtract, and hyphenation.

The disk is copy-protected but includes a back-up copy.

WordPro, in its many forms, is the most popular word processor for Commodore equipment. Files are compatible with certain size restrictions between the different versions.

Next Month

In my July column I'll take a look at FORTH as a development tool for commercial software on the Commodore 64. Also, I'll review some new books for the VIC.

You may contact the author at P.O. Box 6502, Amherst, NH 03031.

MICRO

MICRO™

From Here To Atari

Many companies supply products that can be attached to the Atari 400 and almost as many supply products for the Atari 800. Starting with this column, I will include a description of one of these products each month.

InHome B Key Keyboard

I recently installed an InHome keyboard in a customer's Atari 400 computer. This keyboard replaces the membrane keyboard inside the case. The membrane keyboard is smaller than a standard keyboard but the InHome keys are placed at the standard distances from each other. In order to do this and still fit the new keyboard into the same space, a few of the keys at the ends of the rows were relocated to places on each side of the spacebar.

Specifically, there are five relocated keys: CNTL, TAB, CAPS/LOWR, BACK S, and ESC. A period of adjustment is required for touch typists who are very familiar with the layout of the Atari 400 and Atari 800 computer keyboards. The most annoying part of the adjustment is the relocation of the BACK S key. The BREAK key on the InHome keyboard occupies the spot formerly occupied by BACK S. I feel that this point is the keyboard's major weakness; the BREAK and BACK S keys should be reversed.

Once you get past the adjustment period, the keyboard has obvious advantages. All of the keys, including SYSTEM RESET and the three function keys, are full stroke keys on this board. In addition, increasing the distance between the key centers to the standard distance of typewriters makes word processing and other text-oriented tasks even easier. All except one of the keys are labelled almost identically to the keys on the Atari 400 computer keyboard. The only one completely changed is the Atari key, which is now the InHome key.

Adding an InHome keyboard and a 48K memory board to an Atari 400 makes it almost equivalent to an Atari 800 at a savings of up to $200. You still don't have a right cartridge slot, but that is starting to look like a vestigial organ of Atari 800 computers. Also missing is the circuit for attaching a monitor. For more information write to InHome Software Inc., 2485 Dunwin Dr., Mississauga, Ontario, Canada L5L 1T1.

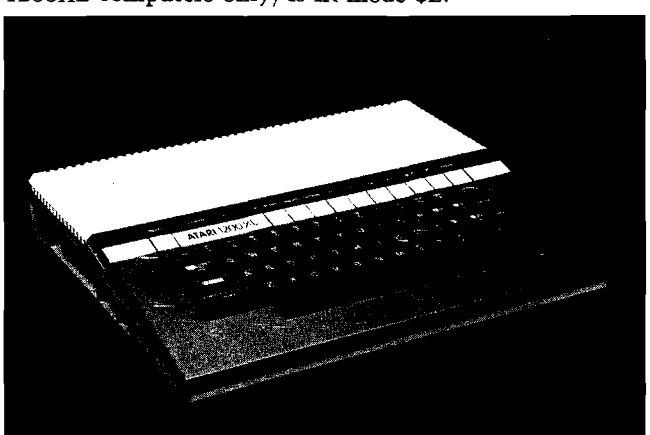
by Paul Swanson

On Map Modes

The new Atari 1200XL computer has two new map modes. One of them is mode 15 which is a four-color mode with 160 dots horizontally and 192 vertically. Trying a GRAPHICS 15 statement on an Atari 400 computer or an Atari 800 computer will produce an error 145 because the operating system does not support that mode. It is available on both of those computers, but you must supply your own display list or alter one supplied by the operating system.

Listing 1 alters the display list that results from a GRAPHICS 8 statement. The FOR/NEXT loop in lines 40 through 70 performs this alteration. Line 30 gets the location of the start of the display list first, then the FOR/NEXT loop investigates the list and makes the changes. The specific differences are two types of commands. A decimal 79 ($4F) loads the memory scan counter for the mode 8 screen. This gets changed to a decimal 78 ($4E) for a mode 15 screen. Decimal 15 ($0F) is a mode line command that displays one mode 8 line on the screen. The decimal 14 ($0E) causes a display of a mode 15 line instead.

The numbering of the commands and the modes can be a little confusing. There are two distinct numbering systems used. The modes declared in GRAPHICS statements are OS (operating system) modes and obey a numbering system quite different from the internal numbering system, referred to as the IR (internal register) mode. OS mode 8 is IR mode $F and OS mode 15 (Atari 1200XL computers only) is IR mode $E.



ATARI 1200XL HOME COMPUTER™

The FOR/NEXT loop occupying lines 80 through 150 draws three diagonal bars of color on the screen. The BASIC ? (PRINT) statement to a map mode screen is not documented at all as far as I have been able to determine. It interprets one screen dot per character in the string. If the screen is a two-color screen, as in this example, only the last bit is used in each character. Therefore, if the ATASCII value of the character is an odd number, the foreground color is used for the dot and the even numbers produce background colored dots. The ?#6 method is used in this example because DRAWTO cannot be restricted to only even numbered or only odd numbered columns.

The requirement for the even or odd numbered columns is due to the operating system "thinking" that it is maintaining a mode 8 screen. Altering the display list causes the screen image to be interpreted in bit pairs. Beginning on an even row, bit pairs of 01 (binary) result in a dot of the color in register 0, 10 to register 1, and 11 to register 2. Note that this follows the same scheme used in the BASIC COLOR statement numbering.

Listing 2 produces the same display on an Atari 1200XL computer as does Listing 1. Since the operating system "knows" that it is a four-color screen, the numbers 1, 2, and 3 are used instead of the pairs 01, 10, and 11. On four-color screens, each character in the string contributes the last two bits of its ATASCII value.

Listing 3 demonstrates the increased resolution between mode 7 and mode 15. This listing (and Listing 1) will work on the Atari 400 computer and the Atari 800 computer as well as on the Atari 1200XL computer. Lines 10 through 70 draw the three colored bars on a mode 7 screen. Lines 80 through 100 wait for the RETURN key. Lines 110 through 140 convert the display list to mode 15. The screen will shrink to one half height because mode 7 has only half the number of lines as a full mode 15 screen.

POKEY Times

I recently received a letter from Ian Chadwick, the Associate Editor of InfoAge and author of *Mapping the Atari*. He suggested that I include some information on the POKEY timers in my column. His observation that the Atari documentation does not adequately cover them has some merit.

There are three of these timers available for use. All of the times use the AUDF values for initialization, which are the same values used for sound channels 0 through 2. STIMR is another location actually used to start the counters. Each timer sets an interrupt when it counts down to zero.

A couple of peculiarities about the timers reveal some interesting insights into how they operate. They are enabled by setting the corresponding bit at location 16 (decimal) to one. Set timer 1 by POKE 16,193. Note that the keyboard, including the BREAK key, no longer functions. Press SYSTEM RESET to restore the keyboard. The reset puts the value 192 (decimal) back in location 16, disabling the interrupt.

POKEing 193 into location 16 enabled timer 1 when the AUDF value was zero. This causes a constant interrupt, leaving no processor time available and masking all other maskable interrupts. SYSTEM RESET is a non-maskable interrupt that can override the timer interrupt.

Now POKE 53760,10, which sets AUDF1 to 10, giving timer 1 some time when it is counting before it generates the interrupt. POKE 16,193 to enable the timer and hold down the space bar. Notice that the auto-repeat is irregular and slower and that the keyboard click changed its tune. If you POKE a number smaller than 10 into AUDF1, the auto-repeat will be even slower, the limit being a zero in AUDF1, which stops everything.

Although the examples are BASIC, the BASIC language cannot really function with these timers because interrupt routines cannot be written in BASIC. In machine language, an interrupt routine can handle the timer 1 interrupt very effectively. One of the best uses for the timer interrupts is to time two external events like pulses on the controller jack pins. Set AUDF1 to the interval you want to use to time the event first. Alter $0210 and $0211, which is the interrupt vector. Store your interrupt routine starting address here. Set up a flag and a counter in memory and set the flag to zero. Last, enable the interrupt.

The interrupt routine is in three sections, controlled by the value in the flag. When it is zero, just poll the first signal. If it is present, set the flag to one, store anything in location $D209 (this is STIMR — writing to this location initializes all timers to the AUDF value). Then zero your counter, PLA and RTI. Make sure that you pull A before any RTI and always restore X and Y and any other registers you affect or you will probably crash the system. If the flag is one, increment the counter and test the second signal. If it is present, set the flag to 2 before you restore the registers and RTI. If the flag is 2, just restore registers and RTI.

That sequence will time the duration between the two events polled in the units determined by the value in AUDF1. To start the sequence, just store a zero in the flag. Check for the flag equal to 2 for a completed timing sequence. When the flag equals 2, the counter value is valid. Note that the units are N/63921 seconds, where N is the AUDF1 value, so using 64 for AUDF1 produces units reasonably close to milliseconds. Altering AUDCTL can change the frequency used if this is an inconvenient unit — the default 64 KHz can be altered to about 15 KHz or about 1.79 MHz. The exact frequencies are 15.6999 KHz and 1.78979 MHz for these alternates.

Next Month

The 850 interface seems to be an interesting yet misunderstood device. July's column will clear up questions you may have. Next month's hardware product description will be of the new 80-column RGB interface recently announced by Austin Franklin Associates.

MICRO™

Send your letters to: Paul Swanson, 97 Jackson St., Cambridge MA 02140.

CoCo Bits

by John Steiner

Last month I commented about accessories that connect to the RF output of the CoCo and provide video output for a monitor. I was not precisely correct: the monitor adapter hooks into the input of the modulator. The unit from Computerware requires removal of the 1372 video IC, which is installed in a socket that comes with the kit. They are reinstalled as a package in the 1372 socket. Three wires leave this assembly to provide power and video to the preamplifier circuit board. The board is postage-stamp size and can be attached to the RF modulator case with the double-stick foam tape provided. A single clip is attached to the audio input line of the modulator, and a jumper is provided that must be cut if you are using a color monitor. Two cables exit the assembly, allowing separate audio and video signals to be available.

Needless to say, I had to justify the purchase of this board, so I bought a green screen monitor. The conversion was worth the expense as the crispness and clarity compared to a television are amazing. The RF output is available in addition to the video signal, which is a real advantage for demonstration purposes. There is a disadvantage in that selecting black and white mode kills color at the RF output, while selecting color puts a fine cross-hatch pattern on the monitor, making it difficult to read. Interestingly, high-resolution color graphics are still available in B/W mode even though there is no colorburst signal present.

Recently I had a long chat with Bob Rosen of Spectrum Projects. While talking with Bob about using the Color Computer with a TV, the subject of radio frequency interference (RFI) came up. If you own a CoCo disk system, probably you are already aware of interference caused by the drive cable. Repositioning and coiling the video cable have been my only remedies for the problem, and yet the interference still persisted. In addition, selecting the 64K RAM mode increased the interference to a point where it was extremely annoying. The problem was one of the major reasons I wanted to use a monitor, which isn't affected by RFI.

Bob suggested a solution to the problem: replace the standard audio-type cable Radio Shack provides to connect CoCo to the monitor with a higher-quality 75-ohm video cable with phono plugs installed. The cable can be made easily using RG-59 Coax and two RCA-style phono plugs; or many video specialty stores have them readily available. I have a video dubbing cable I purchased for my video tape recorder that contains video and audio lines. Replacing the Radio Shack cable significantly decreased the interference, though it was not eliminated entirely. My next step will be to coil the cable through a 1-inch toroid coil.

While on the subject of video interference, removing the TV/computer switch, running the cable directly to the coax inputs or through an adapter to the 300-ohm VHF terminals is advantageous. Don't try to ground the cable or shield to the TV set chassis ground, as I have heard some people suggest. Connecting to chassis on many TVs may be unhealthy, not only to your computer equipment, but to yourself. Some TV chassis are tied directly to the AC line terminals, and connecting the TV plug backwards would be an unforgetable experience

If you do much work with machine-language files, probably you have wished that you could log the start, end, and execute addresses of your files. Ken Christiansen provides a short utility that will provide you with that pertinent information from either a disk or tape machine-language file. To use the routine, first load (or CLOAD) the program in listing 1, then load your machine-language file. Once loaded, type RUN. The screen will clear and provide both decimal and hex values for the file. There are a few limitations: first, HEX$ is available only on Extended Color BASIC CoCos. Secondly, auto executing programs and programs that must occupy workspace required for listing 1 must be loaded with an offset. For example, CLOADM "filename", 2000 will load the program 2000 bytes higher than it normally resides.

Last month I promised to comment further on FLEX as I get more accustomed to working with it. For those who may not be familiar with using a DOS, commands may be memory-resident or disk-resident. If you specify a command, the DOS first looks in memory to see if the command routine is stored there; if not, it turns on the drive and searches the disk. This allows the flexibility of writing your own commands, which can be added to the disk. If a command is not on the disk, a "NOT FOUND" response is printed.

Most commands can be given with files or operation data specified after the command. For example, "LIST, <filename>" will list the textfile called filename to the CRT or terminal. "DATE,month,day,year" will install a new date. In all, there are over 50 commands, files, and utilities included in the package. The EXEC command has the ability to execute FLEX commands stored in a text file. For example, initializing a new disk requires at least three separate commands. These commands can be stored in a textfile and implemented by typing "EXEC, filename." Each command in the file will be read and executed. You can use the BUILD command to build a textfile that contains the desired commands. BUILD is not a text editor but allows entry of single lines of text.

FLEX files can be individually protected, unlike R/S DOS, by using the PROT command. Files can be write-, delete-, or catalog-protected. A write-protect automatically delete-protects as well. The catalog-protect prevents the file from being displayed during a CATALOG command.

LISTING 1

```
10 CLS : P=PEEK(487)*256 + PEEK
   (487)*256 + PEEK(488) : PRINT
   "START DEC";P; : PRINT "HEX";
   HEX$(P) : 'START
20 P=PEEK(126)*256 + PEEK
   (127)-1 : PRINT "END  DEC";P;
   : PRINT "HEX";HEX$(P) : 'END
30 P=PEEK(157)*256 + PEEK(158) :
   PRINT "EXEC  DEC";P; : PRINT
   "HEX";HEX$(P) : 'EXEC
```

You may contact Mr. Steiner at 508 Fourth Ave. NW, Riverside, ND 58078.

MICRO™

APPLE II Operating Systems

by Phil Daley

The operating systems that I have seen available for the Apple use essentially the same read/write routines, but they have been modified or changed. All the disks created by any one system can be read by any other system if you know the procedure. Modifying disks to make them copy-protected (unreadable) is a different technique and a different topic. The Operating Systems covered in this article all use standard DOS 3.3 format disks. The list is as follows: DOS 3.3®, Pascal®, ORCA/M®, Flex®, OS-9® and CP/M®. There are also many varieties and colors of patches and fixes for DOS 3.3 on the market, ranging from Craig Peterson's and the Floeters' articles in *Nibble* to 'DIVERSI DOS' and 'MASTER DOS'.

Except for ORCA, all of the systems have a special {format} command. Whether a user types 'INIT', 'NEWDISK' or 'FORMAT' the results are a disk that is DOS 3.3 compatible as far as the individual sectors are concerned. The difference is in the boot program installed (or not installed) on the disk, and the directory and other housekeeping type sectors (for instance: the VTOC) written to various tracks on the disk. Having talked to some of the individuals responsible for converting operating systems for other microcomputers to the Apple, I discovered that the lowest common denominator and the reason for this compatability at the low level is the DISK II Apple disk drive and controller card, which impose certain hardware limitations on the software involved.

DOS 3.3 from Apple Computer Co.

Other than the CONTROL-D kludge, the standard operating system is highly efficient, moderately user friendly, and extremely error free. Other than the misadventures I had with a Corvus Systems hard disk running DOS 3.3, I have crashed a disk only two or three times. Apparently the people at Corvus do their field testing on the first group of unsuspecting customers that come along, with predictable results. I was able to resurrect most of the files on my crashed disks using a disk zap program.

Due to the lack of disk commands in the BASIC ROM, it is necessary for DOS to intercept all of the commands and check to see if any particular one is a disk command. This is facilitated (for DOS, not for us) by placing a CHR$(4) at the beginnng of every disk command. It is also wise policy to place a CHR$(13) immediately preceding the CONTROL-D to insure that DOS sees it. That is the 'basic' problem with Apple DOS. I can't remember how many programs I've written that didn't work the first time because of the lack of a carriage return.

The only file-handling capability that I find missing is a 'LIST' function for textfiles. It would be convenient not to have to run a file reading program to see what is in a particular file.

The technique of a track/sector list of program sectors seems to waste disk space at first glance. (If you want to talk about waste, consider RS DOS for the Color Computer. The minimum amount of information that can be read or written is a granule — 2,304 bytes of information: one half of a track.) The extra space required for short programs is more than compensated for in quick disk access using random records on large files. DOS is able to calculate the exact track and sector that any particular random-access record resides on, and immediately seek that sector. This is a tremendous improvement over the sector-linking so prevalent in other systems.

The best consideration for using Apple DOS is that most of the commands are loaded into RAM and stored there while the computer is left on. This means that disk access to a system disk is kept to the absolute minimum and a program rarely requires a specific system program to reside on the program disk (for example, the 'CHAIN' program). In addition, a single drive system is a practical reality. This is not true of the other systems available.

There are also many different patches to the standard DOS on the market today. If one fits a specific

(Continued on page 23)

ARK COMPUTING

P.O Box 2025
CORONA, CA 91720

(714) 735-2250

ARK'S APPLE Software!

WORD PROCESSORS	List	ARK
Magic Window	99.95	69.95
Magic Window II	149.95	99.95
Bank Street Writer	69.95	64.95
Pie Writer	149.95	109.95
Easywriter	99.95	69.95
Pro Easywriter	175.00	119.95
Screenwriter II	129.95	79.95
Word Handler	199.00	134.95

BOOKS	List	ARK
Apple Graphics Arcade Book	19.95	15.95
Book of Apple Software	19.95	15.95
Elem. Apple	14.95	11.95
Beneath Apple DOS	19.95	14.95
Kids & the Apple	19.95	14.95
*Using 6502 Assem. Lang.	19.95	11.95

UTILITIES	LIST	ARK
Anix	49.95	39.95
Apple Doc	39.95	29.95
Applewriter Preboot	19.00	15.00
Bag of Tricks	39.95	29.95
The Dictionary	99.95	69.95
Disk Recovery	30.00	21.95
Transend II	149.00	119.95
Dosource	39.95	24.95
E P F IV	79.95	49.95
Lazer Pascal	39.95	29.95
Graforth	75.00	54.95
ALDS	125.00	89.95
Multi Disk Catalog III	24.95	19.95
Sensible Speller	125.00	89.95
Super Disk Copy III	30.00	21.95
TASC	175.00	129.95
UCSD P-System Software Set	635.00	450.00
Graphics Magician	59.95	49.95
Visicalc Preboot	49.95	39.95
Locksmith	99.95	69.95
Inspector (Disk)	59.95	49.95
Watson (Disk)	49.95	39.95
*Omega Pack	209.85	145.00

(Locksmith, Inspector, Watson)

Beagle Bros.	List	ARK
Alpha Plot	39.50	29.95
Apple Mechanic	29.95	21.95
Beagle Bag	29.50	21.95
DOS Boss	24.00	15.95
Flex Text	29.50	21.95
Frame up	29.50	21.95
Tip Disk #1	20.00	14.95
Typefaces	20.00	14.95
Utility City	29.50	21.95
Pronto DOS	29.95	21.95

Peripherals	List	ARK
*ALS CP/M Card	399.00	334.95
CPS Multi-function Card	239.00	149.95
Guardian Angel	595.00	499.95
Kraft Joystick	64.95	49.95
The Mill Pascal Speed-up kits	369.00	289.95
Micromodem II	379.00	264.95
Micromodem w/Term Pack	409.00	309.95
Microsoft Premium Pack	685.00	494.95
Microsoft 16K Ram Card	99.95	69.95
Parallel Interface	139.50	64.95
Microbuffer II	295.00	219.95
RAM Plus 16K	189.00	139.95
STB 128K Memory Card	499.00	359.95
Super Fan II	79.95	59.95
Super Switcher	295.00	219.95
*212 Smartcat 1200 Baud	595.00	499.95
Versa Card	199.00	149.95
Graphics + Plus	159.95	99.95
Z80 Softcard	345.00	224.95

LISA

In celebration of the arrival of the new Apple Lisa Computer to **ARK'S** *office, we are having a* **SPECIAL** on the **APPLE II, II+** and **//e** programs of the same name...

LISA	49.95	79.95
LISA ED SYS.	74.95	119.95

Don't Miss The Boat . . . Upgrade your Apple II & II+ to an APPLE //e!

	List	ARK
Lazer Lower Case + Plus	64.95	29.95
Lazer Keyboard + Plus	99.95	49.95

These two enhancements working *together* allow upper/lower case with shift key, full 128 character keyboard and a 64 character type ahead buffer which allows faster data entry.

To further upgrade your Apple II System, buy both the Lower Case Plus and the Keyboard + Plus then choose on of the following outrageous offers!

16K Ramcard	**49.95**
WIZ-80	**144.95**
Omnivision	**99.95**
Character Set + Plus	**5.00**

SPECIALS	List	ARK
Axlon RAM Disk 320	1395.00	**1099.00**
Wizard 16K BPO	179.00	**119.95**
S.A.M.	124.95	**84.95**
Videx Combo	375.00	**235.00**

Disk Drives at BLOWOUT PRICES!!!

High quality "Slim-Line" disk drives produced by TEAC

Single Drive	259.95
Single Drive w/Controller	299.95
2 Drives w/Controller	549.95

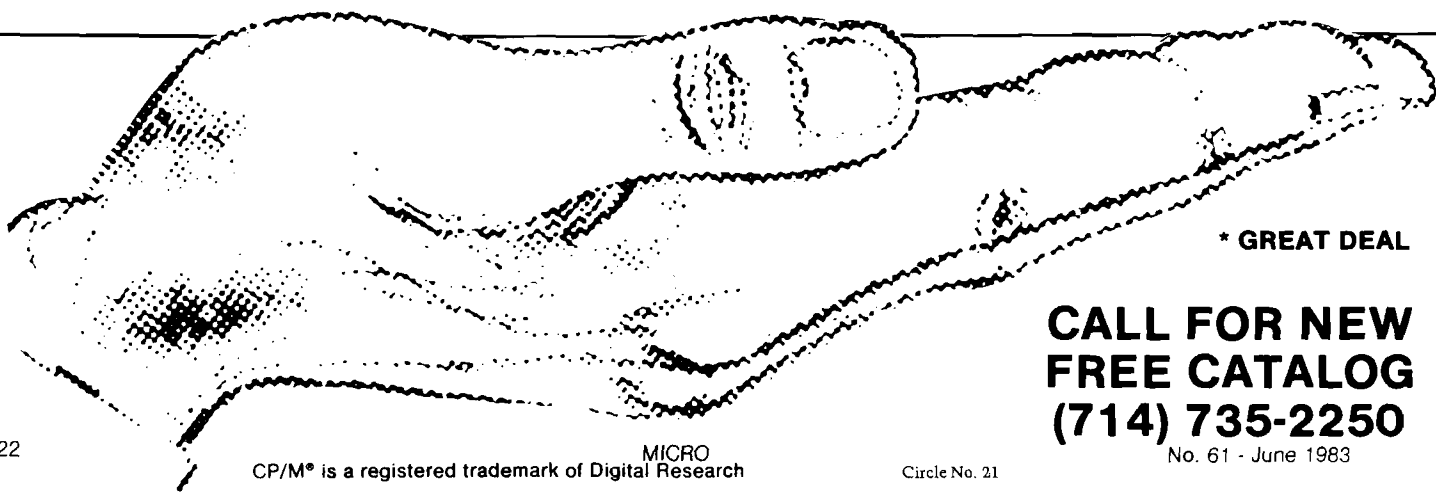
DISKETTES	List	ARK
Elephant 5¼"s/s s/d (box of 10)	29.95	19.95
Verbatim 5¼"s/s d/d (box of 10)	45.00	29.95
Verbatim 5¼" d/s d/d (box of 10)	65.00	44.95

MONITORS	List	ARK
*Comrex 12" Gr.	149.95	89.95
NEC Gr. Screen	285.00	169.9
Amdek Color I	449.95	349.9

Great Games	List	ARK
*Mandy's new all time favorite . . . Miner 2049er	39.95	29.95
Pinball Constr. Set	39.95	29.95
Zaxxon	39.95	29.95
Eggs It	29.95	21.95
Dark Crystal	39.95	29.95
Time Zone	99.95	69.95
Blade of Blackpoole	39.95	29.95

DISCLAIMER

WE ACCEPT VISA/MASTERCARD, PERSONAL CHECKS (ALLOW 10 DAYS TO CLEAR) OR COD ($2.00 CHARGE). PLEASE INCLUDE 3% FOR SHIPPING ($2.00 MIN.) OR 5% FOR BLUE LABEL ($3.00 MIN.). FOREIGN SHIPPING 10% ($5.00 MIN.). CALIFORNIA RESIDENTS ADD 6% SALES TAX. ALL ITEMS ARE NEW AND CARRY MANUFACTURERS WARRANTY. PRICES AND AVAILABILITY ARE SUBJECT TO CHANGE WITHOUT NOTICE.



*** GREAT DEAL**

CALL FOR NEW FREE CATALOG (714) 735-2250

CP/M® is a registered trademark of Digital Research

Circle No. 21

(Continued from page 20)
application you can use, then by all means, use it. Generally speaking, none of them do everything well, or even the same as standard DOS, and many of them won't work on anything that is sophisticated - assemblers/editors, word processing, or programs with a write-protection scheme. In other words, about 75% of my disks. If you have a large collection of simple BASIC programs, then one of these may suit you quite well.

PASCAL from Apple Computer Co.

The Pascal operating system requires a 16K RAM card in slot 0. I certainly recommend an 80-column card in slot 3. Having worked in Pascal in 40-column mode, I found it difficult to follow the screen flipping sideways to accommodate the long lines. A nice option is to have a clock card, such as the CPS Multi-function card, included. This will set the system date for you and allow a parallel printer in slot 1 and serial I/O in slot 2.

The text editor is powerful and easy to use, once you get the hang of it. The assembler is a non-standard version, which I avoid using when possible. The compiler is too fussy, and while diagnosing a zillion errors, should correct some of them for you, especially the silly ones, like where all those ';'s belong.

If Pascal were easier to use, I might be tempted to write more programs with it. A straightforward writing of the simplest program starts with

1. Loading the OS from APPLE1:.
2. Calling the text editor.
3. Typing in the appropriate program text.
4. Saving the WORKFILE.
5. Exiting the text editor.
6. Calling the compiler (that's when you discover all those 'syntax errors').
7. Making a list of the mistakes.
8. Calling the text editor.
9. Correcting the errors.
10. Saving the WORKFILE.
11. Quiting the text editor.
12. Compiling the program.
13. Executing the program.
14. Discovering a flaw in the program logic (which sends you back to step 7). That's assuming you corrected all the syntax errors on the first go-round.

Not only is this process inconvenient, but the system drive has to be on-line most of the time, or you get 'diskitis' of the thumb from swapping disks. A two-drive system is minimal but three are better. If you really want to cut the turn-around time between edit/compile, get a hard disk. They were invented with Pascal in mind.

I did not intend to give the impression that I don't like Pascal as a language. I just don't like the Apple implementation. A microcomputer is a one person computer and shouldn't make you wait. If you are familiar with time-sharing systems that make you wait no matter what language you are using, then Pascal is probably a better choice than some of the other languages available on those systems.

ORCA/M from Hayden Software

A new entry into the language development system market, ORCA/M is a self-contained OS and (currently) assembler. In the near future, Hayden intends to add a Pascal compiler into the system. (All the commands are already included.) The Operating System is packed into $B000-BFFF. Some of the system is also incorporated into the various overlaying parts of the program - text editor, assembler, linker and soon-to-be compiler.

While omitting the BASIC file commands (for obvious reasons), they have included several disk utilities not normally resident in Apple DOS: a PEEK command that invokes a sector editor; a VOLUME command that sets the disk volume number; a TIME command for the current date and time (if you have a clock); an APPEND command that adds a disk file to the current file in memory; a CHECK command that looks for bad sectors, lists a warning if a file is endangered and marks the sector as unusable in the VTOC; a COMPRESS command to either alphabetize the directory or move deleted files to the end of the directory; a COPY command that works like 'FID'; a RESTORE command to restore deleted files; and a SWITCH command to switch directory entries in the catalog.

All commands are memory resident and may be abbreviated to the shortest definable string. The command search is linear, so that a 'C' command would produce the first command with starting with a 'C'. To address a different 'C' command, only enough letters have to be typed to distinguish it from the other 'C' commands.

The text editor produces 'S' (for Source) type files, and the assembler creates 'R' type relocatable object files. The R file must be 'linked' into an address-oriented, BRUNable binary file. During linking, a subroutine library is searched for undefined addresses, allowing the addition of often used routines to the program without having to manually include them in the source file. ORCA/M has a SUB.LIB file with a raft of subroutines for your use. The assembler uses a fairly standard source file format.

Flex for the Apple from Norell Data Systems

Norell has adapted TSC Flex for the Apple computer using their FLEX09 6809 board. Flex is a small, easily adapted OS because it is mainly disk resident. Almost all of the commands reside on disk and are called into use by typing the appropriate command name. This means the system commands are easily modified or appended because each module is an executable binary file. It also means the system disk must be on-line all the time. Two drives are necessary to perform most functions.

The TSC text editor that comes with the system is an elemental line-oriented editor. It would be great with a hard-copy terminal, but most of us work with CRTs. Now, I have some good and some bad news. The good news is that there are several good screen-oriented text editors available for Flex. The bad news is that you would have to get the source file and type it into the Apple, because the disk format is totally different at the hardware level.

Flex has several convenient commands from the system level. The 'P' and 'O' commands, prefaced to any other command, send the output to printer or disk respectively. The 'LIST' command will print any ASCII (BASIC or Text type) file. 'APPEND' will join any number of files (of any type) together into a single file. 'COPY' is like 'FID', including wildcards, but has no prompting. Sometimes you copy programs you didn't intend to by mistake.

A nice feature of Flex is the ease of changing the system commands. At MICRO, we have rewritten several of the commands and added several commands to the original list. Any 6809 machine-language executable file can be a command if its name ends with '.CMD', and it can run at $C100.

Unfortunately, TSC's extended BASIC is not available as of this date, but Norell Data Systems considers implementation of the Apple version a high priority.

OS-9 from MICROWARE

OS-9 is a newer and more advanced operating system than Flex. Flex is a holdover from the 6800 microprocessor days and was rewritten in 6809 code, but not updated. OS-9 was originally written in 6809 code and utilizes the capabilities of the microprocessor more fully. It includes more advanced technology, such as multi-tasking, and in its higher levels, a multi-user environment.

The OS commands are similar to Flex: they are disk resident and called when needed. However, OS-9 allows loading commands into memory as an option, and removing them when they are no longer needed. Since OS-9 takes up more room in memory than Flex, I still recommend two disk drives.

I don't have access to another system that runs OS-9, so I can't compare it to a standard implementation. The Apple OS-9 disks are DOS 3.3 compatible, and as such must not be compatible with the rest of the OS-9 world.

It is a real pleasure, when working with a word processor, to finish one letter and send it to the printer, while at the same time, working on the next letter on the stack without having to wait for the printer to get done before starting in again (there goes the coffee breaks). There doesn't seem to be any particular limit to the number of tasks that can be specified to run at the same time, and I gave up after running four tasks simultaneously.

The BASIC that comes with OS-9 is called BASIC09 and is the most Pascal-like language I have seen, without being called Pascal. It includes named procedures with parameter passing, data typing, print using, IF. .THEN. .ELSE. .ENDIF, REPEAT. .UNTIL, WHILE. .DO. .ENDWHILE, LOOP. .ENDLOOP, EXITIF. .THEN. .END EXIT, and variables and line numbers (optional) local to procedures. The text editor checks syntax during line entry, and loop and subroutine nesting upon exit from the editor. It almost makes BASIC too easy. Compilation is fast and errors return you to the text editor, text file intact. A trace mode aids in debugging. The BASIC file can be PACKed to remove REMs and spaces and reduce the space requirements of the program. BASIC09 computes to 9 decimal digits.

The text editor is line-oriented, common to most BASICs. It uses commands to move from line to line, as line numbers are optional, includes line insert and delete, and has string search and replace. The Debug mode allows controlled program execution, trace on/off, variables examined/changed, procedure-nesting listing and stepping one or more steps through the program.

BASIC09 improves many of the shortcomings of the standard BASIC language by incorporating ideas and structures of the PASCAL language, without adding the faults of PASCAL and losing the interactivity of BASIC.

CP/M from Digital Research

The Appli-Card from Personal Computer Products, Inc., arrived at our office too late to be included in last month's new boards article, so I will briefly mention its attributes now. The board includes a six megahertz Z-80 microprocessor and 64K of RAM. The software includes CP/M 2.2 and drivers for an 80-column card or a 70-column hi-res display, horizontal scrolling of up to 255 characters screen width, and display of all 96 printable ASCII characters. It also has a rewritten FORMAT command that includes formatting, copying the CP/M tracks, and disk copy.

The documentation is clearly written in a step-by-step fashion that should allow someone without extensive computer knowledge to install an Appli-Card and have it running in less than 30 minutes. I also booted up Wordstar, MBASIC, GBASIC, and FORTRAN in less time than it takes to write about it. I waited about eight months to receive this card, and it was worth the wait.

The only problem that I've found is that the CPS Multi-function card software won't recognize the Appli-Card as a CPM card, and the program exits immediately, meaning that I have to use the old Apple Serial card to do my printing. The company is currently working on drivers for specialized cards.

The Control Program for Microcomputers is probably the most popular microcomputer operating system, and if you are looking for a system with a large installed base and corresponding software availability to add to your Apple (no small amount of software already), then CP/M is the system for you. In spite of its problems and slowness, the number of people currently using the system assures its place in the future of microcomputers.

One complaint is that the DIR command does not display the length of the files, although that can be determined with the STAT command. Another complaint is the amount of memory that CP/M requires compared to DOS. A standard 48K Apple without DOS (just for reference, not really too useful — have you used a tape recorder lately?) has 47101 bytes of free memory. Adding DOS to the system reduces the total to 36349 bytes. With MBASIC you have 32883 bytes free without the use of hi-res graphics. GBASIC contains hi-res graphics commands and further reduces the amount of memory free to 23793 bytes. That is on a 64K Z-80 board. If you have a RAM card in slot 0, you can increase the amount of space available for programs to 46076 bytes by moving DOS onto the card.

The Microsoft BASICs available, both M and G, are a more standard implementation (like the big machine versions) than Applesoft, and are easier to learn for someone who knows another BASIC. The file commands eliminate the Control-D, and include reading EOF. The text editor also allows whatever indentation and line spacing that you desire.

Wrapping it all up

Since the Apple is so versatile due to its plug-in slots and OEM support by Apple Computer, many additional operating systems are available that increase the quantity of useable software. The one big disadvantage with all the other systems is the lack of disk compatibility between the Apple Disk][and other microcomputer disk drives. This means that while the programs will execute without many changes, getting the programs into Apple disk format is the biggest hurdle.

MICRO

You may contact Phil at MICRO, P.O. Box 6502, Chelmsford, MA 01824.

U.C.S.D. PASCAL DIRECTORY

A byte and bit-level demonstration of a brute force method to access U.C.S.D. directory blocks.

by Steven Lesh

The F(iler program, which comes as part of the Apple Pascal Language System, provides the applications software *developer* with a number of useful capabilities; e.g., listing titles for files currently in the diskette directory, deleting files from the directory, and changing the date to be associated in a directory entry for each newly written file. Unfortunately, these capabilities are not conveniently available to the applications program *user*.

In this article, the format of the Apple Pascal diskette directory is examined. In addition to information about specific files, the first directory block contains information relating to the entire diskette. Since the structure of the fields containing this information resembles that of functionally similar fields for a file directory entry, the format for a directory entry is covered first.

The Apple Pascal Language System supposedly conforms to U.C.S.D. Pascal specifications, and so what you read here should apply to other implementations of U.C.S.D. Pascal as well. Hopefully, enough information about directory entry formats will be provided to allow you to develop most directory-related capabilities your applications software might require. Following the article is a listing for an intrinsic unit that demonstrates how to access file directory entries.

FILELIST requires:
Apple with Pascal

The FILELIST unit source code accompanying this article provides a 'bare bones' directory list capability. It performs a function similar to the U.C.S.D. Pascal Filer L(ist Directory command. FILELIST differs from the Filer L(ist command in that it allows the user to restrict the directory listing to titles for a specific file type even though the file name contains no information about what type of file it is. The procedure INITTITLEREAD permits a user to specify a drive number from which the directory is to be read, choose between displaying a file titles list on the monitor or listing it on the printer, and to select the type of file for which directory entries are to be listed (i.e., text, code, or data). If this information is already known, the procedure TITLEREAD can be called directly.

Non-Apple users may have to change the starting and ending directory block addresses (STARTDIR and ENDDIR in FILELIST) to addresses applicable to the U.C.S.D. Pascal implementation for their system.

To illustrate the format of a diskette directory entry, let us examine an entry for the file 'SYSTEM.WRK.TEXT'. If the Filer E(xtended Directory List command is used, the following information would be provided for this file:

```
SYSTEM.WRK.TEXT  4  27-Sep-82  255
   text
```

The hexadecimal characters stored on disk to represent this entry are:

```
(file block addresses & file-type
   identifier)
                         nul
FF 00 03 01 03 00
(file name)
   S  Y  S  T  E  M  .  W  R  K  .  T  E  X  T nul
0F 53 59 53 54 45 4D 2E 57 52 4B 2E 54
45 58 54 00
(file creation date)
stx
02 B9 A5
```

File Block Addresses and File Identification Byte

The first data items in a U.C.S.D. directory entry are the starting and ending block addresses for the file and the file-type byte. The ending block address is actually the address of the first block following the block(s) occupied by the file with which the ending address is associated in the directory.

A directory block address consists of a two-byte hexadecimal value representing the number of the block at which a file begins or ends. On the Apple diskette there is room for 280 file blocks. Since one byte can represent a maximum value of 255 decimal, when a block address of 256 or greater is reached, a '1' is added to the second byte of the value containing the block address and the left byte of the block address "wraps around" starting at x'00' again. For the example file, the block addresses would be translated as follows:

```
255 (starting address)
      FF 00
259 (ending address)
      03 01
```

Following the ending block address for the file is a file-type byte. The three file-type values I have been able to identify are:

x'02' = code file
x'03' = text file
x'05' = data file

The file identification byte for a code file can be distinguished from the x'02' ('stx') character that sometimes precedes the date (see Locating the File Creation Date below) by the 'nul' character that always immediately follows the file identification byte.

File Title

The next grouping of characters in a directory entry stores the file name. For the example file, the string of hexadecimal characters that represent the file title 'SYSTEM.WRK.TEXT' is:

```
S  Y  S  T  E  M  .  W  R  K  .  T  E  X  T nul
0F 53 59 53 54 45 4D 2E 57 52 4B 2E 54
45 58 54 00
```

Fifteen bytes are allocated for file title storage regardless of the actual title length. The file title itself is immediately preceeded by a one-byte value giving the length, in characters, for the file name (x'0F' in the example). When a title is R(emove'd from a disk using the Filer R(emove command, this value is changed to a 'nul' character. (Unfortunately, restoring an accidently removed file seems to require more than reinserting the proper file title length again.)

Locating the File Creation Date

Following the fifteen bytes allocated to file title storage and preceding the date are two bytes whose exact use I was not able to determine. The directory entries for files generated by U.C.S.D. system software — e.g., the E(ditor, C(ompiler, Library etc. — appeared to have a 'nul' character after the file title followed by a 'stx'. (In the hexadecimal breakdowns for the example shown above, I grouped the 'nul' character with the file title.) The byte preceding the date in files generated by applications software using the system procedures REWRITE and CLOSE contained a 'soh' (x'01') character; however, dates for some system 'data' files were preceded by another 'nul' character. By looking for 'nul', 'soh', and 'stx' I was able to find the dates for all files on my Pascal diskettes using FILELIST.

File Creation Date

The file creation date (day, month, and year) is the last item in the directory entry. An elegant scheme is used to pack all the date information into two bytes (one word). The two bytes containing the date for the above entry are B9 A5. The bit allocations for the date word are as follows:

```
dddd mmmm yyyy yyyd
where: 'd' = day
       'm' = month
       'y' = year
```

Five bits are required for a range of values from 1 to 31 to represent the days of the month. In the U.C.S.D. date-word, however, these bits are not contiguous. The leftmost four bits help represent the day of the month, but the next four bits are used to represent the month. Storing a numerical equivalent for a month in the second four bits presents no difficulty; there are 16 possible values and only 12 months. So let's take a look at how the year is stored.

In any given century there are 0 to 99 years. If a value representing one of these years is doubled, it still fits nicely into the last eight bits; in fact, the rightmost bit will always be zero. This is precisely the way the year is stored. Why? Because the right-most bit of the date word is used to help represent the day. For any day after the sixteenth of the month, a '1' is added to the byte containing the doubled value for the year. Thus, if the value contained in the rightmost eight bits is odd, 16 is added to the value contained in the leftmost four bits to derive a value for days in the last half of the month. The status of the rightmost bit is ignored when determining the value for the year stored in the other seven bits.

Diskette Title, System Date, and Block Address Information

Information about an Apple Pascal diskette is located in the first 26 bytes of block two. The array locations given below assume that you have read block two into a PACKED ARRAY of type CHAR.

The field specifying the first block in which code, text, or data files can be stored begins in block array element three. Armed with this information, you can thieve a couple of blocks from the directory if you need them, and if your directory does not need room for 77 files, of course.

***Listing 1:* FILELIST Program**

```
{ $L #6: }
{ $C @ STEVEN LESH 1982 }
{ $S+ }
UNIT FILELIST;
INTRINSIC CODE 18 DATA 26;
INTERFACE
CONST
   ALLFILES=Ø;CODEFILE=2;TEXTFILE=3;DATAFILE=5;
   STARTDIR=2;ENDDIR=5;
VAR
   MONTH,DAY,YEAR:STRING;
   OUTPUTDEV:INTERACTIVE;
{ SUPPLY TWO 'CHARS' CONTAINING THE DATE IN SYSTEM FORMAT }
PROCEDURE READDATE(FIRSTDATECHAR,LASTDATECHAR:CHAR;
                   VAR DAY,MONTH,YEAR:STRING);
{ CALL TITLEREAD DIRECTLY IF YOU KNOW:        }
{ 1-THE DRIVE # FOR THE DIRECTORY TO BE READ }
{ 2-THE TYPE OF FILE FOR WHICH DIRECTORY     }
{    ENTRIES ARE TO BE PRINTED               }
{    **YOU CAN LIST ALL FILES IN THE **      }
{    **      WITH A FILETYPE OF 'Ø' **       }
{ 3-THE OUTPUT DEVICE FOR A FILE LIST        }
PROCEDURE TITLEREAD(DISKUNIT:INTEGER;
                    FILETYPE:INTEGER;
                    PRINTTITLE:BOOLEAN);
{ INITTITLEREAD ASKS THE USER:                      }
{ 1-SCREEN OR PRINTER OUTPUT FOR FILE TITLES LIST? }
{ 2-READ THE DIRECTORY FOR WHAT DISK DRIVE?        }
{ 3-LIST TITLES FOR WHAT FILE TYPE (OR ALL FILES)? }
PROCEDURE INITTITLEREAD;
{ $P }
IMPLEMENTATION
VAR
   DISKNO,FILETYPE,PRINT:CHAR;
   DISKUNITS:SET OF CHAR;
   PRINTIT:BOOLEAN;
   NUMROFILES,FILECOUNT,VOLNO,FILEID:INTEGER;
PROCEDURE HALTDISPLAY;
```

(Continued on next page)

The eight-byte field allocated to the diskette title (or "volume name") starts in the seventh element of the character array with the first character specifying the length of the diskette title.

The field giving the total number of blocks available on an Apple Pascal diskette starts in the fifteenth element and the number of files currently in the directory is found in the seventeenth element of the character array into which block two is read.

The same format used for the file creation date associated with each entry in the directory is also used to store the current system date (i.e., the date associated with any newly written files). The current system date is stored starting in the twenty-first element of block two. The FILELIST procedure WRITESYSDATE should be called prior to creating new or updating existing disk files to set the current system date.

Conclusion

With an understanding of the way file directories are stored on disk, a variety of procedures could be developed to give Language System applications software users more control over vital program disk files. Beyond merely emulating existing F(iler program capabilities, new file maintenance capabilities could be developed: e.g., date-stamping compiled program listings, changing the date associated with existing directory entries and encoded prefixes and suffixes to allow longer, more meaningful file names.

It would be nice if Apple, Softech, or an ambitious reader would provide us with a 'fleshed out' library of units that emulated the capabilities of the Language System F(iler program. Until this happens, however, we must fend for ourselves. I hope this article will be of some use to those of you seeking to add file maintenance capabilities to your U.C.S.D. applications software.

Steven Lesh has programmed telecommunications computer system software for the last eight years. Programming became a hobby when the first microcomputers were marketed, though he still programs an old UNIVAC 9300 to support his habit. You can reach Mr. Lesh at General Delivery, Sierra Vista, AZ 85616.

Listing 1 *(continued)*

```
BEGIN
   WRITELN('PRESS ''C'' TO CONTINUE..');
   REPEAT
      READ(KEYBOARD,PRINT)
   UNTIL PRINT='C';
END;
{ $P }
PROCEDURE READDATE;
TYPE
   { THESE SUBRANGES MUST BE ALLOWED TO ACCEPT 'Ø' FOR }
   { INTERMEDIATE AND EXCEPTION PROCESSING             }
   DAYS=Ø..31;
   MONTHNMRS=Ø..12;
   YEARS=Ø..99;
VAR
   DAYNUM:DAYS;
   MONTHNUM:MONTHNMRS;
   YEARNUM:YEARS;
   WORKAREA:INTEGER;
BEGIN
   WORKAREA:=ORD(FIRSTDATECHAR);
   MONTHNUM:=WORKAREA MOD 16;
   IF WORKAREA > 15 THEN DAYNUM:=WORKAREA DIV 16
      ELSE DAYNUM:=Ø;
   WORKAREA:=ORD(LASTDATECHAR);
   IF ODD(WORKAREA)=TRUE THEN DAYNUM:=DAYNUM + 16;
   YEARNUM:=WORKAREA DIV 2;
   CASE MONTHNUM OF
      1:MONTH:='JAN';
      2:MONTH:='FEB';
      3:MONTH:='MAR';
      4:MONTH:='APR';
      5:MONTH:='MAY';
      6:MONTH:='JUN';
      7:MONTH:='JUL';
      8:MONTH:='AUG';
      9:MONTH:='SEP';
      1Ø:MONTH:='OCT';
      11:MONTH:='NOV';
      12:MONTH:='DEC';
   END;{ CASE MONTHNUM }
   STR(DAYNUM,DAY);
   STR(YEARNUM,YEAR);
END;
{ $P }
PROCEDURE TITLEREAD;
CONST
   BLOCKSIZE=512;
   BOTTOMLINE=22;
   NUL=Ø;
   SOT=1;
   STX=2;
VAR
   DATECHECKED,DATEFOUND,FIRSTBLOCK,MIDFILE,MIDTITLE:BOOLEAN;
   PRINT:CHAR;
   BLOCKTEXT:PACKED ARRAY[Ø..BLOCKSIZE] OF CHAR;
   TITLELINE:PACKED ARRAY[Ø..15] OF CHAR;
   BLOCKINDEX,DIRBLOCKINDEX,LINECOUNT,
   PRINTINDEX,DATEFINDER,TITLEINDEX,TITLELENGTH:INTEGER;
   DISPLAY:STRING[8];
BEGIN
   IF PRINTTITLE=TRUE THEN DISPLAY:='PRINTER:'
      ELSE DISPLAY:='CONSOLE:';
   REWRITE(OUTPUTDEV,DISPLAY);
   DATECHECKED:=FALSE;
   FILECOUNT:=Ø;
   FIRSTBLOCK:=TRUE;
   LINECOUNT:=Ø;
   MIDFILE:=FALSE;
   MIDTITLE:=FALSE;
   FOR DIRBLOCKINDEX:= STARTDIR TO ENDDIR DO
   BEGIN
      UNITREAD(DISKUNIT,BLOCKTEXT,BLOCKSIZE,DIRBLOCKINDEX);
      IF FIRSTBLOCK=TRUE THEN
      BEGIN
         TITLELENGTH:=ORD(BLOCKTEXT[6]);
         BLOCKINDEX:=7;
         FOR TITLEINDEX:=1 TO TITLELENGTH DO
         BEGIN
            WRITE(OUTPUTDEV,BLOCKTEXT[BLOCKINDEX]);
            BLOCKINDEX:=BLOCKINDEX+1;
         END;
         WRITELN(OUTPUTDEV,':');
         NUMROFILES:=ORD(BLOCKTEXT[16]);
      END;
```

(continued)

Listing 1 *(continued)*

```
REPEAT { STEPPING THRU BLOCK }
   { FIND A FILE TITLE }
   WHILE (MIDFILE=FALSE) AND
         (BLOCKINDEX < BLOCKSIZE-2) AND
         NOT (BLOCKTEXT[BLOCKINDEX] IN
              [CHR(CODEFILE),CHR(TEXTFILE),CHR(DATAFILE)])
                DO BLOCKINDEX:=BLOCKINDEX+1;
   IF (MIDFILE=FALSE) AND
      (BLOCKINDEX < BLOCKSIZE-2) AND
      (BLOCKTEXT[BLOCKINDEX+1] = CHR(NUL)) THEN
         BEGIN
            { GET CHARACTER FOR TITLE LENGTH }
            TITLELENGTH:=ORD(BLOCKTEXT[BLOCKINDEX+2]);
            IF (TITLELENGTH > 0) AND
               (TITLELENGTH < 16) THEN FILECOUNT:=FILECOUNT+1;
            { IF FILE TITLE FOUND SET UP TO GET CHARACTERS }
            IF ((BLOCKTEXT[BLOCKINDEX] = CHR(FILETYPE)) OR
               (FILETYPE=ALLFILES)) THEN
            BEGIN
               IF (FIRSTBLOCK=FALSE) OR
                  { THE NEXT CHECK PREVENTS A LOW BLOCK ADDRESS }
                  {     --FOR THE FIRST DIRECTORY ENTRY ONLY--   }
                  { FROM BEING MISTAKEN FOR A FILETYPE BYTE      }
                  { >THIS IS NOT NORMALLY A PROBLEM FOR THE    < }
                  { >THIS IS NOT NORMALLY A PROBLEM FOR THE    < }
                  { >STANDARD PASCAL FILE TYPES UNLESS YOU     < }
                  { >REDUCE THE NUMBER OF BLOCKS ALLOCATED TO  < }
                  { >THE DIRECTORY OR USE THIS CODE WITH OTHER < }
                  { >PASCAL-BASED LANGUAGES (E.G. PILOT)       < }
                  ((FIRSTBLOCK=TRUE) AND (BLOCKINDEX > 29)) THEN
                  BEGIN
                     MIDTITLE:=TRUE;
                     TITLEINDEX:=0;
                     MIDFILE:=TRUE;
                     BLOCKINDEX:=BLOCKINDEX+3;
                  END;
            END;
         END;
   { GET FILE NAME FOR DISPLAY }
   WHILE (BLOCKINDEX < BLOCKSIZE-1) AND
         (MIDTITLE=TRUE)  DO
         BEGIN
            REPEAT
               TITLELINE[TITLEINDEX]:=BLOCKTEXT[BLOCKINDEX];
               BLOCKINDEX:=BLOCKINDEX+1;
               TITLEINDEX:=TITLEINDEX+1
            UNTIL (TITLEINDEX=TITLELENGTH) OR
                  (BLOCKINDEX=BLOCKSIZE);
            { SEE IF ANOTHER 'BLOCKREAD' REQD TO FINISH TITLE }
            IF TITLEINDEX = TITLELENGTH THEN
            BEGIN
               MIDTITLE:=FALSE;
               { POSITION TO END OF TITLE SPACE FOR SHORT TITLES }
               IF TITLEINDEX < 15 THEN
               BEGIN
                  DATEFINDER:=15 - TITLELENGTH;
                  BLOCKINDEX:=BLOCKINDEX + DATEFINDER;
                  { DO THIS FOR SHORT TITLES ENDING ON A BLK BNDRY WITH }
                  { UNUSED CHARS BEFORE THE DATE IN THE NEXT BLOCK }
                  IF BLOCKINDEX > (BLOCKSIZE-2) THEN
                   DATEFINDER:=BLOCKINDEX-BLOCKSIZE;
               END;
               { SET INDEX TO WHERE DATE 'STX' SHOULD BE }
               BLOCKINDEX:=BLOCKINDEX + 1;
            END;
      END;
{ FIND THE FILE CREATION DATE }
WHILE (MIDFILE=TRUE) AND
      (BLOCKINDEX < BLOCKSIZE - 1) AND
      (DATECHECKED=FALSE) DO
      BEGIN
         DATEFOUND:=FALSE;
         { FOR TITLES WHICH END ON A BLOCK BOUNDARY }
         IF BLOCKINDEX=0 THEN BLOCKINDEX:=DATEFINDER+1;
         IF (BLOCKTEXT[BLOCKINDEX] IN
            [CHR(STX),CHR(SOT),CHR(NUL)]) THEN DATEFOUND:=TRUE
            ELSE
               REPEAT
                  BLOCKINDEX:=BLOCKINDEX+1;
               UNTIL (BLOCKTEXT[BLOCKINDEX] IN [CHR(NUL),CHR(STX)]);
         IF (BLOCKTEXT[BLOCKINDEX] = CHR(STX)) THEN DATEFOUND:=TRUE;
         DATECHECKED:=TRUE;
      END;
{ DISPLAY A TITLE }
IF (TITLEINDEX > 0) AND (DATECHECKED=TRUE) THEN
BEGIN
   { PRINT FILE NAME }
```

(continued)

```
            FOR PRINTINDEX:=0 TO TITLELENGTH-1 DO
               WRITE(OUTPUTDEV,TITLELINE[PRINTINDEX]);
            { TAB TO DATE-WRITE AREA }
            FOR PRINTINDEX:=0 TO 17-TITLELENGTH DO
               WRITE(OUTPUTDEV,' ');
            { CONVERT DATE FOR DISPLAY }
            IF DATEFOUND = TRUE THEN
            BEGIN
               READDATE(BLOCKTEXT[BLOCKINDEX+1],BLOCKTEXT[BLOCKINDEX+2],
                    DAY,MONTH,YEAR);
               WRITE(OUTPUTDEV,DAY:2,'-',MONTH,'-',YEAR);
            END
               ELSE WRITE(OUTPUTDEV,'CAN''T FIND DATE');
            WRITELN(OUTPUTDEV,);
            LINECOUNT:=LINECOUNT+1;
            DATECHECKED:=FALSE;
            MIDFILE:=FALSE;
            IF (LINECOUNT=BOTTOMLINE) AND (PRINTTITLE=FALSE) THEN
            BEGIN
               HALTDISPLAY;
               LINECOUNT:=0;
            END;
         END;
         { BUMP TO THE NEXT CHARACTER FOR CHECKS }
         IF MIDFILE=FALSE THEN BLOCKINDEX:=BLOCKINDEX+1;
         { IF WE HAVE PROCESSED ALL FILE TITLES ON THE }
         { DISK EXIT W.O. READING ALL DIRECTORY BLOCKS }
         IF FILECOUNT > (NUMROFILES-1) THEN
         BEGIN
            HALTDISPLAY;
            CLOSE(OUTPUTDEV,NORMAL);
            EXIT(TITLEREAD);
         END;
      { END REPEAT BLOCK STEPPING }
      UNTIL BLOCKINDEX > BLOCKSIZE -1;
      FIRSTBLOCK:=FALSE;
END; { FOR DIRBLOCKINDEX:= STARTDIR TO ENDDIR }
CLOSE(OUTPUTDEV,NORMAL);
{ ALLOW TIME TO READ LAST SCREEN }
HALTDISPLAY;
END;
PROCEDURE INITTITLEREAD;
BEGIN
   REPEAT
      DISKUNITS:=['1'..'6'];
      PAGE(INPUT);
      REPEAT
         WRITELN('TYPE ''P'' FOR HARDCOPY;');
         WRITELN('''S'' FOR SCREEN OUTPUT..');
         READ(KEYBOARD,PRINT)
      UNTIL PRINT IN ['P','S'];
      IF PRINT='P' THEN PRINTIT:=TRUE
         ELSE PRINTIT:=FALSE;
      REPEAT
         WRITELN('LIST TEXT FILE DIRECTORY FOR WHICH DISK?');
         WRITELN('**TYPE 1-6..');
         READ(KEYBOARD,DISKNO)
      UNTIL DISKNO IN DISKUNITS;
      CASE DISKNO OF
         '1':VOLNO:=4;
         '2':VOLNO:=5;
         '3':VOLNO:=9;
         '4':VOLNO:=10;
         '5':VOLNO:=11;
         '6':VOLNO:=12;
      END { CASE DISKNO };
      REPEAT
         WRITELN('ENTER FILE TYPE..');
         WRITELN('  0 -> ALL FILE TYPES');
         WRITELN('  2 -> CODE FILES');
         WRITELN('  3 -> TEXT FILES');
         WRITELN('  5 -> DATA FILES');
         WRITELN('  9 -> **QUIT**');
         READ(KEYBOARD,FILETYPE);
      UNTIL FILETYPE IN ['0','2','3','5','9'];
      CASE FILETYPE OF
         '0':FILEID:=0;
         '2':FILEID:=2;
         '3':FILEID:=3;
         '5':FILEID:=5;
         '9':FILEID:=9;
      END { CASE FILETYPE };
      IF FILEID < 6 THEN TITLEREAD(VOLNO,FILEID,PRINTIT);
UNTIL FILEID=9;
END;
BEGIN
END.
```

MICRO

When something new comes along most of us tend to be conservative about giving up the familiar. A good example of this behavior is the use of structured programming languages on microcomputers. Languages like Pascal have been available to us for a couple of years; they are easier to program in and more efficient than BASIC. However, only a relative minority of microcomputer users have switched to a structured language, and none of the major manufacturers offer anything other than BASIC as standard equipment.

Recently a number of new operating systems have come on the market. These operating systems bear about the same relationship to the currently dominant systems (CP/M, Apple DOS, Flex) as Pascal does to BASIC. They are more powerful and, usually, easier to use.

OS-9

a structured operating system

by Mark G. Boyd

Structured operating systems have the same type of structure found in a structured program, looking like an upside-down tree. The highest level is called the root node and is the overall control structure and most abstract part of the system. The root node is connected to the highest level of branch nodes, each of which are connected to their own set of branch nodes on the next lower level. On any level, a node may not be connected to any lower-level nodes. This type of node is called a leaf node and is connected only to a single branch node on the next higher level. In a structured program the leaf nodes are the most detailed part of the program. In a structured operating system they are the I/O device drivers, the data files, and the lowest level routines in the programs.

There are different types of tree structures used in structured programming and as data structures. I will restrict my discussion to one particular type: the hierarchical structure. In this structure the root node is called the ancestor of all other nodes. The next lower level of nodes are the children of the root node, each of which may have children of its own. As you work down the tree, each level of nodes represents a new generation of children. Each node has only one parent node, but it may have any number of child nodes. A path from the root node to any other node in the structure is simply a list of descendants, starting with the root node and ending with the desired node. The path from the root node to any other node is unique. Any node may be reached from any other node by working up the structure until a common ancestor node is found and then working down to the desired node.

Data flow in a hierarchical structure is allowed only along the paths connecting the nodes. All data is local to the procedures/files that are the nodes. Data may be passed from a parent to its child or from a child to its parent; it cannot be passed to any other node without working through a path that involves a common ancestor. This system sounds complex but, as you shall see, it is the basis for very simple, but powerful operating systems.

OS-9, which uses this hierarchical structure and is a by-product of BASIC-09, is a result of Motorola's 6809 development process. The software was developed simultaneously with the hardware it is designed to use. Motorola and Microware, realizing that BASIC-09 needed an equally well-structured operating system to support it, developed OS-9 to allow BASIC-09 users realize the full potential of a modern programming language.

I feel OS-9 is the best of the new operating systems. It is one of the most powerful 8-bit operating systems available today and is the only truly powerful operating system that can run on a relatively small system. A 24K-byte system can support OS9, and 48K system can run several users simultaneously in a high-level language. A fully extended OS-9 system can have 1 megabyte of main memory, hard disk drives, and many users.

A Structured Operating System

OS-9 is a descendant of UNIX, the Bell Telephone Laboratories operating system for large minicomputer systems. UNIX has become the standard for multiprogramming minicomputer systems because of it's versatility, power, and elegantly simple design. OS-9 looks much like UNIX, but its actual operation is quite different. UNIX dynamically swaps programs into memory from large, fast disk systems. OS-9 cannot do this because of the slow disk systems used with microcomputers. In order to support mulitple users, OS-9 makes use of position-independent re-entrant programs in RAM or ROM. Because the programs are re-entrant, multiple users can use the same code while maintaining different data and stack areas, and because the code is position independent, it can be brought into memory, in any available location, as needed. These two factors allow OS-9 to be much more efficient in its use of RAM and ROM than UNIX.

Multiprogramming means that the system can execute many different programs simultaneously. CPU time is divided in time slices (about .1 seconds in OS-9), which are allocated by the system to the various tasks currently running and to the operating system overhead. With a powerful microprocessor like the 6809 and a well-designed multiprogramming operating system like OS-9, users are not aware that they have only part of the CPU time.

OS-9 can run processes sequentially or concurrently. Each process is created by an existing process and may in turn create child processes of its own. This leads to a hierarchical structure where all processes are descendants of the original process run when OS-9 was brought up. To the user the root of each tree of processes is a process called Shell. A Shell process is executed when the system is started, and a new incarnation of Shell is created for each user who logs onto the system. Shell is a command interpreter that accepts input from the user and creates new processes in response to that input. Shell also can pass parameters to the process it creates.

When a Shell creates a process it puts itself into a waiting state until that process is finished. The user has the option of causing the Shell to create another incarnation of itself before it goes into its waiting state. This new Shell then can be used to create a new process and another incarnation of Shell, allowing a single user to make effective use of OS-9's multiprogramming capabilities. I'll give an example of this at the end of the article.

A hierarchical system also is used for all input and output. The top level (or root) of this tree is a general I/O manager. At this level all data is essentially a stream of bytes. Data being sent out to an external device passes down through the tree to a device driver, which is a leaf node (for single file devices) or the root node of the device file structure (for disk drives). The characteristics required for the data actually sent to the device are added as it passes down the tree. For example, data going to the disk would be blocked to sector size at one level, the appropriate preamble and postamble added at another level, and the actual sectors to be written determined at yet another level. Input data undergoes the reverse of this process as it passes up the tree. Data at the top, which is the data passed to or from a process, has the same structure regardless of its course or its destination.

The device independence of I/O data has two advantages. First, it means that I/O paths can be redirected at any time. A program that normally outputs to the printer can have its output redirected to a disk file for later printing. Input to the Shell can be redirected to allow a disk file to control the system. In other words, printer spooling and procedure files are inherent in OS-9. Second, OS-9 programs are essentially hardware independent. The program is not aware of the device driver modules, so it cannot depend on the details of the I/O hardware. A program that runs on a small OS-9 system with one minifloppy and a serial printer will run, without modification, on a large system with hard disks and a chain printer. No more problems with software transportability or system upgrades!

The heirarchical structure extends to the file structure on the disk drive(s). A file is accessed by specifying a path to it. This path is simply a list of all of its direct ancestors. Each entry in the list, except for the first and last, is a directory file. Directory files are the branch nodes of the tree structure and contain only the names of their immediate descendants and pointers to them. The leaf nodes of the tree are the actual data or program files. The first entry in the path list may be a device driver (e.g.,/D1) or it may be a directory file in the current directory. The last entry is the name of the desired file. If the desired file is in the current directory, only the last entry is required.

Finally, lets explore some Shell commands. These commands are entered in response to the prompt OS9 and consist of a process name that may be followed by parameters for the process, a parameter that modifies the amount of memory used by the process, parameters that redirect the input/output paths of the process, and finally by a parameter that results in concurrent processing (i.e., creates a new incarnation of Shell). The parameters are separated by spaces and the entry is terminated by a return. Some examples are:

1. OS9: LIST FILE 7
2. OS9: COPY FILE3 FILE7
3. OS9: LIST /D1/ASSM-FILES/BPROM
4. OS9: LIST /D0/MARK-LIB/LETTER >/P1
5. OS9: LIST FILE7>/P1 & EXECUTE #7K

The first example runs the process LIST with input from FILE7, which lists FILE7 to the terminal. The second example runs the process COPY with input from FILE3 and output to FILE7. So far things are much the same as in any DOS.

The third example runs the process LIST. Input is from the file BPROM. The path list specifies that BPROM is listed in the directory file ASSM-FILES, which itself is listed in the primary directory for the disk mounted in drive D1. The fourth example is similar to the third but it also demonstrates output redirection (>) to the serial printer driver /P1. This process lists the file LETTER to the serial printer.

The fifth example introduces concurrent execution (&). The Shell creates a process that starts listing FILE7 to the printer. Then it creates another Shell that starts the process EXECUTE (EXECUTE is allocated 7K of RAM (#7K). All processes have a certain minimum amount of RAM that they require. This information is stored on the disk with each process. The #7K is an execution modifier that can be used to allocate larger amounts of RAM at the time the process is created.

On my system EXECUTE is a BASIC-09 program. This information is noted by the system and, when a process using EXECUTE is created, BASIC-09 is loaded automatically and instructed to run the process. EXECUTE must be in packed form and located in the CMDS directory, but that's a subject for another article.

References

1. *OS-9 Level I Operating System V1.1 Users Guide*, Microwave Systems Corporation, 1981.
2. *OS-9 Level I Operating System V1.1 System Programmer's Manual*, Microware Systems Corporation, 1981.

Dr. Boyd teaches chemistry, physics, and computer science at a small liberal arts college. He may be contacted at St. Mary of the Plains College, Dodge City, KS 67801.

Color Disk BASIC:

observations and utilities

by Michael Dudgeon and William Clements, Jr.

The disk system for the Color Computer (CoCo) has been available for over a year now and has proved to be quite popular with CoCo owners. The DOS is ready to go at power-up, doesn't tie up a disk drive in reading program overlays, and since it is in ROM, it can't be overwritten by some renegade program. Best of all, it is easy to use; the commands are simple and direct, with many being easily understood by a complete novice.

In the year that we've used the CoCo DOS, we have discovered a few things that aren't specifically documented by Radio Shack, and have written some utility programs that we'd like to pass along to other disk users. We'll also discuss the structural details of BASIC and machine-language program files on disk. With this information, you can create new files, or modify old ones, directly from the keyboard. You can scroll through a file, or even through an entire disk, and explore the contents on the disk. You can back up the directories on all your disks, using sectors that are "hidden" from the operating system, as a safeguard against directory crashes that can help in recovering accidently KILLed files. You can list to the screen a complete summary of granules and sectors that a file occupies, and you can change individual bytes within a file without rewriting the whole file. Finally, we'll give you a program that lets you print a disk directory in a compact file-list table, including the start, end, and exec addresses for machine-language files, appropriate for taping onto the disk jacket.

Disk Format and File Structure

There are three separate parts to a program file: the directory entry, the file-allocation table entries, and the filed data. The disk is formatted into 35 tracks, with eighteen 256-byte sectors per track. The directory is contained on sectors 3-11 of track 17, and the file-allocation table is on sector 2 of the same track. Sectors 1 and 12-18 of track 17 are not used by the DOS.

Each track is divided into two 4096 byte granules, sectors 1-9 comprising one granule and sectors 10-18 the other. The granules are numbered 0 (track 0, sectors 1-9) through 31 (track 16, sectors 10-18), and 32 (track 18, sectors 1-9) through 67 (track 34, sectors 10-18), skipping the directory track. The directory is located in the middle of the disk to minimize head travel; the directory must always be accessed first when program file operations are carried out by BASIC, and then the sectors containing the actual file are read.

The Directory Format

The directory structure is completely documented on pages 58-59 of the Disk System Owner's Manual and Programming Guide, so we won't repeat every detail here. The directory entries are contained in the first 16 bytes of each 32-byte cluster — beginning in sector 3 of track 17 and stored in the same order that the files were originally created on the disk. Each entry contains file name and extension, a file-type flag, a binary/ASCII flag, the number of the first granule in the file, and the number of bytes used in the last sector of the file.

Sector 2 of the directory track is the file-allocation table, which uses bytes 0-67. Each byte indicates the type of use being made of the granule having the same number as the byte. A value $FF means that granule is not part of a file. A value in the range $0-$43 (0-67 base ten) means that granule is part of a file; the value contained there is the number of the next granule used by the file. A value in the range $C0-$C9 means the corresponding granule is the last one used by that file. The second hex digit (0-9) is the number of sectors in the granule that the file uses, counting from the first sector in the granule. Note that the four lowest order bits (bits 0-3) in the word therefore give the number of sectors, rather than bits 0-5 as the manual says. Table entries in the range $0-$43 and $C0-$C9 form a linked list of the granule allocations to every file.

When a file is killed, the first character of the file name is set to $FF, and the entries in the allocation table that correspond to the granules containing the file are also set to $FF. This destroys all information explaining where a file was stored. The file itself is left unchanged and will be overwritten by new data if the sectors are re-used. The FREE function of BASIC reports the number of table entries in the allocation table that currently equal $FF.

How Program Files Are Stored

Let's look at how a BASIC program is stored in RAM, as a binary file on disk, and as an ASCII disk file. For our example, we'll choose a simple two-line program:

```
10 INPUT A
20 PRINT A;SQR(A):GO TO 10
```

BASIC stores its program lines in tokenized form, replacing all commands and functions with a one-byte code as the lines are entered. The interpretation of lines thus starts even before a program is run, saving some execution time and using less memory.

TABLE 1. RAM STORAGE OF BASIC TOKENIZED LINES

The pointer to the first location used for the lines is in locations $19-$1A. The lines are stored in the following format: two-byte address of next BASIC line, two-byte line number, the BASIC tokenized line, and a one-byte end-of-line marker ($00).

Example: 10 INPUT A
20 PRINT A, SQR(a) : GO TO 10

We find that $19-$1A contains $2601. At $2601, we find the following data:

Hex Loc.	Hex Byte	Comments	Hex Loc.	Hex Byte	Comments
$2601	26	Next line	$2610	3B	;
	09	at $2609		FF	Function flag
	00			9B	SQR token
	0A	Line #10		28	(
	89	INPUT token		41	A
	20	Space		29	)
	41	A		3A	:
	00	End-line marker		81	Token for GO
$2609	26	Another line		20	Space
	1E	would go here		A5	Token for TO
	00			20	Space
	14	Line #20		31	1
	87	PRINT token		30	0
	20	Space		00	End-line marker
	41	A			

While listing or editing the lines, the tokenizing process is reversed to recover the original text. The pointer to the first BASIC statement is in locations $19 and $1A (25 and 26 decimal), and the address of the first free location after the last line is in locations $1B and $1C (26 and 27 decimal).

If we enter the example program with the disk system installed and peek at locations 25-27, we find $2601 and $2620. Peeking at memory in between these limits gives the tokenized BASIC program in Table 1.

Now let's see how this list of bytes looks on disk. Let's take a freshly initialized disk and save the program under the name "TEST.BAS", which records it in binary (tokenized) format. The directory information and the sector containing the program statements can easily be examined using the DISKLOOK program, which we'll describe later. The information found there is summarized in Table 2. The first sector of the file begins with a one byte beginning-of-file mark ($FF) and two bytes containing the total number of bytes in the file. Then the tokenized BASIC statements are copied verbatim from RAM, complete with the address links and zero markers between lines, filling sector after sector and granule after granule as required to hold the entire program.

Table 3 shows the file contents if the program is stored in ASCII format using SAVE "TEST.BAS",A. The tokenized lines in memory are untokenized and converted back to the way they were originally typed in, except for question marks that share the same token as PRINT and thus appear as PRINT in a listing. The file contains the list of program lines with the marker byte $0D before and after every line.

BASIC uses a slightly different format to store machine-language programs, since several absolute addresses must be associated with each file. Table 4 gives a six-byte test program and shows how it is stored. Notice that the end address is not stored with the file; the start address and total number of program bytes are saved at the front of the program, and the execute address appears at the end.

Saving Disk Programs On Cassette

While we're talking about storage of Disk BASIC programs, here is a word of caution about the cassette storage of disk programs. When the computer is powered up, the initialization routines put a table of pointers to the BASIC functions and commands into lower RAM starting at $0120. If you have the Extended BASIC ROM, the pointers to those additional keywords are added to the table, and if the disk system is plugged in, pointers to the disk command and function lists are appended. Be sure the disk controller pack is plugged in if you save a program containing Disk BASIC statements onto cassette tape! Likewise, the disk pack must be in place whenever you read that tape back in. Otherwise, the table of pointers to legal keywords won't contain the ones for Disk BASIC. When you try to list a program containing unrecognizable tokens, BASIC goes ahead with the list but prints an exclamation point in place of the offending tokens as a signal that it couldn't figure out what keyword to put there.

Suppose, for instance, you want a printed listing of a Disk BASIC program but have no printer. You save the program on cassette and take it to a friend who has a printer but no disks. When he reads it in and lists it, he'll get '!' signs for all the disk-system keywords: you can see those now and then in published listings of programs for the disk system. If you use a cassette tape to hold Extended Disk BASIC programs, take your disk controller pack along with the tape; the ribbon cable and drives aren't needed if you just want a listing, but the Disk ROM must be there before a CoCo can understand your tape!

Some Utility Programs For the Disk User

Now that we know the exact form in which the disk system stores programs, we can go in directly and alter the directory, fix a bad byte in the middle of a program, or construct our own files by POKEing in directory entries and file sectors directly from a program. We could even convert a BASIC program file into a machine-language file, and *vice versa,* by properly modifying the directory entry and changing the marker bytes at the beginning and end of the file. Many of these tricks are useful if you need to salvage a damaged file or a miswritten directory. The latter conversion might be helpful in overlaying or chaining program segments that have conflicting numbers by reading a BASIC file into a section of unused RAM and POKEing new line numbers into the statements. Then, by using a machine-language merge and move routine, you can combine them with the original program. Or you can just use the programs as an aid in learning about how Disk BASIC works.

1. The DISKLOOK Program. Listing 1 presents a multipurpose utility for examining and changing all kinds of files, including the directory itself. The program begins in the "disklook" mode, requesting a granule number and a sector number. When that information is

provided, the program reads the sector into memory. Beginning with the first byte in the sector and continuing until the screen is full, it prints the byte count, the hex value of the byte and its ASCII character equivalent. To continue scrolling through the contents of the sector, press the space bar or 'enter'. To reverse-scroll, hit the up-arrow; to interrupt the scrolling and specify another granule and sector, press '@'. If you wish to look at the directory, enter 'D' instead of a granule number. The byte count starts at 0 when the directory is read out, to correspond with the listings in the Disk System Manual and to allow the byte numbers in the allocation table to correspond with the granule numbers. For all other granules, the bytes are numbered starting with 1.

You may also enter a subcommand mode whenever the screen stops scrolling. A 'D' key initiates a search through the directory sectors for the first unused position. Then you are prompted for the information needed to create a new directory entry (you must use as the extension either .BAS, .BIN, or .TXT). A 'C' allows you to change one byte in the sector under examination by specifying the byte number and the new value. An 'F' (for file-analysis) prompts for a file name, then lists all information contained in the directory about that file, including file type and mode, an ordered list of the granules used, the number of sectors in the last granule, and the number of bytes in the last sector.

2. The DIRDUPL Program. If you read the Disk System Manual carefully, you'll see that BASIC leaves nearly half the directory track unused. Sectors 1 and 12-18 can be used for other purposes, such as scratch storage or private files. We use them to back up the information contained in the directory. Most disk users at one time or another have gotten a mangled directory due to power failure, a power-line spike, or other mishap that occurs just as you are writing the disk. Another cause of crashed disks is the corrosion that forms on the contact fingers of the disk drive, causing intermittent connections.

You can use DIRDUPL to back up directory sectors 2-9 into sectors 12-18 and 1, respectively, and then later to rewrite the directory from the backup if it ever becomes necessary. Sectors 10 and 11 are not backed up due to lack of space; however, these are not normally used unless you have more than 54 files on one disk.

By the way, DIRDUPL can provide an easy way to restore a killed file. The KILL command doesn't alter the file, it just flags the directory entry and wipes out that file's granule numbers in the file-allocation table so they all can be used again. If you kill a file and want to recover it later, you can do so by restoring the original directory provided that none of the file's granules have been reused. Of course, if the file space has been overwritten, the original file is unrecoverable by any method.

3. The DISKLIST Program. Several programs have been published that give a printed listing of the names/extensions, length, type, addresses, and other statistics associated with the files on a disk. The most elaborate file statistics we've seen are those generated by F. S. Flack's program in *Color Computer News*, August, 1982, p. 11. Another program giving less detail but an easier-to-read listing is C. J. Roslund's program in *The Rainbow*, March, 1982, p. 31. Other programs have appeared that send the output of the DIR command to a printer. We wanted a program that would provide more information than the DIR command, yet would not use an entire printed page so that the user is forced to

TABLE 2. DISK STORAGE OF BASIC PROGRAM IN BINARY FORMAT

If the program of Table 1 is saved to disk, the sector containing the program will have the following bytes:

Byte No.	Byte	Comments	Byte No.	Byte	Comments
1	FF	Start marker	10	87	PRINT
2	00	Total length	11	20	
3	1F	of file	12	41	A
4	26	Address of	13	3B	;
5	09	next line	14	FF	
6	00	Line #10	15	9B	SQR
7	0A		16	28	(
8	89	INPUT	17	41	A
9	20		18	29	)
A	41	A	19	3A	:
B	00	End-line marker	1A	81	GO
C	26	Where another	1B	20	
D	1E	line could go	1C	A5	TO
E	00		1D	20	
F	14	Line #20	1E	31	1
			1F	30	0
			20	00	End-line marker

TABLE 3. DISK STORAGE OF BASIC PROGRAM IN ASCII FORMAT

The same example program, stored in ASCII format, would appear in its sector as follows:

Byte No.	Byte	Comments	Byte No.	Byte	Comments
1	0D	Start/end marker	15	20	Space
2	31	1	16	41	A
3	30	0	17	3B	;
4	20	Space	18	53	S
5	49	I	19	51	Q
6	4E	N	1A	52	R
7	50	P	1B	28	(
8	55	U	1C	41	A
9	54	T	1D	29	)
A	20	Space	1E	3A	:
B	41	A	1F	47	G
C	0D	Start/end marker	20	4F	O
D	32	2	21	20	Space
E	30	0	22	54	T
F	20	Space	23	4F	O
10	50	P	24	20	Space
11	52	R	25	31	1
12	49	I	26	30	0
13	4E	N	27	0D	Start/end marker
14	54	T			

TABLE 4. DISK STORAGE OF A MACHINE-LANGUAGE PROGRAM

As a sample program in machine-language, consider the following:

Hex Loc.	Machine Code	Mnemonic
6000	86 AA	LDA #$AA
6002	B7 65 00	STA $6500
6005	39	RTS

The command SAVEM "TEST.BIN", &H6000, &H6005, &H6000 produces the following bytes in the sector used to hold this program:

Byte No.	Byte	Comments
1	00	"Begin" marker
2	00	Program is
3	06	6 bytes long
4	60	Start address
5	00	is $6000
6	86	
7	AA	
8	B7	The
9	65	program
A	00	bytes
B	39	
C	FF	"End" marker
D	00	
E	00	
F	60	Execute address
10	00	is $6000

keep the summary separate from the disk. We extensively modified Roslund's program and came up with DISKLIST.

DISKLIST prints the file name/extension, notes whether the file is binary or ASCII, lists the number of granules used, and prints the start, end, and exec addresses for machine-language programs. Most important, it does this on a printout that is just the right size to fasten to the disk jacket. If the table contains more files than will fit on one side of the jacket, DISKLIST prints "continued on back" and finishes the listing with a new table that can be attached to the reverse side of the disk jacket.

The name/extension, file-type flag and ASCII flag, and the three addresses for machine-language files are saved in arrays; if you ever want to do anything else with the information, such as alphabetize the names for instance, just add the program code onto the end of our listing. We think you will agree that this is a very handy way to keep track of disk directories.

In Conclusion

We hope that this discussion of the makeup of BASIC and machine-language program files will give you the information you need to make better use of the disk system in your computing efforts. We also hope that the utilities will be useful, particularly in light of the high cost of many "Disk Doctor" and DISKZAP-type programs that are available commercially. We haven't seen any of them, so we aren't in a position to compare their features with ours, but we feel that what we have done is worth at least the price of this magazine.

We were going to tell you how to beat that intermittent contact problem, weren't we? Unfortunately, there probably isn't any way to cure it permanently, short of soldering everything together, but we think we have found the next best thing. There are products sold for just this purpose: that is, keeping contact fingers clean on circuit boards in critical commercial equipment. We use Gold-Wipes ®, made by the Texwipe Company, Upper Saddle River, NJ 07458. These are small, foil-sealed packets containing pads soaked in a solvent and contact-conditioning agent selling for about 25¢ apiece in boxes of 100. Friends or members of a computer club could go together and buy a box, but even if you have to shell out the whole $25 yourself, it's well worth it. A treatment every month or so with these pads has cured our dirty-contact blues, and until Radio Shack wakes up and puts gold-plated connectors on their disk equipment, it should be a good fix for your system too.

You may contact the authors at Dept. of Chemical & Metallurgical Eng., U. of Alabama, P.O. Box 2662, University, AL 35486.

Listing 1

```
10 'DISKLOOKUTILITY
20 'BY MICHAEL DUDGEON AND BILL CLEMENTS
25 'COPYRIGHT @ 1983 by MICRO Ink
30 CLEAR2ØØØ:DIM E(2Ø)
40 CLS3:PRINT@43,"DISKLOOK";
5Ø PRINT@96,"ENTER GRANULE NO. IN HEX ";
   :LINEINPUTG$:IF G$<>"D" THEN 7Ø
6Ø T=17:INPUT"SECTOR(1-18)";S:GOTO 9Ø
7Ø G=VAL("&H"+G$):IF G>33 THEN T=INT(G/2)+1
   ELSE T=INT(G/2)' TRACK NO.
8Ø S1=G/2-INT(G/2):IF S1=Ø THEN INPUT"SECTOR(1-9)";S
   ELSE INPUT"SECTOR(1Ø-18)";S
9Ø S1=Ø:PRINT"TRACK"T"SECTOR"S:DSKI$ Ø,T,S,A$(1),A$(2)
1ØØ FOR Y=1 TO 2:IF T=17 AND S=2
    THEN PRINT"BYTE NUMBERS=GRANULE NUMBERS"
11Ø IF T=17 THEN FOR X=Ø TO 127 ELSE FOR X=1 TO 128
12Ø IF T=17 THEN P$=MID$(A$(Y),X+1,1)
    ELSE P$=MID$(A$(Y),X,1)
13Ø PRINTUSING"%%   % %           % %";
    HEX$(X),HEX$(ASC(P$)),P$
14Ø IF X=Ø OR X/14<>INT(X/14) THEN 18Ø
15Ø ' <@>: RETURN TO DISKLOOK
    <D>: CREATE NEW DIRECTORY ENTRY
    <C>: CHANGE BYTE IN FILE  <F>: FILE ANALYSIS
16Ø A$=INKEY$:IF A$="" THEN 16Ø ELSE IF A$="@" THEN 4Ø
17Ø IF A$="D" THEN 2ØØ ELSE IF A$="C" THEN 32Ø
    ELSE IF A$="F" THEN 35Ø ELSE IF A$="↑" THEN X=X-28
18Ø NEXT X,Y:GOTO 4Ø
19Ø ' CREATE NEW DIRECTORY LISTING
2ØØ CLS2:PRINT"CREATE NEW DIRECTORY ENTRY";:
    FOR S=3 TO 9:DSKI$ Ø,17,S,A$,B$:C$=A$
    +LEFT$(B$,127):FOR I=1 TO 225 STEP 32:
    IF ASC(MID$(C$,I,1))<>Ø AND ASC(MID$(C$,I,1))
    <>255 THEN NEXT I,S' FIND FIRST UNUSED SLOT
21Ø IF I>128 THEN L9=2:I=I-128 ELSE L9=1
22Ø A$(1)=A$:A$(2)=B$
23Ø PRINT@96,"FILENAME.EXT: ";:LINEINPUTXX$:
    XX=INSTR(XX$,".")+INSTR(XX$,"/"):XY=
    LEN(XX$):R$=RIGHT$(XX$,3):XX$=LEFT$(XX$,XY-4)
    +STRING$(12-XY," ")+R$:IF R3$="DAT"
    THEN R2=255 ELSE R2=Ø
24Ø R3$=RIGHT$(XX$,3):IF R3$="BAS" THEN FT=Ø
    ELSE IF R3$="DAT" THEN FT=1 ELSE IF
    R3$="BIN" THEN FT=2
25Ø XX$=XX$+CHR$(FT)+CHR$(R2):X=1:
    INPUT"SECTORS IN LAST GRANULE";DD
26Ø LINE INPUT"ENTER GRANULE NOS. IN HEX,
    <@> AFTER LAST ONE. ";K$:E(X)=1+
    VAL("&H"+K$):IF K$="@" THEN X=X-1 ELSE X=X+1:
    GOTO 26Ø' INPUT GRANULE NOS. IN ORDER
27Ø DSKI$ Ø,17,2,A$,B$:FOR P=1 TO X-1:
    MID$(A$,E(P),1)=CHR$(E(P+1)-1):NEXT P
28Ø MID$(A$,E(X),1)=CHR$(&HCØ+DD):
    DSKO$ Ø,17,2,A$,B$' INSERT GRANULE NOS.
    IN FILE ALLOCATION TABLE
29Ø XX$=XX$+CHR$(E(1)-1):INPUT"NUMBER OF BYTES
    IN LAST SECTOR";Z:XX$=XX$+CHR$(Ø)
    +CHR$(Z)+STRING$(16,Ø)
3ØØ MID$(A$(L9),I,32)=XX$:DSKO$ Ø,17,S,A$(1),A$(2):
    SOUND 5,1Ø:CLS2:GOTO 5Ø
31Ø ' CHANGE BYTE
32Ø IF T=17 THEN EØ=1 ELSE EØ=Ø
```

(Continued)

Listing 1 *(continued)*

```
330 CLS4:PRINT@8,"CHANGE ONE BYTE";:PRINT@96,
    "BYTE NO.(HEX) TO CHANGE";:INPUTBB$:
    INPUT"NEW BYTE";NC$:MID$(A$(Y),VAL("&H"+BB$)+EØ,1)
    =CHR$(VAL("&H"+NC$)):DSKO$ Ø,T,S,
    A$(1),A$(2):CLS2:GOTO 5Ø
34Ø ' FILE ANALYSIS
35Ø CLS8:PRINT@41,"FILE ANALYSIS";:PRINT@96,
    "FILE NAME: ";:LINEINPUTF$:IF LEN(F$)<8
    THEN F$=F$+STRING$(8-LEN(F$)," ")
36Ø PRINT@128,"EXTENSION: ";:LINEINPUTE$:F$=F$+E$
37Ø FOR X=3 TO 9:DSKI$ Ø,17,X,A$,B$:
    IF INSTR(A$,F$) OR INSTR(B$,F$) THEN 38Ø
    ELSE NEXT X:PRINT"ENTRY NOT FOUND":
    FOR X=1TO2ØØØ:NEXTX:GOTO35Ø
38Ø X=INSTR(A$,F$):IF X=Ø THEN X=INSTR(B$,F$):
    A$=B$' X=BYTE NO. FOR ENTRY LOCATED
39Ø A$=MID$(A$,X,32)' A$ NOW=THAT SINGLE ENTRY REQUESTED
4ØØ C=ASC(MID$(A$,12,1)):IF C=Ø
    THEN PRINT"BASIC" ELSE IF C=1 THEN PRINT"DATA"
    ELSE PRINT"MACHINE LANGUAGE"
41Ø IF ASC(MID$(A$,13,1)) THEN PRINT"ASCII" ELSE PRINT"BINARY"
42Ø E(1)=ASC(MID$(A$,14,1))
43Ø N=ASC(MID$(A$,16,1))
44Ø DSKI$ Ø,17,2,A$,B$
45Ø X=1
46Ø E(X+1)=ASC(MID$(A$,E(X)+1,1)):
    IF E(X+1)>&HBF THEN SC=E(X+1)-&HCØ
    ELSE X=X+1:GOTO 46Ø
47Ø PRINT"GRANULES:":FOR P=1 TO X:PRINT"$"
    +HEX$(E(P)):NEXT P:PRINT"SECTORS IN LAST
    GRANULE:"SC:PRINT"BYTES IN LAST SECTOR: ";
    "$"+HEX$(N):PRINT:PRINT"HIT ANY KEY
    TO CONTINUE";
48Ø IF INKEY$<>"" THEN CLS3:GOTO4Ø: ELSE 48Ø
```

Listing 2

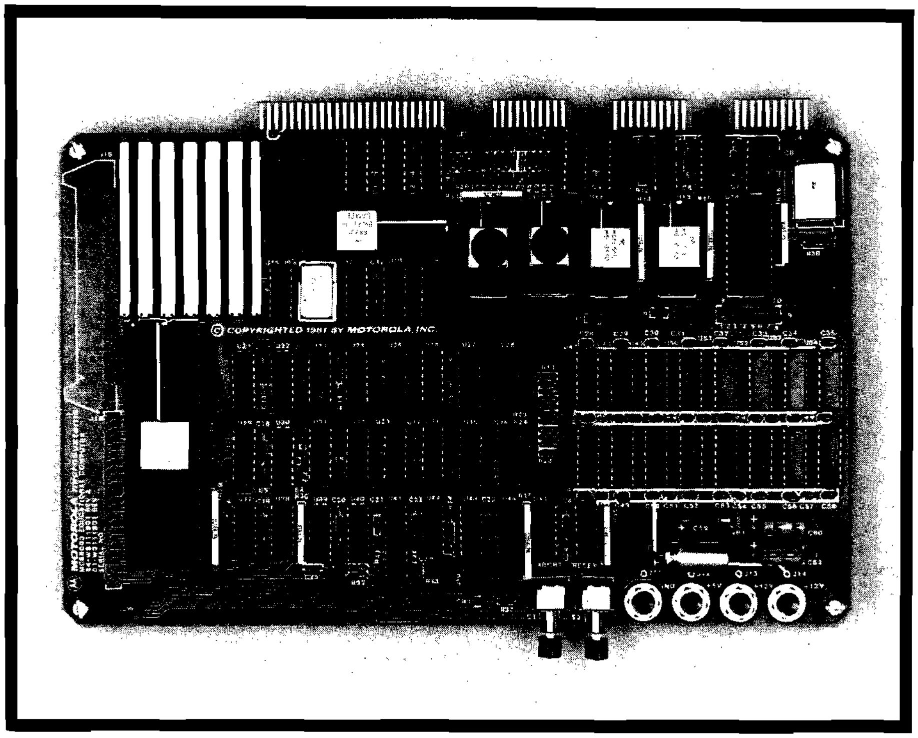
```
1Ø ' DIRDUPL UTILITY
2Ø ' DIRECTORY BACKUP AND RETRIEVAL
3Ø ' BY MICHAEL DUDGEON AND BILL CLEMENTS
35 ' COPYRIGHT © 1983 by MICRO Ink
4Ø CLEAR3ØØØ:CLS3
5Ø PRINT@5,"DISK DIRECTORY BACKUP":
   PRINT@69,"(1) BACK UP DIRECTORY":
   PRINT@1Ø1,"(2) RETRIEVE DIRECTORY":
   PRINT@165,"WHICH";:INPUTW
6Ø PRINT@229,"DRIVE NO.";:INPUTDN
7Ø ON W GOTO 9Ø,14Ø
8Ø GOTO5Ø
9Ø FOR X=2 TO 8
1ØØ DSKI$ DN,17,X,A$,B$:DSKO$ DN,17,X+1Ø,A$,B$
11Ø NEXT X
12Ø DSKI$ DN,17,9,A$,B$:DSKO$ DN,17,1,A$,B$
13Ø END
14Ø FOR X=2 TO 8
15Ø DSKI$ DN,17,X+1Ø,A$,B$:DSKO$ DN,17,X,A$,B$
16Ø NEXT X
17Ø DSKI$DN,17,1,A$,B$:DSKO$ DN,17,9,A$,B$
18Ø END
```

Listing 3

```
1Ø 'DISKLIST - DIRECTORY PRINTING UTILITY
2Ø ' BY BILL CLEMENTS
25 ' COPYRIGHT © 1983 by MICRO Ink
3Ø CLS:CLEAR2ØØØ:C$=CHR$(13):
   PRINTTAB(6)" DIRECTORY PRINTER"C$
4Ø DIMGR(67),N$(68),SA(68),EA(68),XA(68),T(68),F(68)
5Ø INPUT"DRIVE NO";DN:PRINT"PRINTED OUTPUT (Y/N)":
   LINEINPUT"(DEFAULT IS 'N'):";Q$:
   IFQ$="Y" THEN Q=-2 ELSE Q=Ø
6Ø L=Ø:LX=25:LINEINPUT"DISK NAME? ";D$:IFQ=Ø THEN CLS
7Ø PRINT#Q,TAB(1Ø)"DISK: "D$C$:GOSUB37Ø
8Ø DSKI$DN,17,2,A$,B$:B$=LEFT$(A$,68):
   FORI=1TO68:GR(I-1)=ASC(MID$(B$,I,1)):NEXT
   'LINKED LIST OF FILE GRANULES
9Ø FORI=3TO11:DSKI$DN,17,I,X$,Y$:
   X$=X$+LEFT$(Y$,116) 'GET DIRECTORY ENTRIES
1ØØ FORJ=ØTO7:L=L+1:JJ=32*J:N$(L)=MID$(X$,JJ+1,8)
    +"."+MID$(X$,JJ+9,3):G=ASC(MID$(X$,JJ+14,1)):
    FG=G 'NAME, EXTENSION, FIRST GRANULE
11Ø T(L)=ASC(MID$(X$,JJ+12,1)):F(L)=
    ASC(MID$(X$,JJ+13,1)) 'FILE TYPE, ASCII FLAG
12Ø IFF(L)=Ø THEN T$="BIN" ELSE T$="ASC"
13Ø B=ASC(LEFT$(N$(L),1)):IF B=Ø THEN 2ØØ
    ELSE IF B=255 THEN 21Ø 'SKIP IF KILLED OR UNUSED
14Ø FORK=1TO68:IF GR(G)<68 THEN G=GR(G):
    NEXTK 'SEARCH FOR LAST GRANULE
15Ø IF T(L)=2 THEN 23Ø 'GO FIND ML ADDRESSES
16Ø IFL>LX  THEN 35Ø 'COUNT FILES
17Ø PRINT#Q,TAB(3)N$(L)TAB(17)T$TAB(2Ø)K;
18Ø IF T(L)=2 THEN PRINT#Q,TAB(22)"$"+SA$+",$"+EA$+",$"+XA$;
19Ø PRINT#Q
2ØØ NEXTJ,I
21Ø PRINT#Q,C$TAB(12)"FREE GRANULES:";FREE(DN)
22Ø STOP
23Ø LS=GR(G)AND31 'NO. SECTORS USED IN LAST GRANULE
24Ø LB=ASC(MID$(X$,JJ+16,1)) 'BYTES IN LAST SECTOR
25Ø T=INT(FG/2)-(FG>=34) 'TRACK NO. OF FIRST GRANULE
26Ø S=1+9*(FG AND1) 'FIRST SECTOR FOR
    EVEN GRANULES =1, FOR ODD GRANULES=1Ø
27Ø DSKI$DN,T,S,A$,B$ 'GET ML ADDRESSES
28Ø SA(L)=256*ASC(MID$(A$,4,1))+ASC(MID$(A$,5,1)):
    SA$=HEX$(SA(L)):SA$=STRING$(4-
    LEN(SA$),"Ø")+SA$ 'START ADDRESS
29Ø EA(L)=SA(L)+256*ASC(MID$(A$,2,1))
    +ASC(MID$(A$,3,1))-1:EA$=HEX$(EA(L)):EA$=STRING$
    (4-LEN(EA$),"Ø")+EA$ 'END ADDRESS
3ØØ T=INT(G/2)-(G>=34) 'TRACK NO. OF LAST GRANULE
31Ø S=LS+9*(G AND1) 'LAST SECTOR NO.
32Ø DSKI$DN,T,S,A$,B$:A$=A$+LEFT$(B$,127)
33Ø XA(L)=256*ASC(MID$(A$,LB-1,1))
    +ASC(MID$(A$,LB,1)):XA$=HEX$(XA(L)):XA$=STRING$
    (4-LEN(XA$),"Ø")+XA$ 'EXEC ADDRESS
34Ø GOTO16Ø
35Ø PRINT#Q,C$TAB(9)"CONTINUED ON BACK"C$C$
36Ø GOSUB37Ø:LX=68:GOTO17Ø
37Ø PRINT#Q,TAB(7)"NAME"TAB(15)"TYPE,GR.
    START,END,EXEC":PRINT#Q,TAB(7)STRING$(32,".")
38Ø RETURN
```

MICRO™

The 68000 Educational Computer Board



by Terry A. Jackson

One of the exciting items in the micro world these days is the Motorola 68000 16-bit microprocessor. A recent equipment item from Motorola, the MEX68KECB educational computer board, should also prove to be quite popular with those who want to learn how to use the 68000. Recently I purchased one of these boards to help upgrade my skills as a microprocessor instructor, and I am quite pleased with its capabilities. In this article I outline what I feel to be the board's important features so that you can judge whether or not it suits your needs.

Hardware Highlights

The board contains a 4MHz 68000 processor, 32K of user-programmable dynamic RAM, a 16K ROM monitor, two EIA RS-232C serial ports, a cassette tape recorder port, a parallel printer port, a 24-bit general-purpose programmable timer, and a very small breadboard area (1 7/8" x 2 3/8"). You must provide a CRT terminal (or a teleprinter) and a power supply with +5-volt and ±12-volt outputs.

The 32K RAM memory removes a major limitation possessed by most of the popular "evaluation kits" and allows you to do some very ambitious programming. You can graduate from games to mini-interpreters, multitasking experiments, and other more sophisticated diversions. One of the RS-232C ports is assigned to the system terminal, and the other to a host system. The ports are separately jumper-adjustable to baud rates from 110 to 9600. The presence of a host is not required, since the system runs comfortably without it, but if such a luxury is available, the saving and reloading of programs is greatly enhanced. The cassette recorder port, operating at approximately 1300 baud, means economical mass storage for students with more limited resources.

The 7" x 10.5" - board is not a kit; it comes assembled and tested. Sockets are used for the 68000, the 68230, the 4116 RAMs, the monitor ROMs, the 6850 serial interface chips, and the 14411 baud-rate generator. An envelope of push-on jumpers is provided for selection of various options, and even a set of insulated spacers and screws is included for mounting the board. Two pushbuttons are mounted on the board; one restarts and reinitializes TUTOR, the ROM-resident monitor, and the other simply aborts any user program, saves registers, and returns control to TUTOR. The abort function is particularly useful for debugging if a user program is in an infinite loop.

Those who want to experiment with hardware add-ons may find the breadboard area adequate, but two connectors can be attached easily for more ambitious expansion. All data lines, fifteen address lines, and most control signals are brought to a 46-pin connector pattern, and a 2 x 25 (0.1" spacing) uncommitted connector pattern allows access to the upper address bit decoder, other control signals, port connections, or the breadboard area. The 68000 is designed to interface easily not only with its own family of peripherals, but also with the 6800 family.

I would like to pass along some helpful hardware-related hints. First, although the set-up instructions are generally quite detailed, there's a point I did not find mentioned. Your terminal must support the DTR (data terminal ready) line or your system will just stare back at you when you power up. Second, the markings on the lines to the terminal and to the host seem at first glance to be mixed up. Pin 3 on the host connector carries outgoing data and is marked TX DATA. Pin 3 on the system terminal connector carries incoming data and is also marked TX DATA. The reason for this is that the board is intended to look like a modem to the system terminal, and to look like a data terminal to the modem communicating with the host. With this perspective, everything looks normal. Third, many cassette tape recorders do not have the low side of the microphone input tied to the same point in the circuit as the low side of the output jack. If your recorder is one of these, check to see what the effect is of tying these two points together. On my machine, it simply changes the output level. If your recorder cannot tolerate having these two points tied together, you will have to connect one cable at a time, depending on whether you are dumping or loading. (This problem is not unique to the educational computer board. Every tape-oriented system I own or have encountered makes the same assumption, and I have witnessed resultant problems more than once.)

Software Features

A wide variety of keyboard commands is available using TUTOR. The most significant of these, the single-line assembler and the disassembler, deserve a detailed description. All valid 68000 operations can be entered in source form, one line at a time, from the keyboard. The object code is generated by TUTOR and stored sequentially in memory; however, the source lines are not saved. If you want to make changes, the memory-modify command with the disassembly option can be used to review and change selected instructions. The object code and disassembled equivalent source are displayed, one line at a time, with the option of entering a different source line or simply advancing to the next instruction. At any time, a printer-attach command can be given and the memory-display command with the disassembly option can be used to produce a disassembled hard copy of the entire program. Finally, the object code can be dumped to the host or to cassette tape for future reloading.

The inability to save source is not as much of a problem as it might seem at first. If further revisions are to be made, a program may be reloaded and the revise-disassemble-dump cycle repeated. The disadvantages of a lack of source storage capability and the inability to use symbolic addressing do not appear to me to be serious for small programming exercises suited to learning the instruction set, but the support of a host system with an editor and assembler would be needed to effectively utilize the 32K of RAM. The equipment manual describes the object program storage format completely. This permits anyone with access to a full-blown assembler with an incompatible format to write a format conversion program. (This is typically a relatively easy job once the initial and target formats are known.)

One of the keyboard commands sets up a transparent mode in which a direct path exists between the system terminal and the host. This mode allows you to do the entire program development task, editing, assembly, and debugging from a single operating position. A user-selectable exit character will return control to TUTOR.

Several keyboard commands are available to support program debugging. Trace and breakpoint capabilities are included. In trace mode, a program may be stepped through, one instruction at a time. After each step, current register contents are displayed, and the next instruction to be executed is disassembled and displayed. Tracing may be set to run continuously without operator intervention, or to pause after each step and wait for a go-ahead from the operator. If the trace command is preceded by a printer-attach command, a hard copy of the action can be obtained for later review. Up to eight breakpoints can be set at any one time. Each breakpoint may have an optional count entered as well. If a count, N, is entered, the program will halt just before the *Nth* execution of the designated instruction (and at each subsequent encounter), but not before. Breakpoints may be combined with continuous tracing if desired.

The only software flaw I have encountered is in the ASCII string handling for DC directives in the single-line assembler. If a blank is imbedded in the string, the assembler thinks you did not complete the line properly. This problem is avoided easily, and I understand will be fixed in a subsequent revision of TUTOR. The avoidance procedure consists of substituting the memory set command with its ASCII string option, and targeting the command to the same memory location that the disassembler was at when the problem was encountered.

The version of TUTOR that I have is 1.0; revision 1.1 is being shipped as of this writing, and revision 1.2 will be ready to go soon. Motorola sources indicated that enhancements as well as problem corrections can be expected in this newest version. It is apparent to me that they are supporting this product solidly as part of a program to capture a big chunk of the 16-bit market.

Conclusions

The educational computer board is an excellent choice for those who want to learn a lot about the 68000. It is a learning tool, not a computer to get data processing jobs done. Connected to a host with good language tools, the MEX68KECB's power is tremendously magnified.

The board requires no knowledge of hardware for its use, but permits hardware-oriented users to have some fun, too. No clever gimmicks interfere with almost unlimited hardware add-on projects.

Good software and thorough documentation, combined with a board that has lots of program room, make this an excellent buy at $495.

Terry Jackson is a quality control engineer at Electro-Motive, a locomotive manufacturer in La Grange, IL. He is also an assistant professor at Midwest College of Engineering in Lombard, IL, teaching courses in microprocessor applications. You may contact him at 147 E. View St., Lombard, IL 60148.

MICRO

A UNIX-like Operating System for 6809 microprocessors

by Stephen L. Childress

Unless you've been on sabbatical to Siberia of late you will have noticed the swell of interest in the Unix™ operating system software. Most new and all the old popular 16-bit computers are supporting Unix or one of the numerous look-alikes. Why all the furor? It seems at last we've begun to rethink computer programming and usage. Recognizing that software development is expensive and time-consuming, we must exploit the falling cost of today's hardware. In this article, I explain how a new operating system software philosphy is being applied to small microcomputers capable of supporting a MC6809 processor. This software is called "OS-9" and is a good example of the rewards of rethinking system software.

This article is not so much to sing the praises of OS-9 as it is to point out the disappointing fact that most new computers being introduced today are a rehash of the disk operating systems of the 1970's (CP/M, PC-DOS, Apple DOS, etc.) There is a real catch-22 here exemplified by the IBM PC, which has a large memory space; it is running an improved but non-the-less CP/M derivative and therefore does not take advantage of the memory size to make the system more cost-effective. Another good example is Apple DOS, which, when outfitted with a new inexpensive hard disk, must resort to treating it in BASIC with some 72 independent floppy disks.

To date, the 6809 has not found the success of the Z-80, et. al., not because it is inferior, but because it was introduced too near in time to the 16-bit chips. The 8086 and 8088 are used in systems that are not much better, faster, or cheaper than good Z-80 systems. But for the hundreds of thousands of byte-wide (8-bit) computers, OS-9 and the 6809 can be retrofitted to bolster the capabilities of existing systems. OS-9 is, at the time this is written, some two years mature and running on all of the "SS 50" computers, the Apple II, and several European computers. But lacking marketing giants like Tandy and IBM, the machines remain in the fringe areas and have a limited collection of off-the-shelf applications software. Rumor has it that Tandy is developing OS-9 for the Color Computer.

Time for Change

The Unix supporters favor a switch from yesterday's system software philosophies, which have evolved into stubborn, unwiedly enemies of the programmer and, consequently, the end-user. Bell Labs launched the Unix philosophy, the bottom line which might be described as: "Since software is increasingly expensive to develop, let's change the priorities from frugality in hardware and to abstraction in software design."

The Unix philosophy is reflected in the jargon: "Shell," "Kernal," "Filter," "Pipe," "Tee," and "Socket" - each suggests a simple abstract idea about data processing. For example, Kernal and Shell simply refer to parts of the operating system that can be compared to layers. The concepts Filter, Pipe, and Tee deal with problems such as data base management more easily than the older "query, sort, merge, and report." General abstraction does cost more in terms of hardware, but in today's systems where hardware is a small part of a complete system price, the new priorities are encouraging.

OS-9 Evolution

While Unix was finding its way from DEC PDP-11/45's and 70's to other minicomputers, Motorla and Microware Systems Corporation teamed to produce software that would exploit the capabilities of the new MC6809 "pseudo 16-bit" microprocessor chip, which has all the memory addressing mode power of the minicomputers. The feature that distinguishes micros originally intended for use in industrial controllers from computers for general use is strength in addressing modes. The idea these two companies had was to develop an extremely modular set of reusable software. The benefits of modular software has been known for some time; but it has been realized only in limited terms, requiring the programmer to use monolithic compilers, assemblers, and linkers to effect a merger of modules. Although this works well, it is time consuming and far from ideal. Motorola wanted modularization to the extent of mass-produced "software-on-silicon" (ROMs). Before processors of the 6809's power, such an idea was impractical because of the lack of addressing power.

What is the ideal format for modularization? Here is a wish list, with the scientific name of the species in brackets:

1. Software modules that can be placed anywhere in memory without reassembly or link-loading — just copy it to memory verbatim from some media such as a disk. [Position-independence]
2. Modules that reside in EPROM or ROM already plugged into the address space of the machine. This would be good for modules that are used often. [ROM-able, Reentrant]

3. Modules that intercommunicate in a standard fashion without subtle sneak-paths that can get fouled up. Forbid fragile, spider-web arrangements of software interaction. [Stack-oriented]
4. The modules should contain "pure code" only and the variables used by a module should be in RAM supplied by the modules' parent (caller).
5. Allow programmer to "activate" modules coming from the software toolbox. These should have all the flexibility of old, stable modules. [Loadable]
6. For non-ROM modules (RAM), allow those not needed for the job at hand to be removed from memory, making more space for other modules.
7. Since requirements change and mistakes do happen, allow a new module to temporarily supercede an old one without hassle. [Precedence]
8. Allow modules to be shared among several users. [Reentrancy]
9. The system software should worry about which language is being used in a particular module — assembly, BASIC, etc..
10. The modules should be able to perform I/O without any knowledge of who/what/how regarding the I/O devices.
11. And last but not least, KISS! That's "Keep It Simple, Stupid." Simple means small, and don't forget that modularization is supposed to mean that a non-Ph.D. can understand the overall system by concentrating on one piece at a time.

This is a tall order. What would the user's benefit be? The idea is to eliminate the aggravations caused by the older system philosophy. Consider this list of nuisances the computerist must face every day:

1. He needs to run Program B while in A, but A is incompatible with B because they both use the same memory region for their code.
2. A jury-rigged version of program B is made up to let B hide from A, say in high memory.
3. But A and B still fight over the same memory cells for variables. The hide and seek continues with some successes, some hoakey fixes, and a few subtle disasters.
4. The programmer would like to swap A and B but he needs fast (expensive) disks.
5. Multiple users on a micro? There's not enough memory for two copies of the 20K-byte language program let alone the 8K or so for the programs.
6. The operating system doesn't support multiple terminals.
7. The operating system is too complex and will not allow changing I/O conditions without surgery, hacking, patching, and kludging.
8. A change to the I/O would require adjustments to almost all software. But much of it is copy-protected and cannot be modified.
9. Why should the user have to spend $400 for a "print spooler" hardware box? Why can't the computer take 2% of its time to copy a disk file to the printer port without hassle?

These gripes go a long way back in time and remain today, even in improved operating systems like PC-DOS for the IBM. If you look, you'll see that the module wish list covers all of these headaches. Now let's look at what Motorola and Microware did to achieve these goals.

Figure 1 OS-9 Memory-Resident Module Directory

Module Directory at 21:28:13

Addr	Size	Typ	Rev	Attr	Use	Module	Name
F000	4D0	C1	1	r. . .	1	OS9p2	Kernal, part 2
F4D0	2E	C0	1	r. . .	1	Init	System initialization module
F4FE	1BA	C1	1	r. . .	1	Boot	System Boot media fetcher
F800	7BF	C1	1	r. . .		OS9	Kernal, part 1
BA00	7F	C1	1	r. . .	1	SysGo	Starts up main user on "TERM"
BA7F	193	E1	1	r. . .	8	ACIA	Driver for RS232 serial devices
BC12	38	F1	1	r. . .	2	TERM	Device module for 1st CRT
BC4A	651	C1	1	r. . .	1	IOMAN	Chief executive for all I/O
C29B	BBC	D1	1	r. . .	1	RBF	Chief of all File I/O, any device
CE57	41C	D1	1	r. . .	8	SCF	Chief of all Character I/O device
D273	33A	E1	1	r. . .		DC3	Device driver for mini-floppy
D5AD	2C	F1	1	r. . .		D0	Device module for minifloppy #0
D5D9	2F	F1	1	r. . .	1	H0	Device module for hard disk #0
D608	36	F1	1	r. . .	6	T1	Device module for 2nd CRT
D63E	216	E1	1	r. . .	1	WD1000	Device driver for hard disk
D854	CA	C1	1	r. . .	1	CLOCK	Device driver for 60Hz line clock
D91E	472	11	1	r. . .	2	Shell	Unix-like user interface (CLI)
DD90	2E	11	1	r. . .		Load	Utility to get module from disk
DDBE	48	11	1	r. . .		Unlink	Utility to remove module from mem.
AC00	1DA	11	1	r. . .	1	Mdir	Utility producing this report

The OS-9 Big Picture

The OS-9 operating system is constructed from modules of machine code, each of which has:

- A string of ASCII characters that gives the module a name
- A module "header" that gives the module type, size, etc.
- Module's revision number
- Distance to the module's first instruction
- For program modules (as opposed to subroutines), the memory storage (RAM) required by the module
- A CRC module checksum to prevent bogus disk files (text) or improper ROMs from being treated as modules

Programs consist of one or more modules and may call subroutines or programs located in other modules. Compilers and interpreters (e.g., BASIC) are merely program modules. If a module is invoked by some program and that module is not in memory, OS-9 will automatically retrieve the module from disk. When no program needs use of a certain module, OS-9 removes it from memory and returns that memory to the pool of uncommitted memory. Though the users really need not be concerned with where modules are in memory, the "MDIR" utility shows a directory of in-memory modules as shown in figure 1.

The module directory shows each module's name, address in memory, size, attributes (program, subroutine, language, etc.), revision, and "use" count. The I/O drivers and the peripherals (device modules) are each handled by independent modules subordinate to modules RBF or SCF,

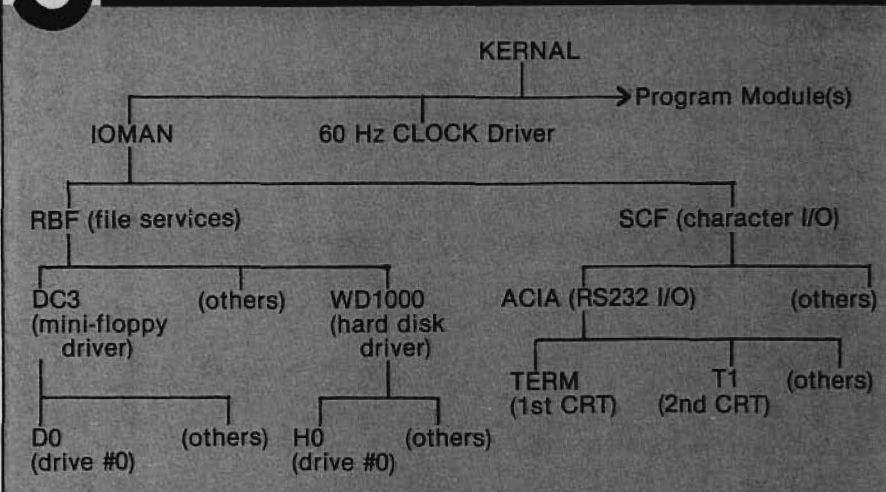


Figure 2 Major OS-9 Modules

which are, in turn, subordinate to IOMAN. In keeping with the module concept, device drivers and their device modules (e.g., WD1000 and H0) are loadable from disk. Thus, to add another disk drive, one merely prepares a new device module (a 5-minute job). To add a new type of peripheral, you merely write a new device driver (actually, paraphrase a similar old one), a one-day task. Note that these new modules do not affect the code within the existing modules in any manner whatsoever. Note also the small size of the peripheral drivers, which hints at their simplicity. Score one for the modular I/O goals in the wish list.

Now look at these modules from their kinship perspective rather than from the memory-map view (see figure 2). From this view these modules' functions are:

KERNAL — allocates and manages memory, time shares CPU among programs, coordinates inter-program signals, accepts and hands off jobs to appropriate I/O chiefs.

CLOCK — handles 60-per-second (power line) clock interrupts and keeps time of day and time-sharing slice intervals.

IOMAN — responsible for *all* requests for I/O, regardless of device. Interfaces programs in a uniform way to the various classes of I/O.

RBF — I/O control of random block-oriented devices such as disks. Takes care of directories, files, media allocation, etc. Calls upon various device drivers for physical I/O. Knows nothing about tracks or sectors; works with 32-bit "logical" block numbers.

SCF — I/O control of sequential character-oriented devices like CRTs, which are, to the user programs, files that may be read or written exactly as RBF (disk) files (except for lack of random access). Knows nothing of the device characteristics, leaving that to a subordinate driver such as ACIA.

DC3 and *WD1000* — Device drivers for specific hardware. Converts RBF's logical block number to track, sector, cylinder, or whatever is appropriate for the device. Talks to the device's I/O registers using either polled or interrupt-driven methods. Knows nothing of file structures. Declares that RBF manages I/O for the driver.

D0 — A device (descriptor) module for a mini-floppy. Supplies details of device; e.g., I/O register locations, number of tracks, sectors per track, drive number, seek time, double/single density/sided, etc. Contains data only, no code. D0 is the name programs use to refer to the peripheral for I/O activities on files on that device. This module states that its device is handled by the driver named DC3.

H0 — Identical to D0, except values unique to the hard disk. Defines WD1000 as the driver for device H0.

ACIA — Driver for RS-232 UART peripherals. Handles the characters from/to serial channels. Buffers incoming data for type-ahead. Buffers outgoing data to allow calling program to get on with concurrent work. Allows editing of typos, recall of last line entered, etc. Using device modules, it adapts to the terminal for upper/lower case, auto-line feed, nulls, etc. Declares that SCF manages the I/O for the driver.

TERM and *T1* — Like D0 and H0, these device descriptor modules define the driver name for the TERM and T1 peripherals (CRTs) as ACIA, and the terminals' unique needs.

To perform I/O to a certain device, a program (in some language) says in effect: "READ from D0" or "WRITE to TERM" or whatever. If, for example, the desired device is H0 (the hard disk) the KERNAL catches the program's request and, since it is I/O, calls upon IOMAN. This module then looks for the device name (H0) in the module directory and finds the name of the driver (WD1000) within the H0 module. Within the WD1000 module is the name of the driver, RBF. Then IOMAN merely sends the programs I/O desires to the I/O chief, RBF, along with the addresses of the driver (WD1000) and device (H0) modules. From here, RBF takes care of the rest, with help from the driver WD1000.

Although it's not obvious, this maze-running has one simple advantage. The user's program requested I/O to some device H0, and OS-9 figured out that H0 with WD1000 and RBF could do the job. The beauty of the scheme is that the system handled the device I/O despite the fact that the device "H0" was unknown at the time the main system was written, assembled, and configured (and no "hacking and patching" was done). Indeed, WD1000 and H0 were merely loaded into memory just after booting from the floppy. The same is true for T1, the second CRT and, though not shown, for a printer attached as device "P", managed by ACIA and SCF. Clearly, more printers, say P1, or more CRTs, say T2, or more disks, say D1 or H1 or X8580 may be added without affecting the rest of the system in any way whatsoever. Remember, these modules are just small pieces of data or code loaded from some disk into memory whenever a peripheral is added. Indeed, the printer module P is loaded only when that printer is being used!

This concludes part 1. *See the July issue of MICRO for part 2.*

Steve Childress has been involved with mini and micro systems for over 15 years. He developed a discrete logic microcomputer using shift-register memory and an IBM output writer for I/O in the days of the $300 8008 chip. Recently, he has contributed to the Apple II adaptation of the OS-9 system. You may contact him at 31220 La Baya Dr., Suite 110, Westlake Village, CA 91362.

Operating System Commands

Color Computer DOS

Disk Operating Commands :

Disk directory index	DIR <dn>
File copy	COPY <file spec1> TO <file spec2>
Disk copy	BACKUP <d1> [TO <d2>]
Assign I/O buffers	OPEN "<mode>",#<buff>,"<file spec>", <reclen>
Delete disk file	KILL "<file spec>"
Execute binary file	EXEC (File must be in memory)
Load file into memory	LOADM "<file spec>"[,<offsetadr>]
Merge file into memory file	MERGE "<file spec>"[,R]
Take input from disk file	DSKI$ <dn>,<trk>,<sec>,<str1>, <str2>
Output disk file in ASCII	DSKO$ <dn>,<trk>,<sec>,<str1>, <str2>
Format disk	DSKINI<dn>
Rename disk file	RENAME "<file spec1>" TO "<file spec2>"
Close open files	UNLOAD [dn]
Save memory to disk file	SAVEM "<file spec>",<strtadr>,<endadr>, <execadr>
Error-check on disk writes	VERIFY [ON or OFF]
Select default drive	DRIVE <dn>

BASIC Commands :

Open new sequential file	OPEN "I",#<buff>,<file spec>,<reclen>
Open old sequential file	Same
Open random array file	OPEN "D",#<buff>,<file spec>,<reclen>
Close file channel(s)	CLOSE #<buff>[, <buff# list>]
Delete disk file	KILL "<file spec>"
Rename disk file	RENAME "<file spec1>" TO "<file spec2>"
Display free space on disk	PRINT FREE(<dn>)
Write to sequential file	PRINT #<buff>;<data> WRITE #<buff>;<data>
Read from sequential file	INPUT #<buff>;<data> LINE INPUT #<buff>;<data>
Write to random record file	PUT #<buff>[,<recno>]
Read from random record file	GET #<buff>[,<recno>]
Save a program on disk	SAVE "<file spec>"[,A]
Load a program from disk	LOAD "<file spec>"
Execute a BASIC program from disk	RUN "<file spec>"[,R] LOAD "<file spec>",R MERGE "<file spec>",R

OS—65D

Disk Operating Commands :

Disk sector directory	DIR <tt> (tt=00-76)
List disk directory	RUN "DIR"
Create new file	RUN "CREATE"
Delete disk file	RUN "DELETE"
Rename disk file	RUN "RENAME"
Copy disk or disk files	RUN "COPIER"
Copy data files	RUN "DATRAN"
Compare disks or files	RUN "COMPAR"
Sort records in a disk file	RUN "GOSORT"
Pack files to front of disk	RUN "REPACK"
Fill a file with nulls	RUN "ZERO"
File copy	LOAD <file spec1>, PUT <file spec2>
Assign I/O buffers	RUN "CHANGE"
Load file	LOAD <file spec>
Execute binary file	CA <adr>=<trk>,<sec>,GO <adr>
Execute binary file in BASIC workspace	XQT <file spec>
Load file into memory	CA <adr>=<trk>,<sec>
Load track into memory	EXAM <adr>=<tt>
Take input from disk file	INPUT #<6 or 7>
Format disk	INIT
Format single track	INIT <tt>
Send output to disk file	PRINT #<6 or 7>
Save memory to disk file	SA <trk>,<sec>=>adr>/<sec>=<adr> /<pages>
Save BASIC Workspace	PUT <file spec>
Error-check on disk writes	Default
Select disk drive	SE <dev>
Direct output to I/O channel	IO, <channel no>
Direct input to I/O channel	IO <channel no>

BASIC Commands :

Open new sequential file	DISK OPEN,<dev>,<file spec>
Open old sequential file	Same
Open random array file	DISK OPEN,6,<file spec>
Close file channel	DISK CLOSE <6 or 7>
Append disk files	DISK!"LO <file spec1>", LIST[,] DISK!"LO <file spec2>",CTRL-X
Write to sequential file	PRINT #<dev>,<data>
Read from sequential file	INPUT #<dev>,<var>
Find string in seq'tial file	FIND "<string>"
Write to random record file	DISK GET <rec>, PRINT #6, <var>
Read from random record file	DISK GET <rec>, INPUT #6, <var>
Save a program on disk	DISK!"PUT <file spec>"
Load a program from disk	DISK!"LOAD <file spec>"
Send control to new program	RUN "<file spec>"
Execute a BASIC program from disk	RUN "<file spec>"

APPLE DOS

Disk Operating Commands :

Disk directory index	CATALOG
File copy	BRUN FID
Disk copy	RUN COPYA
Assign I/O buffers	MAXFILES <x> (default 3)
Delete disk file	DELETE <file spec>
Execute disk-input command	EXEC <file spec>
Execute binary file	BRUN <file spec>[,A[$]<adr>]
Load file into memory	BLOAD <file spec>[,A[$]<adr>]
Format disk	INIT <file spec>
Protect disk files	LOCK <file spec>
Unprotect disk files	UNLOCK <file spec>
Rename disk file	RENAME <file spec>
Save memory to disk file	BSAVE <file spec>,A[$]<adr>,L[$] <len>
Error-check on disk writes	VERIFY <file spec>
Monitor I/O channel	MON [C,I,O]
Turn off Monitor	NOMON [C,I,O]
Direct output to I/O channel	PR#<x>
Direct input to I/O channel	IN#<x>

BASIC Commands :

Open new sequential file	PRINT CHR$(4)"OPEN <file spec>"
Open old sequential file	Same
Open random record file	PRINT CHR$(4)"OPEN <file spec>,L<len>"
Close file channel	PRINT CHR$(4)"CLOSE [<file spec>]"
Send control to new program	PRINT CHR$(4)"RUN <file spec> "
Send control to new program saving variables	PRINT CHR$(4)"BLOAD CHAIN,A520" CALL 520"<file spec>"
Delete disk file	PRINT CHR$(4)"DELETE <file spec>"
Rename disk file	PRINT CHR$(4)"RENAME <file spec1>, <file spec 2>"
Write to sequential file	PRINT CHR$(4)"WRITE <file spec>" PRINT <data>
Write to the end of sequential file	PRINT CHR$(4)"APPEND <file spec>" PRINT <data>
Read from sequential file	PRINT CHR$(4)"READ <file spec>" INPUT [or GET] <data>
Read from a specific point in a sequential file	PRINT CHR$(4)"POSITION <file spec>,R<x>"
Write to random record file	PRINT CHR$(4)"WRITE <file spec>,R<x>" PRINT <data>
Read from random record file	PRINT CHR$(4)"READ <file spec>,R<x>" INPUT [or GET] <data>
Save a program on disk	SAVE <file spec>
Load a program from disk	LOAD <file spec>
Execute a BASIC program from disk	RUN <file spec>

(Continued)

Operating System Commands

6809 FLEX

Disk Operating Commands :

Assign system & work drives	ASN [,W=<drv>][,S=<drv>]
Append disk files	APPEND <file spec>[,<file list>], <file spec>
Create new text file	BUILD <file spec>
Disk directory index	CAT [<drv list>][,<match list>]
File copy	COPY <file spec>,<file spec> COPY <file spec>,<drv> COPY <drv>,<drv>[,<match list>]
Delete disk file	DELETE <file spec>[,<file list>]
Execute disk-input command	EXEC <file spec>
Execute binary file	<file spec>[.CMD] (default)
Load file into memory	GET <file spec>[,<file list>]
Take input from disk file	I,<file spec>,<command>
Output disk file	LIST <file spec>[,<line range>] [,+N(or P)]
Format disk	NEWDISK <drv>
Send output to disk file	O,<file spec>,<command>
Protect disk files	PROT <file spec>[,(opts)]
Rename disk file	RENAME <file spec1>,<file spec2>
Save memory to disk file	SAVE <file spec>,<begadr>, <endadr>[,<transadr>]
Error-check on disk writes	VERIFY [,ON (or OFF)]
Check disk file version	VERSION <file spec>
Direct output to printer	P,<command>
Install new boot program	LINK <file spec>

BASIC Commands :

Open new sequential file	OPEN NEW "<file spec>" AS <I/O Channel>
Open old sequential file	OPEN OLD "<file spec>" AS <I/O Channel>
Open random array file	OPEN [NEW or OLD] "<file spec>" AS <I/O Channel> DIM #<I/O Channel>,v[$] (<x>)[=<length>]
Open random record file	OPEN "<file spec>" AS <I/O Channel> FIELD #<I/O channel>,<len> AS <v$> [,<fieldlst>]
Close file channel	CLOSE <I/O channel>
Send control to new program	CHAIN "<file spec> 'x(1{""(default)
Delete disk file	KILL "<file spec>"
Rename disk file	RENAME "<file spec1>","<file spec2>"
Write to sequential file	PRINT #<I/O channel>[,USING <v$>],<data>
Read from sequential file	INPUT #<I/O channel>[,USING <v$>],<data>
Write to random record file	PUT #<I/O channel>[,RECORD <x>]
Read from random record file	GET #<I/O channel>[,RECORD <x>]
Save a program on disk	SAVE "<file spec> [.BAS]" (default)
Load a program from disk	LOAD "<file spec> [.BAS]" (default)
Execute a BASIC proram from disk	RUN "<file spec> [.BAC]" (default)
Compile a Basic program	COMPILE "<file spec> [.BAC]" (default)

OS–9

Disk Operating Commands :

Append disk files	MERGE <path>[<path>]
Create new text file	BUILD <path>
Create a new directory	MAKDIR <path>
Disk directory index	DIR [e][<path>]
Module directory index	MDIR [e]
Change working data dir	CHD <pathlist>
Change working exec dir	CHX <pathlist>
File copy	COPY <path><path>
Delete disk file	DELETE <path>[<path>]
Execute binary file	EX <modname>[<modifiers>] [<parameters>]
Load file into memory	LOAD <path>
Output disk file in ASCII	LIST <path>[<path>]
Output disk file in Hex	DUMP <path<[<path>]
Format disk	FORMAT <devname>[<opts>]
Protect disk files	ATTR <path>[<opts>]
Unprotect disk files	Same
Rename disk file	RENAME <path><newname>
Save memory to disk file	SAVE <modname>[<modname>]
Error-check on disk writes	VERIFY [U]
Install current boot program	COBBLER /<devname>
Install new boot program	OS9GEN /<devname>
Echo input to output path	ECHO <text>
Free space remaining	FREE <devname>
Log user onto system	LOGIN
Abort process	KILL <procID>
Display procdures & status	PROCS [e]
Set process priorities	SETPR <procID><number>
Display memory free	MFREE [e]
Free memory module	UNLINK <modname>[<modname>]
Print errors in English	PRINTERR
Time share monitor	TSMON [<pathlist>]

BASIC Commands :

Open new file	CREATE #<I/O Channel>,<name> [:<access mode>]
Open old sequential file and old random file	OPEN #<I/O Channel>,<name> [:<access mode>]
Close file channel	CLOSE #<I/O Channel>[,<I/O Channel>]
Send control to new program	CHAIN <filename>
Delete disk file	DELETE <filename>
Rename disk file	RENAME <procname>,<newprocname>
Write to sequential file	WRITE #<I/O Channel>,<data>
Read from sequential file	READ #<I/O Channel>,<data>
Read from a specific point in a sequential file	SEEK #<I/O Channel>,<expr>
Write to random record file	SEEK #<I/O Channel>,<expr> PUT #<I/O Channel>,<data struc>
Read from random record file	SEEK #<I/O Channel>,<expr> GET #<I/O Channel>,<data struc>
Save a program on disk	SAVE [<procname>][,<procname>]
Load a program from disk	LOAD <pathlist>
Execute a BASIC program from disk	RUN [<procname>][<erpr>[,<expr>]]
Compile a BASIC program	PACK [<procname>][,<procname>]

ORCA/M

Disk Operating Commands :

Append disk files	APPEND <file spec>
Create new text file	NEW
Edit old file	EDIT
Disk directory index	CATALOG
Disk sector read	PEEK
File copy	COPY
Delete disk file	DELETE <file spec>
Execute binary file	BRUN <file spec>
Load file into memory	LOAD <file spec>
Protect disk files	LOCK <file spec>
Unprotect disk files	UNLOCK <file spec>
Rename disk file	RENAME <file spec1>,<file spec2>
Save memory to disk file	SAVE [<file spec>]
Select disk drive	any command [,S<s>,D<d>]
Select default drive	(default)
Select volume	any command [,V<v>]
Free space remaining	(default on CATALOG)
Abort process	ESCape
Display memory free	FREE
Check disk file version	EXPAND
Reset version number	RESET
Print file in memory	PRINT
Set printer left margin	MARGIN <x>
Boot new disk	DOS [<s>]
Compile file	COMPILE [<opts>]
Assemble file	ASSEMBLE [<opts>]
Disk bad sector check	CHECK
Alphabetize catalog	COMPRESS A
Compress catalog	COMPRESS C
Set disk volume number	VOLUME <v>
Restore deleted file	RESTORE <file spec>
Assemble, Link and Execute	RUN <file spec>
Change catalog order	SWITCH <file spec1>,<file spec2>
Set tab stops	TAB
Print current time	TIME

(Continued)

Operating System Commands

Pascal

Disk Operating Commands :

Create new text file	E(dit
Disk directory index	F(iler L(ist or E(xtended list
Volume directory index	F(iler V(olumes
Zero directory index	F(iler Z(ero dir
File copy	F(iler T(ransfer
Disk copy	F(iler T(ransfer
Delete disk file	F(iler R(emove
Make EXEC file	M(ake exec
Delete disk file	F(iler D(elete
Rename disk file	F(iler C(hange
Pack files to front of disk	F(iler K(runch
Format disk	X(ecute FORMATTER
Save a program on disk	F(iler S(ave
Load a program from disk	F(iler G(et
Execute a PASCAL program from disk	X(ecute
Abort process	CTRL-@
Check for bad disk blocks	F)iler B)ad block scan
Compile a PASCAL program	C(ompile

Pascal Commands :

Open new file	REWRITE(<id>,<file spec>)
Open old file	RESET(<id>[,<file spec>])
Close file channel(s)	CLOSE(<id>[,<opts>])
Write to sequential file	WRITE([<id>,]<data>) WRITELN([<id>,]<data>)
Read from sequential file	READLN([<id>,]<data>)
Write to random record file	SEEK(<id>,<r>) PUT(<id>)
Read from random record file	SEEK(<id>,<r>) GET(<id>)
Load block into memory	UNITREAD(<v>,<array>,<l>[,<block>,<mode>])
Save block to disk	UNITWRITE(<v>,<array>,<l>[,<block>,<mode>])

PET BASIC 4 and *BASIC 1 and 2

Disk Operating Commands :

Append disk files	CONCAT [D<drive>,]"<file spec>"TO[D<drive>,] "<file >"[ON U<dev>]
Disk directory index	DIR[ECTORY][D<drive>][ON U<dev>] *LOAD "$<drive>[:<file spec>]",[<dev>]
File copy	COPY [D<drive>]["<file spec1>"]TO[D<drive>] ["<file spec2>"][ON U<dev>]
Disk copy	BACKUP D<drive> TO D<drive>[ON U<dev>] *PRINT#<I/O Channel>,"D[UPLICATE]<drive>=<drive>"
Delete disk file	SCRATCH [D<drive>],"<file spec>"[ON U<dev>] *PRINT#<I/O Channel>,"S<drive>:<file spec> [<drive>:<file spec>"]
Load file into memory	DLOAD "<file spec>"[,D<drive>][ON U<dev>]
Format disk	HEADER "<disk name>",D<drive>,I<v> *PRINT#<I/O Channel>,"N<drive>:<disk name>,<v>"
Rename disk file	RENAME [D<drive>,]"<file spec1>"TO"<file spec2>" [ON U<dev>]
Close open files	DCLOSE [#<I/O Channel>][ON U<dev>] *CLOSE [<I/O Channel>]
Free space remaining	Default on DIRECTORY
Display procedures & status	PRINT DS$,DS,ST
Direct output to I/O Channel	OPEN <I/O Channel>,<dev>:CMD<I/O Channel>
Update block map	COLLECT [D<drive>][ON U<dev>] *PRINT#<I/O Channel>,"V[ALIDATE][D<drive>]
Initialize disk	PRINT#<I/O Channel>,"I[NITIALIZE][D<drive>]

BASIC Commands :

Open new sequential file	DOPEN#<I/O Channel>,"<file spec>"[,D<drive>] [ON U<dev>][,<access>] *OPEN<I/O Channel>,<dev>,<sa>"<drive>: <file spec>SEQ[,<access>]
Open old sequential file	Same
Open relative record file	DOPEN#<I/O Channel>,"<file spec>",L<reclen> [,D<drive>]
Close file channel(s)	DCLOSE[#<I/O Channel>][ON U<dev>] *CLOSE [<I/O Channel>]
Send control to new program	DLOAD "<file spec>"[,D<drive>][ON U<dev>]
List disk directory	DIR[ECTORY][D<drive>][ON U<dev>]
Delete disk file	SCRATCH [D<drive>],"<file spec>"[ON U<dev>]
Rename disk file	RENAME [D<drive>],"<file spec1>"TO"<file spec2>" [ON U<dev>]
Copy disk or disk files	COPY [D<drive>,]["<file spec1>"]TO[D<drive>,] ["<file spec2>"][ON U<dev>]
Append disk files	APPEND#<I/O Channel>,"<file spec>"[D<drive>] [ON U<dev>] *OPEN<I/O Channel>,<dev>,<sa>"<drive>: <file spec>,A"
Write to sequential file	PRINT#<I/O Channel>,<data>
Read from sequential file	INPUT#<I/O Channel>,<data>
Write to relative rec file	RECORD#<I/O Channel>,<recno>[,<byteno>] PRINT#<I/O Channel>,<data>
Read from relative rec file	RECORD#<I/O Channel>,<recno>[,<byteno>] INPUT#<I/O Channel>,<data>
Save a program on disk	DSAVE "<file spec>"[,D<drive>][ON U<dev>] SAVE "[<drive>:]<file spec>",<dev>
Load a program from disk	DLOAD "<file spec>"[,D<drive>][ON U<dev>] LOAD ["<drive>:]<file spec>",<dev>
Load block into memory	PRINT#<I/O Channel>,"BLOCK-READ:<sa>,<drive>, <trk>,<sec>"
Save block to disk	PRINT#<I/O Channel>,"BLOCK-WRITE:<sa>,<drive>, <trk>,<sec>"

Calibration by Least Squares Polynomials on the Atari

by Mike Dougherty

Homebrew computer sensors are often plagued by calibration problems. The following program allows a set of calibration data points to be fitted with a least squares polynomial, allowing for efficient and compact interpolation of data.

A common problem encountered while building remote sensors for the personal computer is the calibration of these homebrew sensors. In a few cases, the calibration can be computed by a mathematical analysis of the hardware, often tedious and difficult. An easier approach is to take a set of calibration data points and fit a "best" curve through this data. Usually the functional form of the curve is known a priori from the hardware being used and only the parameters of the curve need be determined. LSQPOLY is an Atari 800 BASIC program designed to take a set of calibration data points, perform a polynomial least squares regression upon the calibration data, and visually plot the results. The output of LSQPOLY consists of a set of polynomial coefficients, $COEF_1$, $COEF_2$, ... $COEF_{m+1}$ where m is the highest degree of the polynomial. A point, V, within the range of the interpolation is computed by the polynomial evaluation:

$$F(V) = COEF_1 + COEF_2 \times V + COEF_3 \times V^2 + \ldots + COEF_{m+1} V^m$$

or

$$F(V) = \sum_{i=1}^{m+1} COEF_i \times V_{i-1}$$

The numerical methods used in LSQPOLY have been adapted from *Numerical Methods with Fortran Case Studies* by W.S. Dorn and D.D. McCracken and may be found in most texts on numerical analysis. While the methods used may fit a polynomial of any degree to the calibration data, I chose to limit LSQPOLY to polynomials of the fifth degree or less. From my experience with polynomial approximation, the higher order polynomials fit the calibration data better by "wiggling," instead of finding a

```
1000 REM --------    LSQPOLY    ----------
1001 REM
1002 REM .    by Mike Dougherty
1003 REM
1004 REM
1005 REM FIT A LEAST SQUARE POLYNOMIAL
1006 REM UP TO ORDER 5 THROUGH A SET
1007 REM OF CALIBRATION DATA POINTS.
1008 REM ALLOW THE USER TO GRAPHICALLY
1009 REM JUDGE THE RESULTING LSQ FIT.
1010 REM
1011 REM -----------------------------
1012 REM
1013 REM
1050 DIM X(50),Y(50),OPTION$(1),PAUSE$(1)
1060 DIM SUM(10),RIGHT(6),MATRIX(6,7)
1070 DIM COEF(6),YFIT(159)
1090 REM
1091 REM
1092 REM -----------------------------
1093 REM
1094 REM -- PRESENT THE LSQPOLY USER
1095 REM -- OPTIONS VIA MENU FORMAT.
1096 REM
1097 REM -----------------------------
1098 REM
1099 REM
1100 FOR FOREVER=0 TO 1 STEP 0
1110 GRAPHICS 0
1120 POSITION 3,5:PRINT "Select Option"
1130 POSITION 5,7:PRINT "D - Enter calibration Data"
1140 POSITION 5,8:PRINT "S - Show calibration data"
1150 POSITION 5,9:PRINT "R - Regression up to order 5"
1160 POSITION 5,10:PRINT "C - Print Coefficients"
1170 POSITION 5,11:PRINT "I - Interpolate Y values"
1180 POSITION 5,12:PRINT "G - Generate polynomial plot data"
1190 POSITION 5,13:PRINT "P - Plot polynomial data"
1200 POSITION 18,5
1210 INPUT OPTION$
1310 IF OPTION$="D" THEN GOSUB 2000
1320 IF OPTION$="S" THEN GOSUB 3000
1330 IF OPTION$="R" THEN GOSUB 4000
1340 IF OPTION$="C" THEN GOSUB 5000
1350 IF OPTION$="P" THEN GOSUB 6000
1360 IF OPTION$="I" THEN GOSUB 7000
1370 IF OPTION$="G" THEN GOSUB 8000
1390 NEXT FOREVER
1500 REM
1501 REM
1502 REM -----------------------------
1503 REM
1504 REM EACH OPTION IS HANDLED AS A
1505 REM SEPARATE SUBROUTINE, EACH
1506 REM STARTING ON AN EVEN THOUSAND
1507 REM LINE NUMBER.
1508 REM
1509 REM -----------------------------
1510 REM
1511 REM
1512 REM
2000 REM
2001 REM -- ENTER THE CALIBRATION DATA.
2002 REM
2110 GRAPHICS 0
2120 PRINT "Number of data points ";
2130 INPUT N
2210 FOR NUMBER=1 TO N
2220 PRINT "X(";NUMBER;") ";:INPUT VALUE:X(NUMBER)=VALUE
2230 PRINT "Y(";NUMBER;") ";:INPUT VALUE:Y(NUMBER)=VALUE
2240 PRINT
2250 NEXT NUMBER
2260 RETURN
3000 REM
3001 REM -- SHOW THE CALIBRATION DATA
3002 REM -- FOR VERIFICATION.
3003 REM
3100 GRAPHICS 0:POKE 752,1
```

(continued)

single smooth curve. As a rule of thumb, I choose the lowest order polynomial that gives a uniformly smooth curve reasonably close to the calibration data.

Instead of computing a numerical measure of error, LSQPOLY allows the user to visually compare the raw calibration data to data generated from the polynomial. In my opinion, this visual comparison allows a more meaningful evaluation of the least squares fit. The object is not to see how close the curve can be bent to pass near each calibration datum, but rather to pick a smooth "best" curve which will represent the functional relationship of the physical quantity being measured.

As a word of caution, LSQPOLY should be used to interpolate only within the range of the calibration data — do not try to extrapolate outside of the calibration data range. When using high order polynomials, a smooth monotonic curve within the limits of the calibration data can rapidly change direction outside of that range. In practice, the calibration data should include points at the extremes of the sensor range to properly "nail down" the curve.

Numerical Methods

A full discussion of polynomial regression may be found in the Dorn and McCracken text previously cited. As a summary, minimizing the sum of the square of the *Y* deviation yields the following matrix equation for a fit of *n* data points by a polynomial of order *m:*

[MATRIX] × [COEF] = [RIGHT]
where

$$\text{MATRIX}_{ij} = \begin{cases} n \text{ for } i=j=1 \\ \sum_{k=1}^{n} X_k^{\,i+j-2} \end{cases}$$

$$\text{RIGHT } i = \sum_{k=1}^{n} X_k^{\,i-1} \times Y_k$$

Note: LSQPOLY uses the FORTRAN convention of beginning subscripts with 1.

LSQPOLY solves the above matrix equation for the coefficients, COEF, by Gaussian Elimination.

The resulting coefficients, $COEF_1$, ..., $COEF_{m+1}$ are used to interpolate the functional value of any point within the calibration data range. The polynomial total of a specific abscissa value is:

$$\text{TOTAL} = \sum_{i=1}^{m+1} \text{COEF}_i \times \text{VALUE}^{i-1}$$

This may be evaluated in BASIC by the following methods:

```
TOTAL = COEF(1)
FOR I=2 TO M+1
TOTAL = TOTAL + COEF(I) ×
(VALUE ↑(I-1))
NEXT I
```

— or —

```
TOTAL = 0
FOR I = M+1 TO 1 STEP -1
TOTAL = TOTAL × VALUE +
COEF(I)
NEXT I
```

The second method, requiring no exponentiation, is Horner's method of polynomial evaluation. This method is particularly suited to small computer use.

As a rule, polynomial regression should be applied to the variable without error. That is, if X_i is an error free value, but Y_i contains error due to measurement, then the regression should express Y in terms of X:

$$Y = \text{COEF}_1 + \text{COEF}_2 \times X + \ldots + \text{COEF}_{m+1} \times X^m$$

This regression allows for the interpolated value of Y to be computed, given any value of X. However, for $M > 1$, the interpolated value of X cannot be easily computed, given a value of Y. In this case, the polynomial regression must be applied on the Y values, even though they contain measurement errors:

$$X = \text{COEF}_1 + \text{COEF}_2 \times Y + \ldots + \text{COEF}_{m+1} \times Y^m$$

If the visual fit is reasonable then the regression should pose no serious problem.

Finally, note that the matrix formed by this method may be quite ill-conditioned and subject to severe numerical errors. Such errors are easily detected by the visual comparison of the fit data and the calibration data. Thus far, no numerical difficulties have been encountered through ordinary use of LSQPOLY.

```
3110 FOR NUMBER=1 TO N
3120 PRINT "  X(";NUMBER;"): ";X(NUMBER),
3122 PRINT "Y(";NUMBER;"): ";Y(NUMBER)
3130 NEXT NUMBER
3140 PRINT :PRINT "Press RETURN to continue";
3150 INPUT PAUSE$
3160 RETURN
4000 REM
4001 REM -- PERFORM THE LSQ POLYNOMIAL
4002 REM -- REGRESSION ON THE DATA.
4003 REM
4004 REM -- REFER TO "NUMERICAL METHODS
4005 REM -- WITH FORTRAN CASE STUDIES"
4006 REM -- BY DORN & McCRACKEN
4007 REM
4110 GRAPHICS 0
4120 PRINT "Order of regression ";
4130 INPUT ORDER:IF ORDER>5 THEN RETURN
4150 FOR I=1 TO 2*ORDER
4160 SUM(I)=0
4170 NEXT I
4180 FOR I=1 TO ORDER+1
4190 RIGHT(I)=0
4200 NEXT I
4210 FOR POINT=1 TO N
4220 FOR I=1 TO ORDER*2
4230 SUM(I)=SUM(I)+X(POINT)^I
4240 NEXT I
4250 FOR I=1 TO ORDER+1
4255 IF I=1 THEN RIGHT(I)=RIGHT(I)+Y(POINT)
4260 IF I<>1 THEN RIGHT(I)=RIGHT(I)+Y(POINT)*(X(POINT)^(I-1))
4270 NEXT I
4280 NEXT POINT
4290 MATRIX(1,1)=N
4300 FOR I=1 TO ORDER+1
4310 MATRIX(I,ORDER+2)=RIGHT(I)
4320 FOR J=1 TO ORDER+1
4330 IF I+J<>2 THEN MATRIX(I,J)=SUM(I+J-2)
4340 NEXT J
4350 NEXT I
4410 FOR K=1 TO ORDER
4420 KP1=K+1
4430 L=K
4440 FOR I=KP1 TO ORDER+1
4450 IF ABS(MATRIX(I,K))>ABS(MATRIX(L,K)) THEN L=I
) THEN L=I
4460 NEXT I
4470 IF L=K THEN 4530
4480 FOR J=K TO ORDER+2
4490 TEMP=MATRIX(K,J)
4500 MATRIX(K,J)=MATRIX(L,J)
4510 MATRIX(L,J)=TEMP
4520 NEXT J
4530 FOR I=KP1 TO ORDER+1
4540 FACTOR=MATRIX(I,K)/MATRIX(K,K)
4550 FOR J=KP1 TO ORDER+2
4560 MATRIX(I,J)=MATRIX(I,J)-FACTOR*MATRIX(K,J)
4570 NEXT J
4580 NEXT I
4590 NEXT K
4600 COEF(ORDER+1)=MATRIX(ORDER+1,ORDER+2)/MATRIX(ORDER+1,
     ORDER+1)
4610 I=ORDER
4620 IP1=I+1
4630 TOTAL=0
4640 FOR J=IP1 TO ORDER+1
4650 TOTAL=TOTAL+MATRIX(I,J)*COEF(J)
4660 NEXT J
4670 COEF(I)=(MATRIX(I,ORDER+2)-TOTAL)/MATRIX(I,I)
4680 I=I-1
4690 IF I>=1 THEN 4620
4700 RETURN
5000 REM
5001 REM -- DISPLAY THE LSQ POLYNOMIAL
5002 REM -- COEFFICIENTS.
5003 REM
5110 GRAPHICS 0:POKE 752,1
5120 FOR NUMBER=1 TO ORDER+1
5130 PRINT "COEF(";NUMBER;"):",COEF(NUMBER)
```

(continued)

—Next Month in MICRO—

July Features Hardware!

A comparison of all the microcomputers you have questions about:

- Atari 400, 800, and 1200XL
- VIC-20
- Commodore 64
- Apple IIe
- TRS-80 Color Computer
- Commodore 128
- Sinclair/Timex
- TI 99

Also: An Apple Clock Interrupt
PET Goes ROM
Pascal Solitaire
UltraPainter on the Atari
Interfacing the Atari Joystick to the Color Computer

Variables in LSQPOLY

The following major variables and arrays are used in LSQPOLY.

N	The number of calibration data points.
X(50)	The abscissa values of the calibration data.
Y(50)	The ordinate values of the calibration data.
ORDER	The highest degree, m, of the polynomial.
MATRIX(6,7)	The intermediate matrix of X values.
RIGHT(6)	The intermediate vector of XY values.
SUM(10)	The intermediate vector of X values used to fill MATRIX for Gaussian Elimination.
COEF(6)	The polynomial coefficients.
XMIN	The minimum abscissa limit for plotting.
XMAX	The maximum abscissa limit for plotting.
YMIN	The minimum ordinate limit for plotting.
YMAX	The maximum ordinate limit for plotting.
XINC	The data frequency of the plot, i.e. every XINCth fit data point will be plotted.
YFIT(159)	The least squares interpolated values between XMIN and XMAX to be plotted.

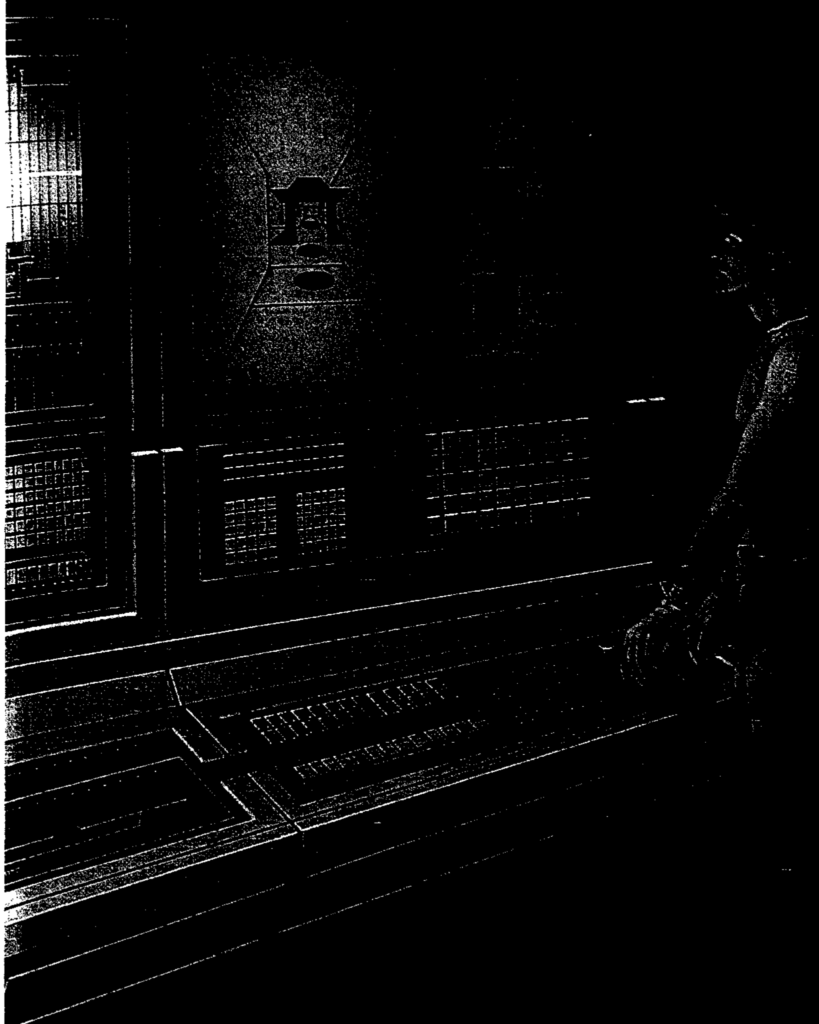
LSQPOLY allows the homebrew designer to visually analyze calibration data for trends and functional relationships. This tool is capable of generating a compact functional description of the data, allowing easy and accurate interpolation — even if the calibration data contains error.

Although the plot routines are specific to the Atari 800, LSQPOLY should require little modification to execute the numerical routines on other BASIC systems. Computer systems supporting graphics resolutions of 160 × 80 or higher (such as Atari GRAPHICS 7 and 8) will allow LSQPOLY to be used most effectively.

Mike Dougherty has an M.S. degree in computer science, and is currently working at Martin Marietta Aerospace in Denver, CO. You may contact him at 7659 W. Fremont Ave., Littleon, CO 80123.

```
5140 NEXT NUMBER
5150 PRINT :PRINT "Press RETURN to continue";
5160 INPUT PAUSE$
5170 RETURN
6000 REM
6001 REM -- PLOT THE GENERATED LSQ FIT.
6002 REM
6010 GRAPHICS 0
6020 PRINT "Minimum y ";:INPUT YMIN
6030 PRINT "Maximum y ";:INPUT YMAX
6040 PRINT "Data Frequency ";:INPUT XINC
6050 XRANGE=XMAX-XMIN
6060 YRANGE=YMAX-YMIN
6070 GRAPHICS 7
6080 SETCOLOR 2,0,0
6100 REM
6101 REM -- PLOT THE CALIBRATION DATA.
6102 REM
6110 COLOR 2
6120 FOR I=1 TO N
6130 IF (X(I)<XMIN) OR (X(I)>XMAX) THEN 6190
6140 IF (Y(I)<YMIN) OR (Y(I)>YMAX) THEN 6190
6150 XPLOT=INT(159*(X(I)-XMIN)/XRANGE)
6160 YPLOT=79-INT(79*(Y(I)-YMIN)/YRANGE)
6170 PLOT XPLOT,YPLOT
6190 NEXT I
6210 PRINT "Press RETURN to continue";
6220 INPUT PAUSE$
6230 REM
6231 REM -- PLOT THE LSQ FIT DATA.
6232 REM
6300 COLOR 1
6310 FOR I=0 TO 159 STEP XINC
6320 IF (YFIT(I)<YMIN) OR (YFIT(I)>YMAX) THEN 6390
6330 YPLOT=79-INT(79*(YFIT(I)-YMIN)/YRANGE)
6340 PLOT I,YPLOT
6390 NEXT I
6900 PRINT "Press RETURN to continue";
6910 INPUT PAUSE$
6920 RETURN
7000 REM
7001 REM -- FORM INTERPOLATION VALUES
7002 REM -- FROM THE LSQ POLYNOMIAL
7003 REM -- COEFFICIENTS.  TERMINATE
7004 REM -- WITH AN X VALUE OF -9999.
7005 REM
7110 GRAPHICS 0
7120 PRINT "X (-9999 to RETURN) ";
7130 INPUT VALUE
7140 IF VALUE=-9999 THEN RETURN
7150 GOSUB 7900
7180 PRINT "y:                    ";TOTAL:PRINT
7190 GOTO 7120
7900 REM
7901 REM -- COMPUTE THE POLYNOMIAL
7902 REM -- TOTAL FROM THE X VALUE.
7903 REM
7910 TOTAL=0
7920 FOR P=ORDER+1 TO 1 STEP -1
7930 TOTAL=TOTAL*VALUE+COEF(P)
7940 NEXT P
7950 RETURN
8000 REM
8001 REM -- GENERATE THE LSQ FIT DATA
8002 REM -- FOR THE VISUAL PLOT.
8003 REM
8010 GRAPHICS 0
8020 PRINT "Minimum x ";:INPUT XMIN
8030 PRINT "Maximum x ";:INPUT XMAX
8040 XINC=(XMAX-XMIN)/160
8050 GRAPHICS 0:POKE 752,1
8100 FOR XOFFSET=0 TO 159
8110 VALUE=XMIN+XOFFSET*XINC
8120 GOSUB 7900
8130 YFIT(XOFFSET)=TOTAL
8135 PRINT "    Y(";VALUE;")=",YFIT(XOFFSET)
8140 NEXT XOFFSET
8150 RETURN
```

MICRO™



MAROONED!

And you're the quarry for the Questers!

You're marooned in a derelict space station trapped between the stars. Waiting for rescue.

But, you may never make it. The deadly space Questers have located you and are ready to attack. Your first line of defense is to close the space ports on A Deck before you're overrun, then use the Teleportation chamber to head them off on the other decks.

As you navigate the lonely corridors . . . turning here, hiding there, attacking or retreating, the swarms of Questers get faster and smarter!

There's no other game like Spectre. Deck after deck, you'll find the most challenging and original 3-D maze action ever!

Get SPECTRE now, only $29.95 for the Apple II* at your computer store, or

8943 Fullbright Ave.,
Chatsworth, CA 91311 (213) 709-1202

VISA/MASTERCARD accepted. $2.00 shipping/handling charge. (California residents add 6½% sales tax)

*Apple II is a trademark of Apple Computer, Inc.

Pinewood Derby with Computer Timing

by Sidney S. Koegler

As anyone who has been a Cub Scout or the father of a Cub Scout can testify, the high point of the scouting year is the pack meeting at which the Pinewood Derby is held. This is a model car race in which 7-inch homemade wooden cars roll down an inclined track while 40 or so 8-to-10-year-old boys and their parents shout encouragement.

I quickly discovered how competitive the racing participants are when I helped judge a Pinewood Derby three years ago. Our Cub Scout pack's track is a four-lane affair on which four cars race simultaneously. It required two judges with keen eyesight and a superb sense of timing to determine the winner. The following year I installed an electronic judging and time-keeping system using my AIM 65 computer and four optosensors.

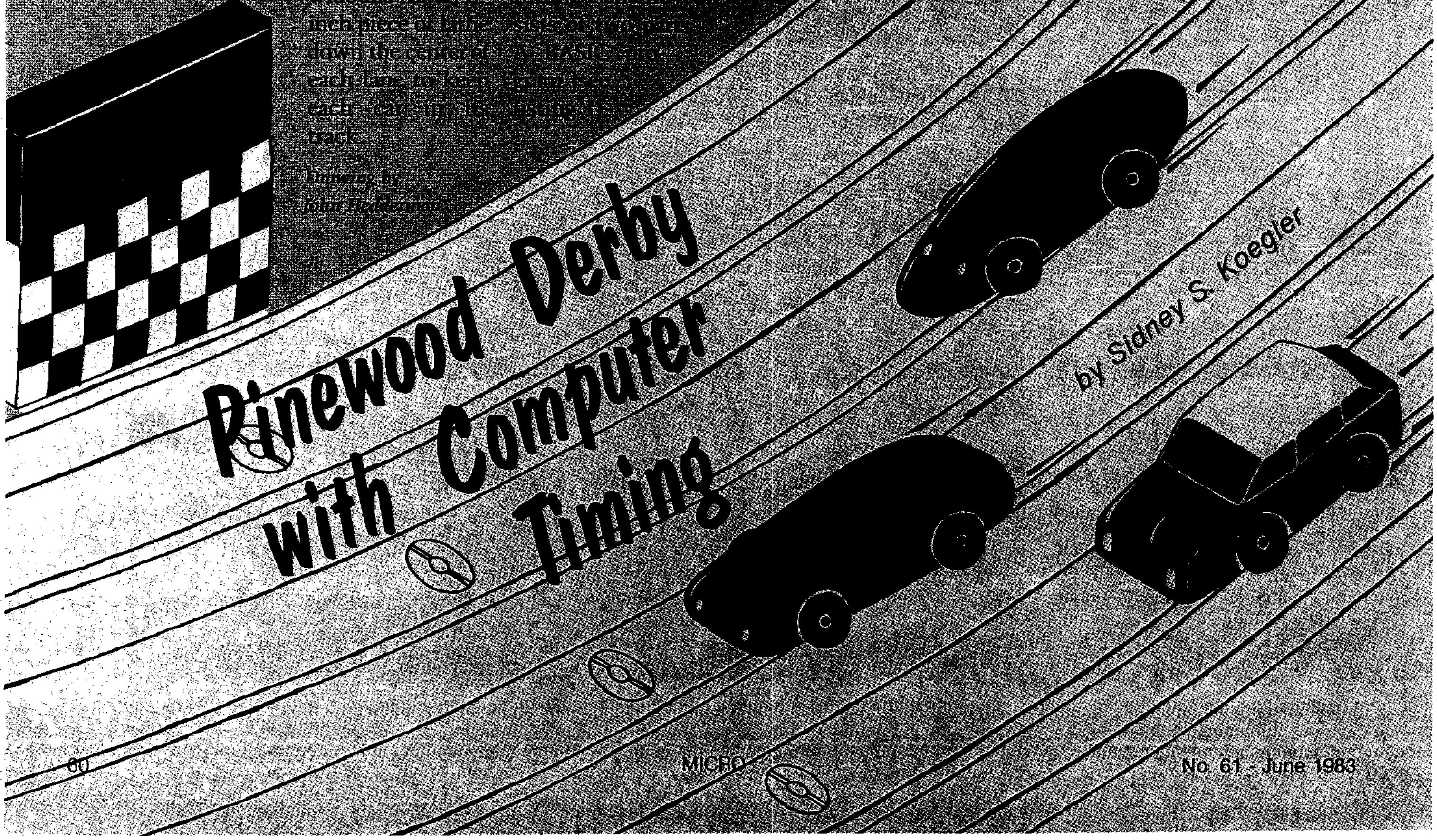
With the computer judging in place, we found the Pinewood Derby ran much smoother and faster, and there were fewer upset fathers and frazzled judges. The computer selects the first-, second-, third-, and fourth-place finishers and gives each car's time to 1/1000 of a second.

The Set-up

The Pinewood Derby track consists of two parts. The initial 8 feet of track are inclined at about 30 degrees to the horizontal. This is followed by 30 feet of horizontal run to the finish line. The cars cover the entire length of track in about 2.5 seconds (don't blink!). The track is 15 inches wide and has a 1/2-inch piece of lathe down the center of each lane to keep each car in its track.

[illegible]

Drawing by John Heldermann



Richvale Telecommunications

10610 BAYVIEW (Bayview Plaza)
RICHMOND HILL, ONTARIO, CANADA L4C 3N8
(416) 884-4165

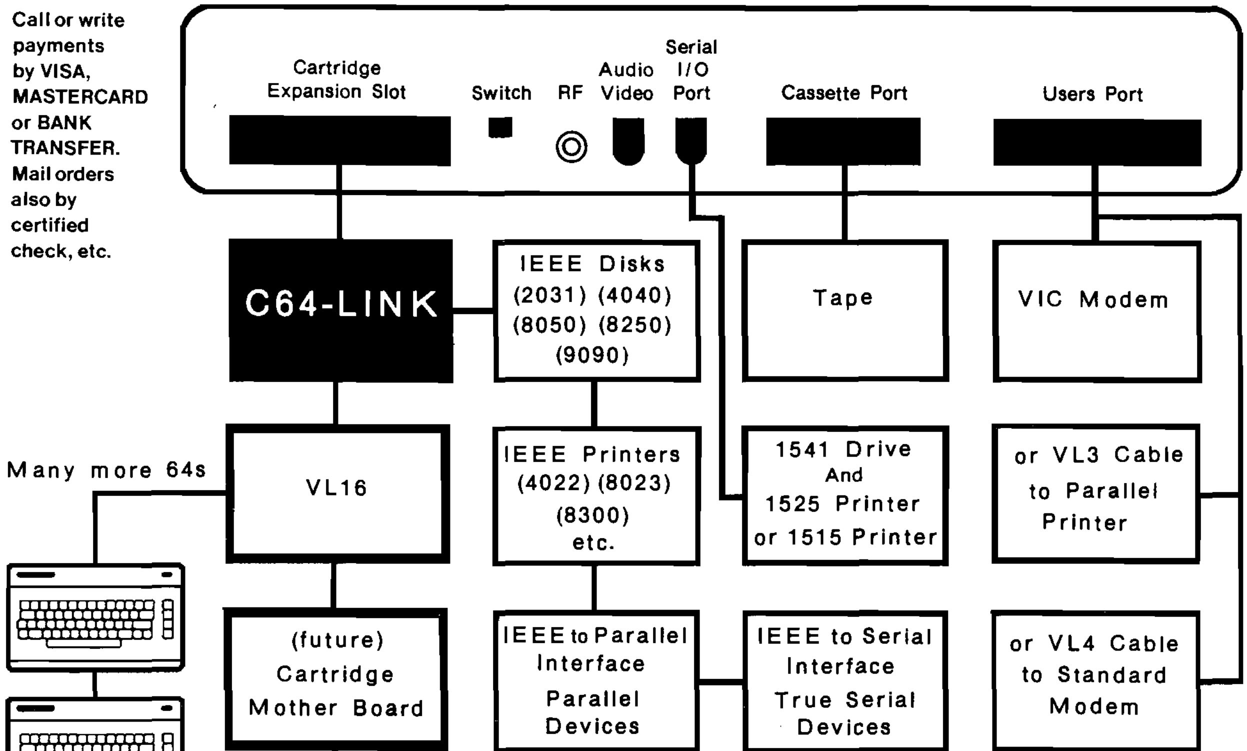
$185.00 Canadian
$149.00 U.S.
PLUS CUSTOMS BROKERAGE, HANDLING AND MAILING CHARGE.

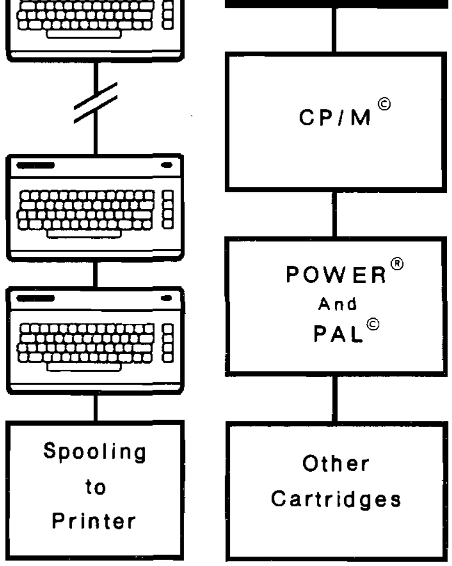
C64-LINK©
The Smart 64

Also available for VIC 20

RTC RTC

Call or write payments by VISA, MASTERCARD or BANK TRANSFER. Mail orders also by certified check, etc.





Give These Expanded Capabilities To Your 64 And VIC

★ The ability to transfer data from any type of device to another (IEEE, Serial, Parallel)

★ BASIC 4.0 Which allows you to run more PET BASIC programs and gives you extended disk and I/O commands.

★ The ability to have several 64s on line together — sharing common IEEE devices such as disks or printers with Spooling Capability.

★ Built-in machine language monitor

★ A built-in terminal or modem program which allows the system to communicate through a modem to many bulletin board systems and other computer mainframes.

★ Compatibility with CP/M.

Contact your local Commodore dealer or RTC.

Circle No. 41

has been counted previously.

3. If the car has not been counted before, the lane number and time are stored in a buffer.
4. Control is returned to the BASIC program when all four cars have crossed the finish line or if a key is depressed on the AIM 65 keyboard; otherwise the routine loops back to the polling step above.

The running time (from the start of the race) is accumulated using the 16-bit timer of the 6522 VIA chip. Each time the VIA times out (.065 second), the value of CLOCK is incremented. This gives about 16 seconds timer capacity before the timer rolls over. Three bytes are needed in the buffer to record the CLOCK value and the 16-bit timer value for each car or lane. This value is converted back to decimal in the BASIC program.

Conclusions

The electronic monitoring of a Pinewood Derby race is relatively easy and inexpensive for anyone owning an AIM 65 or similar microcomputer. This approach is software-oriented, since the only electronic components required are the optosensors. Although the AIM 65 was used here, with minor modifications the code could be used on the PET, KIM, or any 6502-based machine that has four or five free 6522 VIA ports.

Sydney S. Koegler is a chemical engineer specializing in pilot-plant design and operation. You may reach Mr. Koegler at 2339 Carriage Ave., Richland, WA 99352.

Listing 1: BASIC Program for Pinewood Derby

```
100 REM...PINEWOOD DERBY
110 REM...4-23-82
115 V=40961:REM...VIA PORT A
120 REM...DATA BUFFER  $FF0-$FFF
125 BS=4080:BT=BS+4:BU=BT+4:BV=BU+4
130 REM...SET POINTERS FOR SCAN ROUTINE
135 POKE 04,0:POKE 05,15
140 INPUT'ENTER HEAT NO. ',H$
160 INPUT'LANE NO. ';L
162 IF L=0 THEN 190
165 INPUT'CAR NO. ';C(L)
170 IF L<>4 THEN 160
190 PRINT'      READY'
195 S=PEEK(V) OR 16
200 GET K$
205 IF K$='' AND S<>0 THEN 195
210 PRINT'  T I M I N G'
215 XX=USR(0):REM...CALL SCAN ROUTINE
220 FOR I=1 TO 4
225 TM(I)=.065536*PEEK(BT+I)+.000256*
    (255-PEEK(BU+I))
226 TM(I)=TM(I)+(255-PEEK(BV+I)*1E-6
227 TM(I)=INT(1000*TM(I))/1000
230 M=PEEK(BS+I)
233 IF M<=0 THEN P(I)=0:GOTO 250
235 P(I)=INT(1.1+LOG(M)/LOG(2))
240 NEXT
250 PRINT!' ':PRINT!'HEAT #;H$
270 PRINT!' PLAC LANE CAR  TIME'
310 FORJ=1 TO 4
320 IF P(J)=0 THEN 350
330 PRINT!' ';J;'  ';P(J);' ';C(P(J))
    ;TM(J)
340 NEXT J
350 GOTO 140
```

Listing 2: Assembly Listing of Timing Routine

```
; . . . PIN21  4-23-82 . . .
;   DETERMINES 1,2,3,4 PLACE
;    IN PINEWOOD DERBY
;     MODIFIED FOR TIMER FCTN
;
==0000  VIA=$A000         ;VIA BASE ADDRESS
==0000  T2LO=VIA+8        ;TIMER LOW BYTE
==0000  T2HI=VIA+9        ;TIMER HIGH BYTE
==0000  IFR=VIA+$0D       ;INTERUPT FLAG
                          REGISTER
==0000  PLACE=$0FF0       ;BUFFER -
                          FINISHING PLACE
==0000  TIME1=$0FF4       ;BUFFER - TIME,
                          HIGH BYTE
==0000  TIME2=$0FF8       ;BUFFER - TIME,
                          MIDDLE BYTE
==0000  TIME3=$0FFC       ;BUFFER - TIME,
                          LOW BYTE
==0000  CLOCK=$0FEE       ;ACCUMULATED TIME
==0000  TEMP=$0FEF        ;TEMPORARY DATA
                          STORAGE
==0000  OUT=$C0D1         ;AIM BASIC
                          RE-ENTRY POINT
==0000  GETKEY=$E907      ;AIM MONITOR
                          KEYBOARD STROBE
==0000  *=$0F00
;
;   INITIALIZE VARIABLES
;
==0F00  START   A9 00       LDA  #0
                8D 03 A0    STA  VIA+3
                8D EE 0F    STA  CLOCK
                A2 04       LDX  #4
==0F0D  INIT    9D F0 0F    STA  PLACE,X
                9D F4 0F    STA  TIME1,X
```

(continued)

```
                9D F8 0F        STA  TIME2,X
                9D FC 0F        STA  TIME3,X
                CA              DEX
                D0 F1           BNE  INIT
                20 7B 0F        JSR  CLICK
;
;   SAMPLE VIA FOR PASSING CARS
;
==0F1F  SCAN    AD 01 A0        LDA  VIA+1
                8D EF 0F        STA  TEMP
;
;   TIMER TIMED OUT ?
;
                A9 20           LDA  #%00100000
                2C 0D A0        BIT  IFR
                F0 03           BEQ  KEY
                20 7B 0F        JSR  CLICK
;
;   CHECK FOR KEY DEPRESSED
;
==0F2F  KEY     20 07 E9        JSR  GETKEY
                C9 FF           CMP  #$FF
                F0 03           BEQ  CHECK
                4C D1 C0        JMP  OUT
;
;   CHECK EACH BIT FOR PASSING CARS
;
==0F39  CHECK   A9 01           LDA  #1
==0F3B  CHECK1  2C EF 0F        BIT  TEMP
                D0 09           BNE  STORE
==0F40  CHECK2  C9 08           CMP  #8
                F0 DB           BEQ  SCAN
                18              CLC
                2A              ROL  A
                4C 3B 0F        JMP  CHECK1
;
;   STORE LANE NUMBERS
;    IN THE CORRECT SEQUENCE
;
==0F49  STORE   A2 01           LDX  #1
```

(continued)

```
==0F4B  STORE1  48              PHA
                BD F0 0F        LDA  PLACE,X
                F0 11           BEQ  NEXT      ;NO CAR ?
                68              PLA
                E0 04           CPX  #4        ;LAST BIT ?
                D0 03           BNE  COMP
                4C D1 C0        JMP  OUT
;
;   SEE IF LANE # ALREADY STORED
;
==0F59  COMP    DD F0 0F        CMP  PLACE,X
                F0 E2           BEQ  CHECK2
                E8              INX
                4C 4B 0F        JMP  STORE1
;
;   STORE LANE # IN PLACE
;    STORE TIMER VALUES
;     GOTO NEXT BIT IN VIA
;
==0F62  NEXT    68              PLA
                9D F0 0F        STA  PLACE,X
                AD EE 0F        LDA  CLOCK
                9D F4 0F        STA  TIME1,X
                AD 09 A0        LDA  T2HI
                9D F8 0F        STA  TIME2,X
==0F72          AD 08 A0        LDA  T2LO
                9D FC 0F        STA  TIME3,X
                4C 1F 0F        JMP  SCAN
;
;       TIMER ROUTINE
;   (INCREMENTS 'CLOCK' EVERY .065 SEC)
;
==0F7B  CLICK   18              CLC
                EE EE 0F        INC  CLOCK
                B0 09           BCS  STOP
                A9 FF           LDA  #$FF
                8D 08 A0        STA  T2LO
                8D 09 A0        STA  T2HI
                60              RTS
==0F8A  STOP    00              BRK
```

MICRO™

Fractionated BASIC

A BASIC program to automate the calculation of fractions

by LeRoy Moyer

Microsoft BASIC is a good language for doing numerical calculations when you want the result of an arithmetic computation to be in decimal notation. Sometimes, however, when working with fractions you would like the result of a computation to be a fraction. Often it is hard to convert the decimal result that is generated by BASIC into the fraction that it represents. For example, 0.23076923 is not easily recognized as the fraction 3/13. The fractions program I describe does all of its computations using fractions. That is, numbers are input as fractions and are output as fractions that have been reduced to their lowest common denominator.

Included in the program is an interpreter for the evaluation of one-line equations. The syntax for all equations in this interpreter is: (Variable Name) = (Algebraic Expression). The algebraic expression may contain numerical fractions, previously defined variables, or the symbols (,), +, −, *, and /. Hierarchy for the mathematical operations is the same as in BASIC. An example of a valid expression is: VARIABLE = ¾ + X/ (2/3) when X as been defined previously. After this equation has been typed into the computer and return is pressed, the value of VARIABLE is calculated, stored in memory under that name, and the value is printed on the next line. Variable names can be any length and all characters are significant.

Interpretation of the equation is accomplished by using two stacks — one for operators (OP$) and one for numeric values of the fraction (ST and SB). When a number is encountered in the equation it is put on the number stack with the numerator in ST and the denominator in SB. If a variable name is encountered, then the variable is first looked up in the variable name table (VN$) before the numeric value associated with that variable is transferred to the number stack. Finally, when an operator is encountered the precedence is compared with the precedence of the operator on the top of the operator stack. If the precedence of the new operator is higher, then the operator is placed on the operator stack; otherwise the top operator on the stack is used and the operator stack is reduced by one. This continues until the end of the equation at which time all remaining operations on the stack are completed and the results are printed out.

The listing of the program is commented extensively, but none of the REM statements are used as references and so they can be eliminated to make the program shorter and faster. The program logically starts at line 1000, which initializes the arrays. An equation is input as a string (LN$) at line 1100. First, the defined variable (VS$) is separated out of LN$ in line 1150 by looking for a blank space or an equal (=) sign. The subroutine starting at line 1700 then checks to see if there is a unary minus sign. If a minus sign is found, the symbol] is put on the operator stack and used only inside the program. Any symbol will work but this particular symbol is not used commonly on the Apple, for which this program was originally written. From this point, the program steps through the rest of the line (LN$) checking to see if the next part of the equation is a variable (line 1240), a number (line 1250), or a symbol (line 1260). If it is one of these types of items, the program then goes to lines 1500, 1400, or 1300, respectively, for each type.

The section of the program starting at line 1300 handles the symbols in the equation. First it checks to see if the symbol was a (since this has the highest precedence. If it is, the program

(continued on page 67)

Listing 1: Applesoft Listing for Fractionated BASIC

```
10  GOTO 1000
95  REM   ADDITION/SUBTRACTION
100 T = T1 * B2 + T2 * B1:B = B1 * B2: GOTO 130
105  REM   MULTIPLICATION
110 T = T1 * T2:B = B1 * B2: GOTO 130
115  REM   DIVISION
120 T = T1 * B2:B = T2 * B1
125  REM   REDUCE THE FRACTION T/B TO LOWEST TERMS
130  IF T = 0 THEN B = 1: RETURN
140  IF B = 0 THEN  PRINT "DIVISION BY ZERO":
     POP: GOTO 1100
150 TA =  ABS (T):TB =  ABS (B): IF TA < TB THEN
    TM = TA:TA = TB:TB = TM
160 DV =  INT (TA / TB):R = TA - TB * DV
170  IF 0 = R THEN 190
180 TA = TB:TB = R: GOTO 160
190 T =  SGN (B) * T / TB:B =  ABS (B) / TB: RETURN
395  REM DO THE LAST OPERATION ON THE OPERATOR STACK
396  REM IF NO OPERATORS YET THEN PUT THE OPERATOR
           ON THE STACK
400  IF NP = 0 THEN NP = 1:OP$(NP) = TS$: RETURN
410  IF OP$(NP) = "<" THEN NP = NP + 1:OP$(NP) = TS$
     : RETURN
415  REM   CHECK FOR A UNARY MINUS AND DO IF ON
     OPERATOR STACK
420  IF OP$(NP) = "]" THEN ST(NS) =  - ST(NS):
     OP$(NP) = TS$: RETURN
425  REM   PULL TOP TWO NUMBERS OFF OF STACK FOR
      OPERATION
430 T1 = ST(NS - 1):B1 = SB(NS - 1):T2 = ST(NS):
    B2 = SB(NS)
435  REM   PICK THE PROPER SUBROUTINE TO GO TO
440  IF OP$(NP) = "*" THEN  GOSUB 110: GOTO 480
450  IF OP$(NP) = "/" THEN  GOSUB 120: GOTO 480
460  IF OP$(NP) = "+" THEN  GOSUB 100: GOTO 480
470 T2 =  - T2: GOSUB 100
475  REM STORE THE RESULT BACK ON THE NUMBER STACK
480 NS = NS - 1:ST(NS) = T:SB(NS) = B:OP$(NP) = TS$
     : RETURN
995  REM   START OF THE PROGRAM
1000  DIM VN$(200),VT(200),VB(200),
     OP$(50),ST(100),SB(100)
1010 NV = 0
1095  REM   READ IN A LINE TO BE EVALUATED
1100  INPUT "#";LN$
1105  REM CHECK TO SEE IF VARIABLES ARE TO BE PRINTED OUT
1110  IF  LEFT$ (LN$,1) = "?" THEN  FOR Z = 1 TO NV:
     PRINT VN$(Z);" = ";VT(Z);"/";VB(Z): NEXT Z: GOTO1100
1120 LG =  LEN (LN$):CN = 1:NP = 0:NS = 0
1125  REM SKIP LETTERS UNTIL A BLANK OR = SIGN ARE FOUND
1130  IF  MID$ (LN$,CN,1) = " " OR  MID$ (LN$,CN,1)
      = "=" THEN 1160
1140  IF CN < LG THEN CN = CN + 1: GOTO 1130
1150  PRINT "SYNTAX ERROR": GOTO 1100
1155  REM   THE VARIABLE BEING DEFINED IS VS$
1160 VS$ =  LEFT$ (LN$,CN - 1)
1170  IF  MID$ (LN$,CN,1) = "=" THEN 1200
1180  IF CN < LG AND  MID$ (LN$,CN,1) = " " THEN
      CN = CN + 1: GOTO 1170
1190  GOTO 1150
1195  REM   CHECK FOR A UNARY MINUS SIGN
1200  GOSUB 1700
1210 CN = CN + 1
1220  IF  MID$ (LN$,CN,1) = " " AND CN < LG THEN 1210
1230  IF CN > LG THEN 1600
1235  REM   FIND THE FIRST CHARACTER AFTER THE =
1240 TS$ =  MID$ (LN$,CN,1): IF TS$ =  > "A" AND
      TS$ <  = "Z" THEN 1500
1245  REM   IF A NUMBER - GOTO 1400 TO GET NUMBER
1250  IF TS$ <  = "9" AND TS$ =  > "0" THEN 1400
1255  REM   IF A SYMBOL - GOTO 1300 TO EVALUATE
1260  IF TS$ = "(" OR TS$ = ")" OR TS$ = "+" OR TS$
       = "-" OR TS$ = "*" OR TS$ = "/" THEN 1300
1270  GOTO 1150
1295  REM   THIS SECTION CHECKS THE PRECEDENCE OF
      THE OPERATORS
1296  REM   IF A ( - PUT IT ON THE STACK AND CHECK
      FOR A UNARY -
1300  IF TS$ = "(" THEN NP = NP + 1:OP$(NP) = "(":
      GOSUB 1700: GOTO 1210
1305  REM   CHECK FOR + OR -. IF FOUND DO THE PREVIOUS
      OPERATOR
1310  IF TS$ <  > "+" AND TS$ <  > "-" THEN 1330
1320  GOSUB 400: GOTO 1210
1325  REM   IF PRESENT OP IS * OR / AND PREVIOUS
      OP IS * OR / THEN DO
```

(continued)

```
1330  IF TS$ <  > ")" AND (OP$(NP) <  > "+" AND
      OP$(NP) <  > "-" THEN 1320
1335  REM   OPERATION IS NOT TO BE DONE SO PUT IT
      ON STACK
1340  IF TS$ <  > ")" THEN NP = NP + 1:OP$(NP) = TS$:
      GOTO 1210
1345  REM   WORK BACK TO NEXT ( TO COMPLETE A ()
      EXPRESSION
1350  IF OP$(NP) = "(" THEN NP = NP - 1: GOTO 1210
1360  GOSUB 400: IF NP > 0 THEN NP = NP - 1: GOTO 1350
1370  PRINT "STACK ERROR": GOTO 1100
1395  REM   EVALUATE A NUMBER
1396  REM   GET THE FIRST NUMBER
1400 T =  VAL ( MID$ (LN$,CN)):CN = CN +  LEN ( STR$(T))
1410  IF CN =  > LG THEN B = 1: GOTO 1450
1420  IF  MID$ (LN$,CN,1) = " " THEN CN = CN + 1:
      GOTO 1410
1425  REM   IF THE NEXT SYMBOL IS NOT A / THEN NOT
       A FRACTION
1430  IF  MID$ (LN$,CN,1) <  > "/" THEN B = 1:CN =
      CN - 1: GOTO 1450
1440 CN = CN + 1:B =  VAL ( MID$ (LN$,CN)):CN = CN
      +  LEN ( STR$ (B)): IF B = 0 THEN  PRINT "DIVISION
      BY 0": GOTO 1100
1445  REM   PUT NUMBER ON THE STACK
1450 NS = NS + 1:ST(NS) = T:SB(NS) = B: GOTO 1210
1495  REM   FIND THE NAME AND VALUES OF A VARIABLE
1500 CS = CN
1510 TS$ =  MID$ (LN$,CN,1)
1515  REM   LOOK FOR A CHARACTER THAT IS NOT IN THE
      VARIABLE
1520  IF TS$ = " " OR TS$ = ")" OR TS$ = "+" OR TS$
      = "-" OR TS$ = "*" OR TS$ = "/" THEN CN = CN - 1
      : GOTO 1540
1530  IF CN < LG THEN CN = CN + 1: GOTO 1510
1535  REM   FOUND THE NAME OF THE VARIABLE TS$
1540 TS$ =  MID$ (LN$,CS,CN - CS + 1):I = 1
1550  IF NV = 0 THEN 1580
1555  REM IF VARIABLE FOUND PUT ITS VALUE ON THE STACK
1560  IF TS$ = VN$(I) THEN NS = NS + 1:ST(NS) = VT(I):
      SB(NS) = VB(I): GOTO 1210
1570  IF I < NV THEN I = I + 1: GOTO 1560
1575  REM   IF VARIABLE NOT FOUND IN LIST QUIT
1580  PRINT "VARIABLE ";TS$;" NOT DEFINED": GOTO 1100
1595  REM   END OF EQUATION REACH - DO REST OF
      OPERATORS ON STACK
1600  IF NP > 0 AND OP$(NP) = "(" THEN NP = NP - 1
      : GOTO 1600
1610  IF NS > 1 AND NP > 0 THEN TS$ = "#":
      GOSUB 400:NP = NP - 1: GOTO 1600
1615  IF NP = 1 AND OP$(NP) = "]" THEN TS$ = "#":
      GOSUB 400: GOTO 1630
1620  IF NP <  > 0 THEN  PRINT "STACK DISJOINT":
      GOTO 1100
1625  REM   PRINT OUT THE RESULTS
1630  PRINT VS$;" = ";ST(1);: IF SB(1) <  > 1 THEN
      PRINT "/";SB(1);
1640  PRINT :I = 1
1645  REM   SEE IF VARIABLE HAS BEEN USED BEFORE
1646  REM   IF IT HAS PUT NEW VALUES IN
1650  IF NV = 0 THEN NV = 1:VN$(NV) = VS$:VT(NV) =
      ST(1):VB(NV) = SB(1): GOTO 1100
1660  IF VS$ = VN$(I) THEN VT(I) = ST(1):VB(I) = SB(1)
      : GOTO 1100
1670  IF I < NV THEN I = I + 1: GOTO 1660
1675  REM   IF VARIABLE NOT USED BEFORE DEFINE A
      NEW VARIABLE
1680 NV = NV + 1:VN$(NV) = VS$:VT(NV) = ST(1):VB(NV)
      = SB(1): GOTO 1100
1695  REM SEARCH FOR THE NEXT CHARACTER TO SEE IF -
1700 CN = CN + 1: IF  MID$ (LN$,CN,1) = " " THEN 1700
1710  IF  MID$ (LN$,CN,1) <  > "-" THEN CN = CN - 1 :
      RETURN
1715  REM IF MINUS PUT SPECIAL SYMBOL ON OPERATOR STACK
1720 NP = NP + 1:OP$(NP) = "]": RETURN
```

Fractionated requires:

Microcomputer with Microsoft BASIC

puts a (on the operator stack, checks for a unary minus sign, and then returns to line 1210 for the next item in the equation. In line 1310 the program checks to see if a + or − sign was the symbol encountered. If it was, this implies that the precedence was such that the last operation on the stack should be done since these are the lowest precedence. The program accomplishes this by jumping to the subroutine starting at line 400, which checks the symbol on the top of the operator stack and then does the required operation. if the operation is +, −, *, or / then the appropriate subroutine starting at line 100, 110, or 120 is called. These routines for binary operations in lines 100 to 120 in turn feed into line 130, which reduces the fraction to the lowest common denominator. Upon return to the subroutine starting at line 400, the result of the operation is placed on the number stack and the program returns to get the next part of the equation.

The next check (line 1330) is to see if the present symbol is a * or / and the operator on the top of the operator stack is also a * or /. If this is true, then the operation is done by jumping to the subroutine at line 400; otherwise (line 1340) the symbol * or / is put onto the operator stack since it has a higher precedence and is not to be done yet. Finally if a) is encountered, the program unfolds the operator stack until a (is encountered.

Starting at line 1400 the program interprets a number by using the VAL function. If the first number is not followed by a / then the number is assumed to be a whole number and the denominator is set to 1. The numerator and denominator are put on the number stack in line 1450.

In the section of the program starting at line 1500, we parse out the name of a variable used in the equation to be evaluated. This is accomplished by stepping through the characters in LN$ until a character is found that cannot be part of the variable name. After this name is found in line 1540 the list of previously defined variable names is searched. If the name is not found in the list then an error message is printed; otherwise the value associated with that variable, stored in VT and VB, is put on the stack.

Eventually the program comes to the end of the equation it is evaluating and must complete all of the operations that are left on the stack. This is accomplished by the code starting at line 1600. The value calculated is then printed on the screen along with the variable name in line 1630. A search of the variable name list is made to see if this variable has been defined before. If the variable has been defined, then the values are substituted in, and if it has not been defined, then a new variable is created. The program then jumps to line 1100 to get another equation to evaluate.

Fairly complex calculations with fractions are made easier by using this program. It is possible, however, to expand the program to make it even easier to use. For example, with the addition of an editor and a few control commands, the program could be expanded to enable you to write programs that work in fractions similar to the way BASIC works with decimals. Another possible way to change the program is to redefine the arithmetic so that the two numbers now representing the numerator and denominator are interpreted to be the real and imaginary parts of a complex number; then one could have a calculator that does complex number arithmetic.

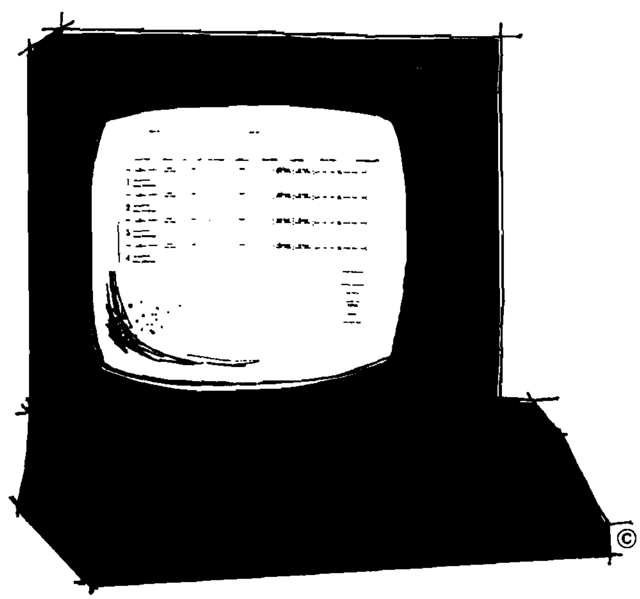
LeRoy Moyer is a physicist for the Army and teaches computer programming part time at Piedmont Virginia Community College. You may contact him at Rt. 9, Box 236, Charlottesville, VA 22901.

MICRO™

The MICRO LEARNING CENTER

Atari 800 **Texas Instrument**

Features:



- **Four-Color Hi-Res Graphics for the VIC-20**
  by Brian S. Zupke

  Use the joystick to draw pictures in four different colors.

- **VIC Player**
  by Phil Daley and Bob Tripp

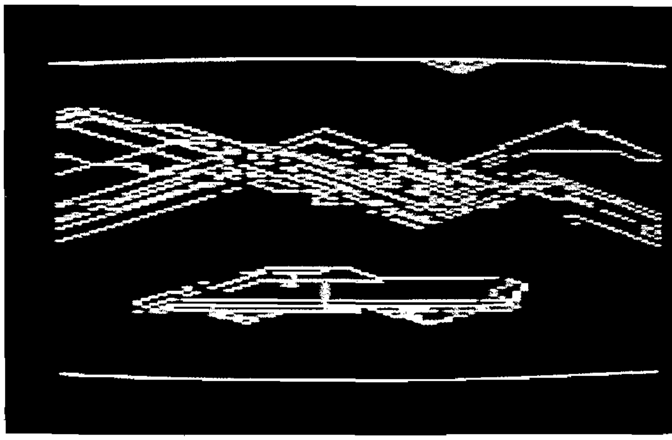
  A five-octave keyboard program.

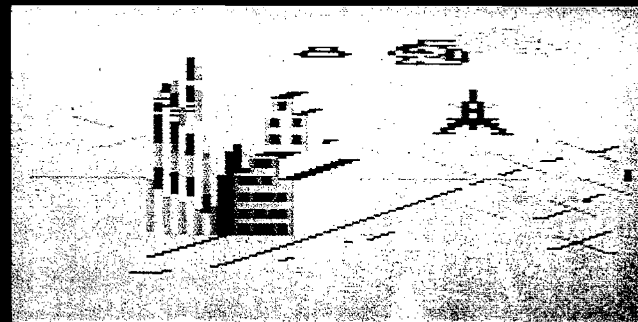
- **An Inexpensive Joystick for the VIC-20, C64, and Atari**
  by David A. Bryson

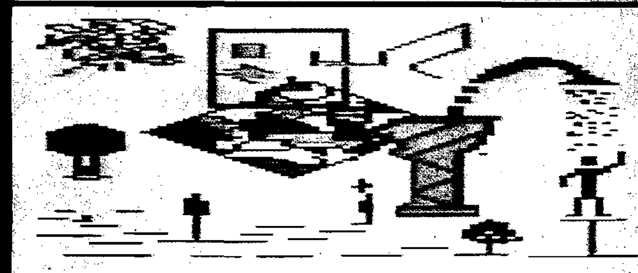
  For only $15.00 and two hours time, you can have a lightpen for all three machines.

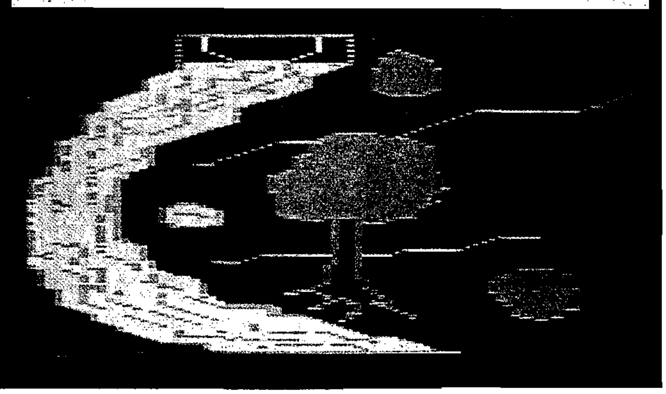
Four Color High-Res Graphics for The VIC-20

by Brian S. Zupke









Multicolor mode graphics on the VIC-20.

Photos taken from AMDEK Color I Monitor.

Many VIC-20 programmers know how high-resolution graphics are done — but with only two colors (screen color and character color). The VIC is capable of displaying four-color hi-res graphics using the multicolor mode. The question is how is it done?

In order for the VIC to distinguish among four colors within one character, a sacrifice has to be made. First, with two-color graphics each pixel in a character has only two states: on or off. One bit is needed to address each pixel. Using the multicolor mode, there are four states in which a pixel can be. Two bits are necessary for describing each pixel. This means that the VIC either must use twice as much memory for the character-bit map to have the same resolution, or it must have half the normal resolution. The VIC is incapable of the first alternative, so you are stuck with a lower resolution. Each color pixel is twice as wide (two regular pixels) as it is high. The entire screen has a total resolution of 16,192 pixels in this mode, and each character becomes 4 pixels wide and 8 or 16 pixels high.

The four colors available in the multicolor mode are screen color, border color, character color, and auxiliary color. Whether the multicolor mode for a character is used or not is determined by the value of the corresponding color memory location. If bit 3 is set, or if the value of the location is from 8 to 15, then the computer views that character as multicolored. It is possible to have regular graphics and multicolored graphics on the screen at the same time by setting the color memory locations accordingly.

Screen color is designated by a 00 (binary) value. If a pixel in a multicolored character has this value, then it will be the same color as the screen. Border color is designated by 01, character color by 10, and auxiliary color by 11. The value of the auxiliary color is located in the four most significant bits of location 36878. It can have a value of from 0 to 15 and is set by

```
POKE 36878,16 * COLOR
or, if you are also using sound,
POKE 36878,PEEK(36878) AND 15 OR
(16 * COLOR)
```

Although only four colors can exist in a character at one time, the colors can be changed easily by changing the color of the screen, border, character, or auxiliary color.

To address multicolor graphics on the screen, a method similar to the one used in two-color plotting is used. The double characters (8 x 16 pixels) are used, and plotting is done by ANDing and ORing bytes in the character memory, except now you have two bits to change at the same time.

First you must determine which character contains the pixel to be changed. The grid will be 20 characters wide (80 color pixels) and eight characters high (128 color pixels). Finding the correct character is done by

CHAR = 20 * INT(Y/16) + INT(X/4)

where X and Y are the coordinates of the point. Y is divided by 16 since each character is 16 pixels high, and it is multiplied by 20 for the 20 characters in each row. X is divided by 4 since there are only four pixels across each character. To determine which byte is to be changed, you must use

BYTELOC = BASE + 16 * CHAR + Y - INT(Y/16) * 16

where BASE is the base address of the character bit map. CHAR is multiplied by 16 to skip over the 16 bytes of each character preceeding it in the table. The remainder of Y divided by 16 gives the correct byte in the character to be changed. Sounds all too familiar, right? Now for the hard part. You must leave six of the eight bits unchanged and change the two correct bits to the color you want. First determine which of the four pairs of bits to change:

PAIR = 4 ↑ (3 - (X - INT(X / 4) * 4))

The remainder of X divided by 4 determines which pair of bits is to be changed. It has a value of from 0 to 3. By raising 4 to the reverse-of-this-remainder'th power (3,2,1,0 instead of 0,1,2,3), the base value of the pair is known. It is the value of the least significant of the two bits in each of the four pairs. Now the color value to be plotted must be replaced with the value that already exists in the bit pair. These two bits must be cleared first. This is done by ANDing the pair with zeros and the remaining six bits with ones:

BYTE = BYTE AND (255 − PAIR * 3)

Now that the bit pair has been cleared, you can add the color you want by ORing the cleared bit pair with the value of that color (from 0 to 3):

BYTE = BYTE OR (PAIR * COLOR)

When the value of BYTE is placed back into the bit map, the added color will be displayed.

The program COLOR DRAW runs on an unexpanded VIC and requires a joystick. Adding the 3K expander will allow you to add more features, but adding 8K or more will, surprisingly, result in not enough memory for the program! By moving the start of BASIC to the beginning of the expansion RAM, you can avoid this problem. The screen, border, character (all characters have the same color in this program), and auxiliary colors may be changed at any time by pressing "C". The computer will ask for their values. The auxiliary and screen colors can be from 0 to 15 while the border and character colors are from 0 to 7. Table 1 shows which color corresponds to each number.

The four function keys are used to switch between the colors (screen, border, etc.): F1 is the screen color, F3 is the border color, F5 is the character color, and F7 is the auxiliary color. The joystick is used to direct the blinking cursor. If you wish to move the cursor without disturbing the screen then hold the fire button down when you move it, which allows you to cross over different colors.

MICRO

Listing 1

```
10 POKE56,20:POKE52,20:V=36864:T=5120:D=16:PB=37152:DD=PB+2:PA=PB-1
20 INPUT"NEW SCREEN";A$:IFA$="Y"THENFORX=TTOT+2560:POKEX,0:NEXT
30 DEFFNA(J)=4↑R*J:GOSUB1000
100 POKEV+15,W:POKEV+14,A:FORX=38400TO38720:POKEX,S:NEXT
110 POKEV+3,47:POKEV+5,253:X%=0:FORY=0TO7
120 FORX=0TO19:SA=7725+Y*22+X:POKESA,X%:POKESA+30720,C:X%=X%+1:NEXTX,Y:J=0
130 X%=X/4:Y%=Y/D:R=3-(X-X%*4):S=20*Y%+X%
140 SA=T+(D*S)+Y-(D*Y%):SC=PEEK(SA):Q=SC:GOTO150
150 GOSUB2000:GETA$:IFA$=""THENK=0:GOTO170
160 K=ASC(A$):IFK>132THENJ=K-133
170 IFK=67THENGOSUB1000:GOTO100
180 POKEDD,127:K=(NOTPEEK(PA))AND60-((PEEK(PB)AND128)=0):POKEDD,255
190 IFK<>0THENX=X+SGN(KAND1)-SGN(KAND16):Y=Y+SGN(KAND8)-SGN(KAND4):GOTO210
200 GOTO150
210 IFX>79THENX=79
220 IFX<0THENX=0
230 IFY>127THENY=127
240 IFY<0THENY=0
250 IF(KAND32)=0THENSC=SCAND(255-FNA(3))ORFNA(J)
260 POKESA,SC
270 GOTO130
1000 POKEV+3,46:POKEV+5,240:POKEV+15,27:INPUT"❏SCRN,BOR,CHA,AUX";S,B,C,A:C=C+8
1010 A=A*D:W=S*D+B:PRINT"❏":RETURN
2000 IFFTHENF=0:Q=QAND(255-FNA(3))ORFNA(J):GOTO2020
2010 F=1:Q=QAND(255-FNA(3))ORFNA(3-J)
2020 POKESA,Q:RETURN
```

Value	Color
0	BLACK
1	WHITE
2	RED
3	CYAN
4	PURPLE
5	GREEN
6	BLUE
7	YELLOW
8	ORANGE
9	LIGHT ORANGE
10	PINK
11	LIGHT CYAN
12	LIGHT PURPLE
13	LIGHT GREEN
14	LIGHT BLUE
15	LIGHT YELLOW

Table 1

You may contact Mr. Zupke at 5152 Marcella Ave., Cypress, CA 90630.

VIC PLAYER

Convert your VIC into a simulated organ with VIC Player. The keys of the organ are represented by the keys on the VIC keyboard.

by Phil Daley and Bob Tripp

With the VIC Player installed in your computer you can make your VIC an entertaining and instructive device. Each note you play can be heard on the VIC sound registers. The keyboard spans three complete octaves, and the range can be extended by selecting among three overlapping registers for a total range of five octaves! Each note that is played is stored in memory so that it can be instantaneously replayed or saved on cassette tape for later use. Then you can load your song from cassette tape and replay it.

We have included a feature in VIC Player that allows you to stop playing, go back to correct mistakes, replay the song from the beginning to the current note, and then continue playing additional notes.

Using the VIC Player

After choosing the 'PLAY SONG' option from the menu, a representation of the VIC keyboard is printed on the screen in the format of a two-keyboard organ. The bottom row of keys represents the lowest notes starting with 'C' and ascending alphabetically. The second row of keys represents the sharps and flats (black keys) that correspond to the first row. Note that there are in-between keys on the second row of the VIC for every pair of first row notes. This is different than the normal organ keyboard and means that some of the second row keys do not sound when pressed (A, F, and K). These keys can be used to introduce rests into your song.

The third row of keys represents the second keyboard of the organ starting at middle 'B' and ascending to high 'C'. 'C' is not listed on the screen display due to space limitations, but is available by pressing the '*' key. The top row of keys represents the sharps and flats corresponding to this second keyboard. When you have mastered the keyboard, you are well on your way to composing your own music. Read on.

At the beginning, the program waits for you to start the song. This is one of the few times when a pause doesn't count. Once you start playing, the computer keeps track of every note and its length and register exactly as you play it. Practice a little bit to get the feel of the keyboard. It is not as simple as a piano, especially with the letters on the keys distracting you from what the true note is. The white keys on the display have the actual name of the note printed over the keyboard name of the key to help keep you oriented.

After you start a song, you may discover that you didn't mean to play a particular note. Fortunately there is a mistake-recovery method. As soon as you realize that you have made an error (sometimes the first note is an error), press the space bar to pause momentarily. You will be presented with several options.

1. CONTINUE allows you to start playing the song at exactly the point where you stopped. This is a useful technique for the times when you become confused as to which note you want to play next; press the space bar to pause, regather your wits, and press 'C' for continue, continuing from where you stopped.
2. REPLAY will play the song up through the current note so that you can inspect your masterpiece as you input and make corrections if necessary. This option can be chosen as many times as you need it.
3. BACKUP is the option for which you've been waiting. This allows you to remove one note at a time from the current song, all the way back to the beginning, if you want. When you make a mistake and press the space bar to pause, press the 'B' option and the note you are erasing will sound. Another 'B' will erase the next note, and so on. Then pressing 'C' will allow you to continue your song from the point to which you have backed up.

If, no matter how hard you try, you can't seem to get the song perfect, then the next step is to use the 'SONG EDITOR' program. This program is described in detail later in the article.

Other Menu Options

There are five additional menu options that allow you to hear your song, load a song from or save a song to the tape player, change the tempo of the song, or quit.

Choosing option number two, 'REPLAY SONG', will play the song currently in memory over the television speaker. The routine uses the current tempo for the speed at which to play the song. If there are no notes in the song, then 'NO SONG IN MEMORY' is printed and you are returned to the menu.

The 'SAVE' and 'LOAD' options are numbered three and four, respectively. To save the current song, choose '3' and answer the 'WHAT IS THE NAME OF THE SONG?' question with the name that you want to call the song. After you press < return > you are prompted to press RECORD and PLAY on the tape player. When the song is saved, you are returned to the menu and the tape player will stop. If there are no notes in the song, you will be so informed and returned to the menu.

Option number four is similar in operation to option number three. Remember that when you LOAD a new song, you will erase any song currently in memory; you must SAVE the current song (if you want to keep it for later use) before loading a new one. The 'WHAT IS THE NAME OF THE SONG?' prompt will appear. If you don't know the name of the song, or you want the next song on the tape, press < return > and the next song will be loaded. If you have several songs on the tape, type the specific name of the song you want and the VIC will search through the tape until it finds the correct song. If the song isn't on the tape, you will have to press the RUN/STOP key to recover.

To change the current tempo setting, choose option five. The minimum (fastest) setting allowed is '1'. There is no restriction on the maximum (slowest) setting, except the limits of the VIC. However, a setting of over one hundred will result in extremely long notes.

When you are finished with VIC PLAYER and want to return to BASIC, choose option six. Always remember to save any song that you are currently working on before choosing this option. If you forget this rule, typing 'GOTO20' *might* enable you to return to the program without losing your song.

The Program

The VIC Player program contains six major functions, which are selected from a menu, three minor functions used during the playing of a song from the keyboard, and some support subroutines.

The main program (lines 10-50) calls subroutines to initialize program, turn off the sound generators, and print the menu on the display. It waits for a character in the range of the menu (1 to 6) then goes to the appropriate subroutine to service the selected function.

The PLAY SONG subroutine (lines 1000-1120) calls a subroutine to print the music keyboard, gets input from the keyboard, calls a subroutine to pack information about the current note and store it in memory, and then, depending on which key you press, will do one of the following things:

Space goes to the 'Continue, Replay, Back Up' subroutine;

Cursor Up or Cursor Right terminates the song by putting a 0 in the next song location and returning to the main menu;

'f7', 'f5', or *'f3'* sets the register number and the current sound register pointer, calls a subroutine to turn off all sound registers, and continues in the

PLAY SONG routine;

An *undefined key* is converted to a 'musical rest'; and,

A *defined key* (i.e., a 'music key') is converted to its sound generator.

The REPLAY SONG subroutine (lines 2000-2070) prints the message 'PLAYING' and the song title, if there is one. If there is no song in memory, it prints the message 'NO SONG IN MEMORY', goes to a subroutine that produces a delay to enable the message to be read, and then returns to the main program.

If there is a song in memory, REPLAY calls a subroutine to unpack the register number, note pointer value, and duration for the next note from the song table. It uses a subroutine to output the note to the sound generator for the specified duration. The sound generator is then turned off and the song pointer incremented to the next note. If the maximum song length has been exceeded, then a return is made to the main program; otherwise the keyboard is checked and if any key is pressed a return is made to the main program.

If no key is pressed, the next note of the song is unpacked and tested. If it is not the 'end of song' indicator (a zero value) the program continues playing the song. At the end of the song, REPLAY uses a subroutine to generate a brief delay and then goes to the main program.

The SAVE SONG subroutine (lines 3000-3030) uses a subroutine to print the message 'WHAT IS THE NAME OF THE SONG?' and inputs a song name. It opens the cassette device for saving information and outputs the song information one note at a time. When it detects the zero note that signals the end of the song it closes the cassette device, prints the message 'SAVED', and exits through the delay routine.

The LOAD SONG subroutine (lines 4000-4030) uses a subroutine to print the message 'WHAT IS THE NAME OF THE SONG?' and inputs a song name. It opens the cassette device for loading information and inputs the song information one note at a time. When it detects an empty note it closes the cassette device, prints the message 'LOADED', and exits through the delay routine.

The CHANGE TEMPO subroutine (lines 5000-5020) prints the current value of the tempo

(continued)

VIC Player Listing

```
10 POKE36879,27:GOSUB12000
20 GOSUB9500:GOSUB10000
30 GETA$:IFA$=""THEN30
40 V=VAL(A$):IFV<1ORV>6THEN30
50 ONVGOSUB1000,2000,3000,4000,5000,6000:GOTO20
1000 V=0:F=0
1010 GOSUB11000:PEM PRINT KEYBOARD
1020 D=0
1030 GETA$:D=D+1:IFA$=""THEN1030
1040 IFF=1THEN1050
1045 GOSUB9700:POKECR,0
1050 F=0:IFA$=" "THENGOSUB8000:GOTO1010
1060 IFA$="■"ORA$="■"THENPOKECR,0:WX(V+1)=0:RETURN
1070 TP=ASC(A$)
1080 IFTP>133ANDTP<137THENR=136-TP:CR=R+RG:GOSUB9500:F=1
     :GOTO1020
1090 NX=TP-42:IFNX>50THEN NX=3
1100 NP=NP%(NX):POKECR,AX(NP)
1110 V=V+1:IFV=LTHENWX(V)=0:RETURN
1120 GOTO1020
2000 PRINT"■PLAYING":PRINTB$:V=1
2010 IFWX(V)=0THENPRINT"NO SONG IN MEMORY":GOTO9000
2020 GOSUB9200
2030 GOSUB9800
2040 POKECR,0:V=V+1:IFV>LTHENRETURN
2050 GETA$:IFA$<>""THENRETURN
2060 GOSUB9200:IFX<>0THEN2030
2070 GOTO9000
3000 GOSUB9100:OPEN1,1,1,B$:PRINT#1,B$,CR$;
3010 PRINT#1,WX(V);CR$;
3020 IFWX(V)=0THENCLOSE1:PRINT"SAVED":GOTO9000
3030 V=V+1:GOTO3010
4000 GOSUB9100:OPEN1,1,0,B$:INPUT#1,B$:PRINTB$
4010 INPUT#1,WX(V)
4020 IFWX(V)=0 THEN CLOSE1:PRINT"LOADED":GOTO9000
4030 V=V+1:GOTO4010
5000 PRINT"TEMPO=";S
5010 PRINT"NEW TEMPO=";:INPUTS:IFS<1THEN5010
5020 RETURN
6000 PRINT"■":END
8000 GOSUB9500:GOSUB9200
8010 PRINT"■■■■CHOICES:"
8020 PRINT"■C■ONTINUE"
8030 PRINT"■R■EPLAY"
8040 PRINT"■B■ACK UP 1 NOTE"
8050 PRINT:PRINT"WHICH?"
8060 GETA$:IFA$=""THEN8060
8070 IFA$="C"THEN V=V-1:GOSUB9200:GOTO9500
8080 IFA$="R"THENGOSUB2000:GOTO8010
8090 IFA$<>"B"THEN8060
8100 IFV=1GOTO8060
8110 V=V-1:GOSUB9200:GOSUB9800
8120 GOSUB9500:GOTO8060
9000 FORI=1TO700:NEXT:RETURN
9100 PRINT"■WHAT IS THE NAME":PRINT"OF THE SONG"
```

(continued)

```
9110 V=1:INPUTB$:RETURN
9200 X=WX(V)
9210 R=INT(X/10000):CR=RG+R
9220 Y=INT(X-R*10000)
9230 NP=INT(Y/100)
9240 D=INT(Y-NP*100)
9250 N=AX(NP):RETURN
9500 FORI=0TO2:POKERG+I,0:NEXT:RETURN
9700 IFD>99THEND=99
9710 WX(V)=INT(R*10000+NP*100+D)
9720 RETURN
9800 POKECR,AX(NP):FORI=1TO(S*D)/8:NEXT:RETURN
10000 PRINT"■■■■■":V=0
10010 PRINT"■■■■VIC PLAYER"
10020 PRINT"■■■■1■ PLAY SONG":PRINT"■2■ REPLAY SONG"
10030 PRINT"■3■ SAVE SONG":PRINT"■4■ LOAD SONG"
10040 PRINT"■5■ CHANGE TEMPO ■■";S;"■■"
10050 PRINT"■6■ QUIT"
10060 PRINT"■CHOOSE?":RETURN
11000 PRINT"■PLAY WHEN READY"
11010 PRINT"■■■SPACE BAR TO PAUSE":PRINT"CRSR TO END SONG"
      :PRINT"A, F, OR K = REST■■"
11020 PRINT"______________________________";
11030 PRINT"||■ ■|■ ■|■ ■|||■ ■|■ ■|||■ ■|■ ■|■ ■|";
11040 PRINT"||■2■|■3■|■4■|||■6■|■7■|||■9■|■0■|■+■|";
11050 PRINT"■■|└┴┴┴┘|└┴┴┘|└┴┴┴┘";
11060 PRINT"■|■F■|■G■|■A■|■B■|■C■|■D■|■E■|■F■|■G■|■A■|■B■";
11070 PRINT"■|Q|W|E|R|T|Y|U|I|O|P|@";
11080 PRINT"■■└┴┴┴┴┴┴┴┴┴┴┴┘";
11090 PRINT"■ ||■ ■|■ ■|||■ ■|■ ■|■ ■|||■ ■|■ ■||";
11100 PRINT"■ ||■S■|■D■|||■G■|■H■|■J■|||■L■|■:■||";
11110 PRINT"■ |└┴┴┘|└┴┴┴┘|└┴┴┘|";
11120 PRINT"■ |■C■|■D■|■E■|■F■|■G■|■A■|■B■|■C■|■D■|■E■|";
11130 PRINT"■ |Z|X|C|V|B|N|M|,|.|/|";
11140 PRINT"■■ └┴┴┴┴┴┴┴┴┴┴┘■";
11150 RETURN
12000 S=50:L=260:DIMWX(L),AX(38),NP%(50):CR$=CHR$(13)
12010 RG=36874:FORI=0TO2:POKERG+I,0:NEXT
12020 POKE36878,15
12030 RESTORE
12040 FORI=0TO38:READAX(I):NEXT
12050 FORI=0TO50:READNP%(I):NEXT
12060 R=1:CR=RG+R:RETURN
13000 DATA135,143,147,151,159,163,167,175
13010 DATA179,183,187,191,195,199,201,203
13020 DATA207,209,212,215,217,219,221,223
13030 DATA225,227,228,229,231,232,233,235
13040 DATA236,237,238,239,240,241,0
14000 DATA36,34,12,38,14,16,32,38,18,20
14010 DATA22,38,25,27,38,30,15,38,38,38
14020 DATA14,16,35,38,7,4,3,21,38,6,8,29
14030 DATA10,38,13,11,9,31,33,17,23,1,24
14040 DATA28,5,19,2,26,0,38,37
```

Perry Peripherals Repairs KIMs!!

(SYMs and AIMs Too)

- We will Diagnose, Repair, and Completely Test your Single Board Computer
- We Socket all replaced Integrated Circuits
- You receive a 30-day Parts and Labor Warranty
- Your repaired S.B.C. returned via U.P.S. — C.O.D., Cash

Don't delay! Send us your S.B.C. for repair today
Ship To: (Preferably via U.P.S.)

Perry Peripherals
6 Brookhaven Drive
Rocky Point, NY 11778

KIM-1 Replacement Modules

- Exact replacement for MOS/Commodore KIM-1 S.B.C.
- Original KIM-1 firmware — 1K and 4K RAM versions

Replacement KIM-1 Keyboards

- Identical to those on early KIMS — SST switch in top right corner
- Easily installed in later model KIMs

Perry Peripherals is an authorized HDE factory service center.

Perry Peripherals carries a full line of the acclaimed HDE expansion components for you KIM, SYM, and AIM, including RAM boards, Disk Systems, and Software like HDE Disk BASIC V1.1. Yes, we also have diskettes. For more information write to: P.O Box 924, Miller Place, NY 11764, or Phone (516) 744-6462.

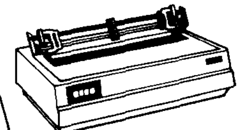
Circle No. 47

WE TOOK A BIG BYTE OUT OF COMPUTER PRICING!

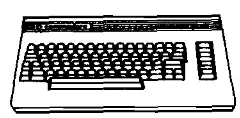
ORDERING INFO
We accept Visa, Mastercard, Money Orders or Certified Check. Personal checks require 2 weeks for bank clearance. All items factory fresh & carry manufacturer's warranty. Prices subject to change without notice.

SPECIALS OF THE MONTH

OKIDATA 82A
$419.



COMMODORE 64
$ CALL



ATARI 800
$ NEW LOWER PRICE!

COMPUTERS

ATARI 400	$197.
ATARI 800	$598.
ATARI 410	$74.
ATARI 810	$439.
COMMODORE 64	CALL
COMMODORE VIC 20	$149.
COMMODORE VIC 1530	$69.
NEC PC 8001A	$739.
NEC PC 8012A	$499.
NEC PC 8031A	$739.
SANYO MCB 1000	$1599.
TIMEX 1000	$84.
XEROX 5¼"	CALL
XEROX 8"	CALL
XEROX 630	CALL

DISKETTES

BASF	CALL
MAXELL	CALL

TERMINALS

TELEVIDEO 910	$589.
TELEVIDEO 950	$945.

PRINTERS

DIABLO 620	$1199.
DIABLO 630	$1675.
OKIDATA 82A	$419.
OKIDATA 83A	$699.
OKIDATA 84P	$1029.
EPSON	CALL
NEC 8023	$479.

SOFTWARE

MICROSOFT	CALL
MICROPRO	CALL
ALL MAJOR BRANDS	CALL

COMPUWAY, INC.

24 LUMBER ROAD
ROSLYN, N.Y. 11576
toll free 800 6451362
516 6211362

Circle No. 48

variable, then requests and inputs a new value. If the value is less than 1, it is ignored. When a valid value is input, the routine returns to the main program.

The QUIT routine (line 6000) clears the screen and executes an END to return to BASIC.

The PLAY SONG MENU routine (lines 8000-8120) provides three additional commands for use while playing a song. When called by pressing the space bar, it first uses a subroutine to turn off all sound generators and another to unpack the current note. It prints its own menu and waits for a keyboard selection. On receipt of a 'C' it backs up the song pointer to the current note, unpacks the note, and turns off the sound generators before returning to the PLAY SONG routine.

The letter 'R' calls the REPLAY SONG subroutine, which plays the current song from the beginning to the current note, and then waits for another menu selection.

The letter 'B' causes the song note pointer to back up one position, unless it is already at the start of the song, and then go to the subroutines to unpack the note's values, output the note for the correct duration, turn off the sound generators when the note is done, and wait for another menu selection.

The support subroutines (lines 9000-13020) provide support for the main program and major subroutines.

Line 9000 provides a several-second delay to permit you to view messages.

Line 9100 prints 'WHAT IS THE NAME OF THE SONG?' and accepts a name from the keyboard.

Line 9200 unpacks the stored note information into its three components: the song register 'R', the note pointer 'NP', and the duration of the note 'D'. It also sets the correct song register and looks up the actual note from the note table. See 'packing information' in the section headed "Numeric Variables."

Line 9500 turns all three song registers off by setting them to 0.

Line 9700 makes sure that the note duration is not greater than 99 and packs the three components of the note (the song register 'R', the note pointer 'NP', and the duration of the note 'D') into a single integer value in the song array. See 'packing information' under the heading "Numeric Variables."

Line 9800 outputs a note by placing its value in the current sound generator. It waits for the duration of the note, which is calculated as the tempo 'S' times the note duration 'D' divided by 8, times the length of the BASIC FOR...NEXT loop.

Line 10000 prints the main menu.

Line 11000 prints the keyboard display.

Line 12000 performs a series of initialization functions. It sets the tempo 'S' to 50 and the length of the song 'L' to 260 notes; it dimensions three integer arrays — (W%(L) to hold the song note information, and A%(38) and NP%(50), which associate the keyboard characters with the notes; it turns off all three sound generators and sets the sound volume to its maximum value of 15; it restores the DATA statement pointer and reads the DATA into the A%(I) array and the NP%(I) array; and it sets the middle sound register as the current register and returns to the main program.

Line 13000 contains the data for the values of each of the notes (three octaves plus two notes, C through C#). The zero at the end is the entry for a rest. These values correspond to the values in the VIC programming manual. Note that they vary slightly from the values in the VIC reference manual.

Line 14000: This data table has the pointers for the keyboard playing routine. Since the keyboard is not in note order and the note table values are in note order, it is necessary to convert from the keyboard code to the position in the note table. For example, 'Z', which is a 'C', has the keycode 90. This number is called the ASCII value and is a standard form of encoding the keys for a computer. Line 1090 subtracts 42 from 90, resulting in 48 as the keyboard position. (The first 42 keycodes are unused, and so we throw them away.) If you look at the table of pointers, you will see that the 48th entry is '0'. Therefore, the note pointer for Z is 0. If you look at the note table data, you will see that the zero*th* (first) entry in this table is 135. This is the value to be POKEd to make the sound 'C'.

Programming Concepts

Using a Menu to Make a Choice

1. Selecting by Number

When you run the PLAYER program, the first display that you see is a list (or menu) that tells you what actions are available. Each item on the menu is selected by pressing the number associated with it. The BASIC program steps required to make the choice are in lines 40 and 50.

```
40 V=VAL(A$):IFV<10RV>6THEN30
50 ONVGOSUB1000,2000,3000,4000,5000,6000:
   GOTO20
```

Line 40 converts the keyboard character in A$ to its numeric value in variable V. If V is less than 1 or greater than 6, then the number is ignored and the program returns to line 30 to get another choice from the keyboard.

Line 50 uses ON V GOSUB to go to the subroutine whose position matches the number: the first subroutine address for a 1 (subroutine 1000), the second subroutine address for a 2 (subroutine 2000), and so forth for numeric values of 1 through 6.

2. Selecting by First Letter

If the space bar is pressed during the 'PLAY SONG', then another menu is displayed.

(continued)

YOU CAN MAKE A FORTUNE IN MICROS!

THE MICRO COMPUTER BUSINESS WILL GROW FROM $10 TO $100 BILLION IN THE NEXT EIGHT YEARS! ARE YOU READY TO CASH IN?

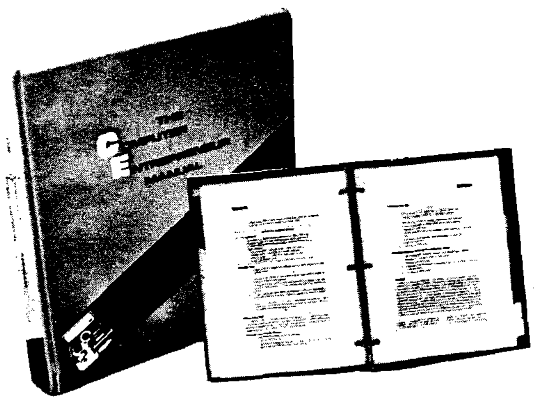
The **micro computer business** is predicted to grow from its present $10 billion to **$100 billion** before 1990! Imagine the possibilities this opens for you! No matter where you live, if you're starting up or presently in business, **no other industry offers you more opportunities!**

Now, finally, all the inside information you need to secure a prosperous future in this dynamic industry is available in **one place - THE COMPUTER ENTREPRENEUR MANUAL!** – An **immense** information source, compiled by our **inquisitive** research team, aided by a panel of **experts** and **business people** from **all areas** of the **computer industry!**

We present the inside story of **more than 100 lucrative computer businesses you can enter**, where you'll find the **real opportunities** for the eighties: from one man operations like **Programming Author, Word Processing Center** or **Consulting,** to **Systems House, Service Bureau, Computer Store etc!** Many at little or no investment! All the invaluable facts and figures: **How to start, Capital needs, Profit estimates and Margins, How to Sell and Market, How missing** technical or business experience **need not stand in your way, Source of Suppliers, etc!** Details that could take **years** to find out on your own!

We'll show you **inside tricks,** like how to **never again pay retail** for computer products and consumer electronics, **even for one item - right now,** while you're starting your business! How to get **free merchandise** and trade show **invitations, etc. This alone** will **more than pay** for the **manual!** You'll read **actual case histories** of other computer entrepreneurs, so you can **learn** from **their mistakes,** and **profit** from their **success stories! Where** you'll be **one year** from now depends on **your actions today!** Let us **show you how** to take the **first crucial steps!**

Order now and take **advantage** of our **limited introduction special, THE COMPUTER ENTREPRENEUR MANUAL,** and a **six month subscription to THE COMPUTER ENTREPRENEUR REPORT/NEWSLETTER** (so you're **always up-to-date** with the industry), **both** for **only $29.95! You must** be **convinced** on **how easy** you can **strike it rich** in the micro computer business - or you may **return** the manual for **a full refund** within thirty days! **USE OUR TOLL FREE NUMBER TO ORDER!**



EVERYTHING YOU NEED TO KNOW TO SUCCEED IN THE COMPUTER BUSINESS IS ALL IN THIS MANUAL!

THE COMPUTER ENTREPRENEUR MANUAL has the **answers** to **all your questions** about **selecting, starting** and **succesfully running a computer business!** There has **never** been such a **comprehensive collection** of **know-how** and **information** about this business **in one place! All the facts you need to plan** and **acheive your goals** in **easy-to-follow, step-by-step instructions!**

These are some of the **100-plus businesses** covered in **PART ONE** of the manual, with the facts on **How to start and run, Start-up Cost (Even** how to operate **on a shoestring**), What **profits to expect, Wholesale prices, Mark-ups, Suppliers, future outlook, case histories** for **each, etc:**

Systems House, Software Author (who to sell to and who to avoid), **Service Bureau, Software Publisher** (How to find programs that sell, **Word Processing Service, Consulting** and **Consultant Broker** (use your skills or those of others, **make $150 - $1000 a day!), The incredible Games Business, Computer Store** (Franchises: Pro and Contra, or a low inventory store **in your home!**), **OEM, Hardware Mfg, Data base** and **Teletext Service (big prospects!**), **Used Computers, Repairs, Rent-A-Computer, Promote Fests** and **Trade Shows, Turnkey Systems, Bartering, Mail Order, Compile and rent mailing lists, Specialized Data Headhunting** and **Temp Help Service, Tech Writer Shop, Custom Engineering,** The **highly profitable Seminars** and **Training Business**, **and many more!**

Many **new ideas** and **ground floor opportunities! Interviews** and **success stories** on **companies** of **all sizes! Privy info on the profits made:** How some **computer store** operators **net $100 - $250,000!** Little known outfits that made their owners **millionaires,** one of these low-key companies, making simple boards, went from **nil to $20,000,000** and **100 employees** in **four years!** Programmers that make **$300,000, Thousands of micro millionaires** in the making, etc!

Whatever your goal is - **Silicon Valley Tycoon,** or just a **business at home** - **we guarantee** you'll find a **business** to **suit you - or your money back!**

PART TWO of the manual is **loaded** with the **know-how** and **"streetfighting" savvy** you need, both as a **novice** or business **veteran,** to get started, to stay and to **prosper** in the micro computer business! **A goldmine of information** in clear and easy-to-use instructions: How to prepare your **Business Plan, Outside financing,** The **mistakes** you **must avoid,** How to **hire** and **manage employees, Incorporation** (when, and how to do it **cheaply**), **Surviving bad times, Record Keeping,** how to **estimate your market before you start,** Use **multiple locations** to **maximize profits,** how to **promote** and **stay** steps **ahead of the competition!** How to get **free advertising, free merchandise, free advice, Power negotiating** with suppliers to **double** your **profit margins, etc!** Even how to **keep a present job while starting a business part time!**

Don't miss this opportunity to be part of this **great industry - the next success story** could be **your own! Order the manual today!** Part one and two, bound in **a deluxe ring binder,** where you can also **collect our newsletter (free for six months with the manual - a $32.50 value!) - all for only $29.95!**

THE COMPUTER ENTREPRENEUR NEWSLETTER – ALL THE LATEST INSIDE BUSINESS NEWS! NOW! SIX MONTHS FREE WITH YOUR MANUAL!

You're **always attuned** to the industry, and **your manual kept up-to-date,** with our **newsletter!** Each issue has the **latest business news, ideas, new suppliers,** our **indispensible "watchdog" column** on **profits, discounts** (don't miss mfg's promos, like recently, when top video monitor sold at $80 - that's half wholesale, **one third of the retail price!), the competition, the big deals, etc! Feature stories** with start-up info and case histories **on new micro businesses!**

You'll get **invitations** to trade shows and conventions, the **usage** of our **advisory service** and our **discount buying service** for your purchases!

You'll find **many items** in our newsletter that will **save you the cost of your manual many times over!**

CALL TOLL FREE!
CHARGE IT!
Credit Card Orders
(MC, VISA only)
accepted 24 hours/day
1-800-227-3800
Ask for extension 1135
In California call
1-800-792-0990

MasterCard VISA

Order by phone (Credit cards only), or use the coupon:

Mail to **THE COMPUTER ENTREPRENEUR PUBLISHING CO. PO BOX 456, Grand Central Station, New York, N.Y. 10163**

Please send me **THE COMPUTER ENTREPRENEUR MANUAL,** and the **six month free subscription** to **THE COMPUTER ENTREPRENEUR REPORT/NEWSLETTER. All for only $29.95,** plus $3 for postage/handling **(NY residents:** add $2.64 for sales tax). If I decide not to keep the manual, I may **return** it **within 30 days** for **a full refund.**

NAME: ______________________

ADDRESS: ______________________

CITY, STATE, ZIP: ______________________

☐ **Check or M.O. enclosed** **Charge to** ☐ **VISA** ☐ **MC**

CARD#: ______________________

Exp. Date: ______________________

SIGNATURE: ______________________

MM0683

The first character of each item is displayed in reversed video to indicate that the letter is to be pressed on the keyboard to select that choice. Each letter is serviced by its own IF...THEN statement.

```
CHOICES:
CONTINUE
REPLAY
BACK UP 1 NOTE

WHICH?
```

```
8070 IFA$="C"THEN V=V-1 :GOSUB9200:
     GOTO9500
8080 IFA$="R"THENGOSUB2000:GOTO8010
8090 IFA$<>"B"THEN8060
```

If there are only a few choices, as in this example, that is not a lot of code. If there were many choices, then the amount of code to service the letters could be significant.

Numeric Variables

VIC BASIC handles two kinds of numbers — integer and floating point. An integer number is a number without a decimal (123,9999,0,-4387 and so forth). A floating-point number has a decimal point (12.34, 98765.1, 0.0, -435.678 and so forth). Often it does not matter which type of number is being used, but it can make a significant difference in some programs.

Integers

An integer number in the VIC may be as large as +32767 or -32768. Each integer number that the program stores requires two *bytes* of memory. An unexpanded VIC has only about 5000 bytes of user memory, so the number of bytes available is limited. Most BASICs identify which information is stored in integer form by using a % after the one- or two-character symbol name. Examples in VIC PLAYER are F1%, F2%, T%, W%, and A%. In VIC PLAYER, W% and A% are arrays — a large number of related values with a common reference. W%, for example, is the memory reserved to save the note, register, and duration of each note of the song. To allow enough memory for a reasonably long song, we found it necessary to perform a few tricks. First, let's examine how *not* to program the storage area.

Each note of the song that you play produces three pieces of information that must be saved by VIC PLAYER:

the register of the note

the number of the note

the duration of the note

We could define a two-dimensional array that contains a floating-point number for each of the three parts of each note: DIM W(299,2) would reserve space for 300 notes, three floating-point numbers per note. How much memory do you think this would take? Well, 300 times 3 is 900. Is that the total number of bytes required? No! Each floating-point number requires five (5) bytes of memory. Therefore, it would take 5 * 900 or 4500 bytes of memory! There is barely enough space in your basic VIC for the song, and that's not counting the space required for the program itself.

One obvious way to save space would be to store the three parts of each note as integer values instead of floating-point values. Since each integer value requires only two (2) bytes of memory for storage, the total requirement for the 300-byte song would be 2 * 900 or 1800 bytes. That is better, but it still uses almost half of the memory in your basic VIC, which does not leave much room for the program.

You have to get a bit tricky to squeeze much more out of the song space; but there is nothing wrong with getting tricky when writing programs. In fact, that can be half the fun! To really squeeze the memory in VIC PLAYER we took advantage of the size of the number that a single integer value might hold. An integer requires two (2) bytes of memory, whether it contains 0, 32335, -32334, or whatever. The three values that we need to keep for each note played are: the register number (1, 2 or 3); the note number (0 to 38); and the duration (which is limited to 0 to 99 units). If only we could pack all three of these individual values into a single integer number for storage and then unpack them when we needed to use them. Well, good news — we can!

The technique to pack the numbers is shown in line 9710.

```
9710 W%(V)=INT(R*10000+NP*100+D)
```

This equation is not as difficult as it may at first appear.

W%(V) is the address of the integer where V is the number of the note in the song;

INT is the BASIC function that converts a floating point (five bytes, remember) into an integer number (only two bytes);

R * 10000 multiplies the register value (1 to 3) by 10000;

+NP * 100 multiplies the note number (0 to 38) by 100;

+D adds the duration value (0 to 99).

(continued)

VIC Player *(continued)*

All we have done is multiply two of the three parts that we need to save by enough to make sure they do not overlap. This insures that we will be able to unpack the separate parts later. The unpacking is a bit more difficult than the packing, but conceptually it is simple. All we need to do is reverse the packing process. This is accomplished in the following lines:

```
9200 X=W%(V)
9210 R=INT(X/10000):
     CR=RG+R
9220 Y=INT(X-R*10000)
9230 NP=INT(Y/100)
9240 D=INT(Y-NP*100)
9250 N=A%(NP):
     RETURN
```

Line 9200 simply copies the packed value of the note into X.

Line 9210 restores the register number by dividing the packed value by 10000, reversing the original saving process. It also sets the current register (CR) to the new register number.

Line 9220 restores the note number and duration combined value that was 'thrown away' in the previous step.

Line 9230 restores the note number by dividing by 100.

Line 9240 restores the duration by subtracting the note number component.

Line 9250 restores the value of the note by looking it up in the note table, A%(X), using the calculated note pointer, NP.

The above lines of program have taken the single integer value and converted it back into three separate parts. The savings of this method result in a 300-note song requiring only 300 integer numbers to store it, at two bytes per number, for a total storage of 600 bytes. Quite a reduction from the original 4500 bytes!

MICRO

Circle No. 53

Circle No. 54

An Inexpensive Lightpen for the VIC-20, C 64, and Atari

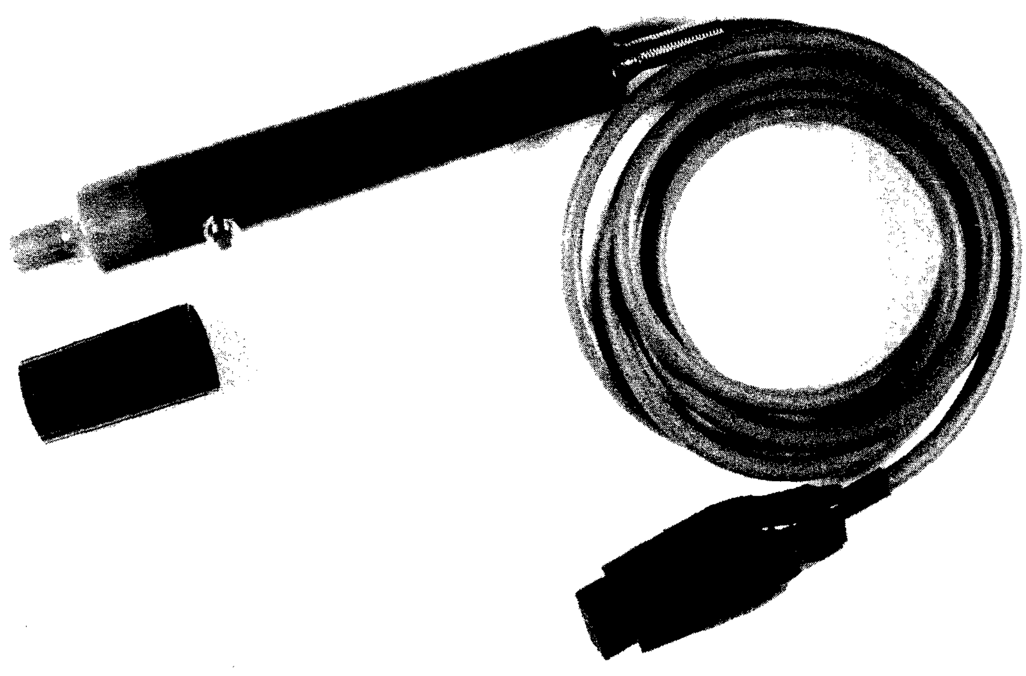
by David Bryson

What is a light pen?

A light pen is a simple device that connects to your computer with a wire. With an appropriate program in memory, you can use your light pen for a wide variety of applications. Possibilities include drawing lines or more complicated pictures and selecting items displayed on the screen by simply pointing at them. For instance, you could have a questionnaire with a box displayed next to each possible answer. By pointing the light pen at the appropriate box and pressing the light pen's button you would select that answer. There is an endless variety of game applications.

A light pen works on a very simple principle. The tip contains a light-sensitive phototransistor that senses the light of a TV's beam. The VIC, VIC II, and ANTIC chips included with VIC-20, Commodore 64, and Atari 400/800/1200 computers (as well as a number of other CRT controllers) continually keep track of the horizontal and vertical position of the TV beam. When the phototransistor detects the passing beam, it sends a signal to the CRT controller, and the current X,Y position is locked into registers that can be read by a program.

Both Commodore and Atari have light pens available. However, if you have very basic soldering and mechanical skills you can construct the inexpensive light pen described by David Bryson in this article. For further help with software, consult the references listed.



I hadn't found a light pen construction article for the VIC or Atari that produced suitable results, so I decided to design a light pen circuit that was both functional and completely compatible with hardware and software for both microcomputers.

VIC-20 Light Pen Basics

The VIC chip constantly monitors the location of the electron beam on the CRT screen and stores integer values for both coordinates in memory addresses 36870 (horizontal) and 36871 (vertical). With the light pen held in close proximity to the CRT screen, the phototransistor in the pen tip will produce a negative pulse as the beam passes over the pen tip. This negative pulse in turn latches the X and Y values being monitored by the VIC chip and stores the results in the appropriate memory address. This cycle is repeated for every frame produced by the CRT raster at a horizontal scan rate of 15,750 Hertz. If an accurate measurement of beam location is to be achieved at these frequencies, the optoelectronic device used as the light detector must be capable of a relatively high switching rate with rise and fall times in the unit microsecond range. The Motorola MRD300 phototransistor incorporated in this design has a typical switching time of two microseconds while maintaining a moderate light current sensitivity.

Light Pen Construction

You need the following parts for construction of the light pen circuit shown schematically in figure 1.

1. Phototransistor, Motorola type MRD300 or equivalent
2. Switch, subminiature SPST momentary contact. Radio Shack #275-1571 or equivalent
3. Resistor, 100Kohm, ¼ watt
4. Cable, 4 feet, 2 conductor stranded #24AWG with shield. Belden #9397-100 or equivalent
5. Connector, D-Subminiature manufactured by Robinson Nugent Inc. Available from:
   Digi-Key Corp
   Thief River Falls
   MN 56701
   Digi-Key #R700-ND or equivalent
6. Light Pen Main Housing (see text)
7. Jerk Relief Spring (optional)
8. Quick-setting Epoxy Devcon "5-minute" or equivalent

Obtain either a new or used felt-tip marking pen to serve as the light pen main housing. Use a brand with the large diameter body (about ¾ inch) to insure ample room for internal switch installation. Disassemble the pen by disconnecting the removable head section; remove the fiber ink reservoir from the main body and the felt tip from the head section. Then clean remnant ink from both pen sections and from the marking pen cover. Save the cover for later use in protecting the delicate tip of the light pen during storage.

Secure the phototransistor in position within the tip assembly and bond to the inside surface with 5-minute epoxy. Use caution during this operation to prevent any glue spillage onto the phototransistor lens area. Note that the lens tip is flush with the tip of the barrel as shown in the pictoral section in figure 2. Drill holes through the barrel end and outside surface to accommodate installation of the cable and switch assemblies. Insert the cable through the hole in the barrel end (with the strain relief spring in place if used) and make the appropriate solder connections between switch, phototransistor, and cable.

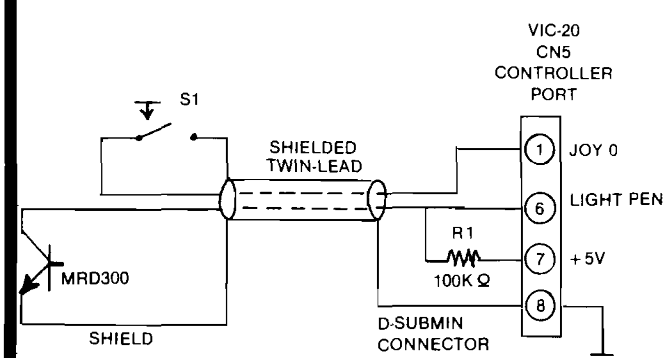


FIGURE 1:
VIC-20 LIGHT PEN SCHEMATIC

Secure the switch to the barrel assembly with the nut provided. Pour a small amount of 5-minute epoxy into the base assembly to secure the cable at the barrel and exit hole and connect the head assembly to the main body. In some felt tip pen designs, this will be a snap-on connection. Others will require a small amount of cement to secure the two pieces. If your design requires the adhesive, leave the bonding of the two pieces as the final step in pen construction in case problems arise during final test that require troubleshooting the internal barrel assembly.

Prepare the two conductors and shield at the cable end for connection to the D-Subminiature connector. If this type of connector is not readily available and if waiting the two weeks for mail order response is out of the question, consider the following options. If a used D-subminiature connector from an Atari joystick or paddle control is available, it can be used after performing razor blade surgery to relocate the internal pin receptacles to the positions needed for light pen connection. The process is not as difficult as it sounds. If carefully applied razor cuts are made along both

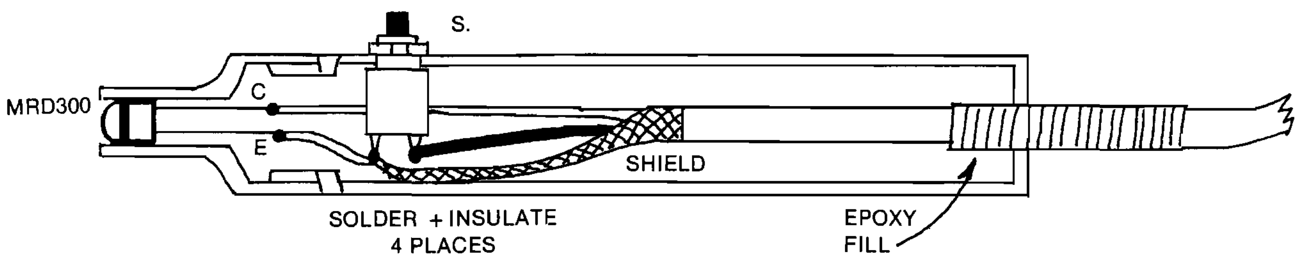


FIGURE 2: **PICTORAL SECTION OF VIC-20 LIGHTPEN**

edges of the connector, the internal halfs hinge open to allow access and switching of the pins. An easier but more expensive alternative is to purchase a Radio Shack D-Subminiature 25-pin female connector (#276-1565) and reshape the connector to fit the VIC receptacle by removing a 16-pin section of the plug body. If this operation is carefully performed, a second 9-pin D-Subminiature connector can be produced from the remnant 16-pin plug section.

Make the required solder connections between cable conductors and connector. Install and solder the 100K ohm resistor between pins 7 (+5V) and pin 6 (light pen) within the connector body. Secure and insulate the connector assembly with electrical tape if necessary.

A photograph of the assembled light pen is presented in figure 3.

Light Pen Application

Plug the completed light pen into the VIC and type in the following program:

```
10 X=PEEK(36870)
20 Y=PEEK(36871)
30 SW=-((PEEK(37151)AND4)=0
40 PRINT"CLEAR"X;Y;SW
50 FORT=1TO50:NEXT
60 GOTO10
```

The run should produce a group of three numbers displayed at the top left corner of the monitor screen. The left value is the contents of the X or horizontal register, the middle number is the contents of Y or vertical register, and the number on the right is either a 0 or a 1 depending on the state of the switch at the tip of the light pen. Note that some adjustment may be required to the brightness and/or contrast controls on the monitor to produce the desired results.

The expression in statement 30, SW=((PEEK(37151)AND4)=0), instructs the computer to monitor the state of the port B output register to determine the position of the switch at the light pen tip. By using a compound statement such as

```
100 SW= -((PEEK(37151)AND4)=0):IF SW=0
THEN 100
```

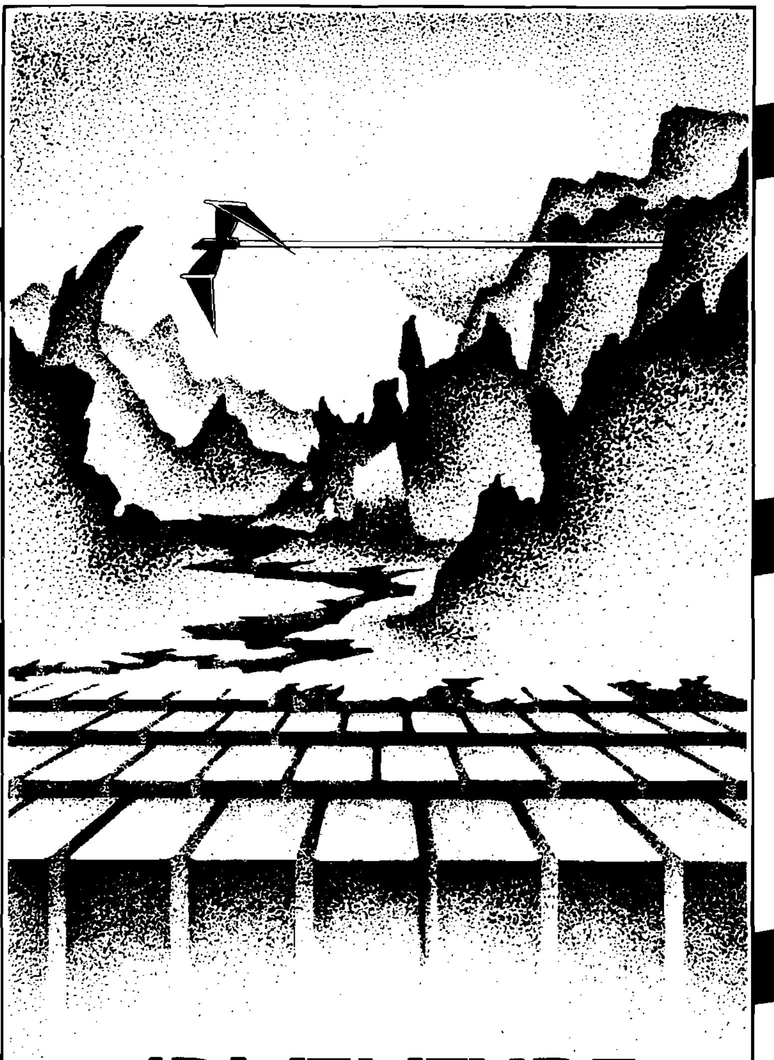
the system can be put into a loop awaiting the activation of the light pen switch before proceding with the next step in the program. One obvious advantage of this scheme is to reduce false input or "noise" caused by ambient light (other than from the CRT screen) producing a light pen signal. With this statement in the program, the pen location registers will only be examined when the switch is depressed. It is also possible to leave this statement out and provide real time screen position monitoring without switch activation, as evident in the six-line light-pen test program presented earlier.

Further Reading

1. Hale, William. "A Light Pen For Under $10," *Compute!*, #27 (August, 1982), 141.
2. Loomis, Sumner S. "Let There Be Light Pens," *The Best of BYTE*, Vol. 1, 153-157.
3. Malmberg, David. "Using The VIC Joystick," *Home and Educational Computing*, Vol. 1, Issue 1, 18-24.
4. Malmberg, David. "VIC Light Pen-Manship," MICRO, #41 (October, 1981), 54-59.
5. Peck, Robert A. "Basics Of Light Pen Operation," *Compute!*, #10 (March, 1981), 36-41.

David Bryson is presently employed as a senior materials engineer-nondestructive testing-with the Pratt and Whitney Aircraft Commercial Engineering Division of United Technologies Corporation. You may contact the author at 9 Luster Lane, Enfield, CT 06082.

MICRO



Design & Illustration: Paul Stephenson

ADVENTURE.
THE KEY IS YOUR COMPUTER

KONGO KONG

Climb ladders, avoid the barrels the crazy ape is rolling at you, and rescue the damsel. Fast machine code action.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

GRAVE ROBBERS

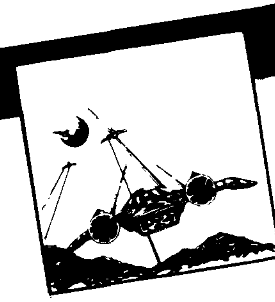
Introducing the first GRAPHIC ADVENTURE ever available on the VIC-20! With realistic audio-visual effects, you explore an old deserted graveyard and actually see the perils that lie beyond.

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$14.95

ANNIHILATOR



Defend your planet against the hostile aliens. All machine code makes this "Defender-like" program one of our best arcade games.

Available for VIC-20 and COMMODORE 64

PLAYED WITH JOYSTICK

$19.95

ADVENTURE PACK I

(3 programs)

Jack and the Beanstalk Defeat the Giant in your quest for his Golden Hen.

Computer Adventure Re-live the "excitement" of getting your computer. An adventure with a very different flavor.

Moon Base Alpha You must find a way to destroy the meteor that is racing towards your base, or else all moon colonies will be demolished!

Available for VIC-20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

ADVENTURE PACK II

(3 programs)

African Escape As the sole survivor of a plane crash, you must find your way out of the dark continent.

Hospital Adventure You are a spy whose mission is to complete the bungled assassination attempt on the evil dictator, who is recuperating in the hospital under heavy guard.

Bomb Threat Get back to town to warn the authorities of the bomb planted by the terrorists who left you prisoner at their hideout.

Available for VIC 20 and COMMODORE 64

PLAYED WITH KEYBOARD

$19.95

CHOMPER MAN

High speed machine action. Don't let the bullies catch you in a game packed full of machine code.

Available for COMMODORE 64

PLAYED WITH JOYSTICK OR KEYBOARD

$19.95

Programs for the **VIC-20** and the **COMMODORE 64**

VICTORY SOFTWARE®

All programs fit in standard memory. All programs are on cassette tape.

VICTORY SOFTWARE INC.
7 Valley Brook Road, Paoli, PA 19301
(215) 296-3787

Check your local dealer or order directly.

ORDERING We accept personal checks, money orders, VISA, and MASTER-CHARGE. Charge orders please include number and expiration date. Overseas orders please use charge or have check payable through a U.S. bank. Add $1.50 postage and handling per order. PA residents please add 6% sales tax.

VISA

master charge

Circle No. 55

68000 ADDRESSING MODES

by Joe Hootman

Immediate Addressing

Immediate addressing is used to load a constant into a register. As an example, to load the data register D0 with the hexadecimal 55, the instruction that would load D0 immediately with 0055 is given below:

Move.W #$55,D0. The equivalent opword code which is entered into memory is 303C. The # sign indicates immediate addressing, and the $ indicates a hexadecimal number.

It is worthwhile at this point to examine the format of the MOVE instruction in the previous example as the format is typical of all the 68000 instructions. The mnemonic MOVE expresses the intention, in this example, of moving data into the data register. The W following the MOVE instruction indicates that it is intended that 16 bits of data be moved into D0. In this particular example the high order byte was zero filled. If the letter following the instruction is a B, one byte of data would have been moved into D0. If the letter following the instruction is an L, a long word (32 bits) would be moved into D0. The size field in the opword designates the length of the data. For each of the different lengths of data to be moved there will be a different opword.

The general format of the MOVE instructions is represented in the following format:

Instruction	Word Size	Source,	Destination.
Mnemonic	B,W,L	Defined by the Addressing Mode	Defined by the Addressing Mode.

The MOVE instruction moves data from a designated source, in this case the source being the immediate Hex data 0055. The destination of data in this case would be the data register D0. The bit pattern for the instruction opword is (0011 000 000 111 100).

Note that the destination field as defined by the opword for MOVE does not allow for movement of data into the address register. The movement of data to the address registers is accomplished by using the instruction MOVEA. MOVE and MOVEA are identical except that MOVEA uses a fixed code for the mode of the destination. When loading data into the address register using the MOVEA instruction, the sign bit is extended.

The addressing modes clearly cannot be used to implement instructions that make no sense. For example, MOVE instructions cannot be used with addressing modes that have no way to designate the register or memory to be the destination. These addressing modes would include the PC offset, PC indexed, and immediate addressing modes for the destination effective address.

If only a byte is to be loaded into a register, the MOVEQ instruction should be used. The byte of data is included as a part of the opword; the low order 8 bits (0-7) are the data bits of the word.

To load D0 using the MOVEQ instruction the following opcode and opword would be used:

MOVEQ #$55,D0 (7055 opword code).

Many cross assemblers automatically use MOVEQ for the instruction to MOVE an immediate byte.

Direct Addressing

Two different direct addressing modes make use of either the address register or a data register. Direct addressing can be used to copy a data register into another data register or an address register. For the MOVE instruction this addressing mode requires that one of the registers be previously loaded with the appropriate data.

For example, if you want to move the contents of A0 to D0, the proper mnemonic is MOVE.L A0,D0. The opword for this instruction is 2008. Note that when dealing with register-to-register transfer of data, byte moves are not allowed. The EXG instruction exchanges the contents of the specified registers.

Implied Addressing

Many instructions do not need to have the addressing modes specified. This type of addressing is called implied addressing. For example, the Branch always (BRA) instruction always uses the PC register and the PC need not be designated each time.

Indirect Addressing

Many variations of the indirect addressing modes are implemented in the 68000:

1. Address register Indirect
2. Address register Indirect with Postincrement
3. Address register Indirect with Predecrement
4. Address register Indirect with Displacement
5. Address register Indirect with Index

When using indirect addressing it is assumed that the address register contains the address where the data is located and/or where the data are to be placed. If you want to load the D0 register with the contents of memory locations $1500 and $1501, the assumption is made that A0 is loaded with $1500. To use address register indirect the following mnemonic is used.

MOVE.W (A0),D0

If the address location $1500 has stored in it AA, then bits 15-8 of D0 will be loaded with AA. Note that the contents of address $1501 will be loaded into bits 7-0 of D0.

Address Register Indirect with Postincrement and Predecrement

Many times it is important in a program that an address be either incremented or decremented from a previously established value. This is particularly true when tables of numbers or other types of tabular data are being searched. In either the post-increment mode or predecrement mode the value of the designated address register is considered to be the base value.

When using the postincrement mode of addressing the base address register is incremented *after* the base address is used. The predecrement mode decrements the base address register and then uses it to point to the desired address. The amount the register is incremented or decremented depends on the size of the operand. Byte increments/decrements by 1, word by 2, and long word by 4. The stack pointer is always incremented/decremented by 2 or 4 to insure that stack pointer stays on a word boundary.

The examples below illustrate the use of the postincrement addressing mode to load data when the size of the word changes from word length data, to byte length data and finally to long word data.

		Memory	
A. MOVE.W (A0)+,D0 (3018) Opword		Address	Data
Before execution	After execution	1500	0F
A0 00001500	A0 00001502	1501	01
D0 00000000	D0 00000F01	1502	02
B. MOVE.B (A0)+,D0 (1018) Opword		1503	03
Before execution	After execution	1504	04
A0 00001500	A0 00001501	1505	05
D0 00000000	D0 0000000F	1506	06
C. MOVE.L (A0)+,D0 (2018) Opword		1507	07
Before execution	After execution	1508	08
A0 00001500	A0 00001504	1509	09
D0 00000000	D0 0F010203		

Note from this example that the base register A is incremented once for a byte transfer, twice for a word transfer, and four times for a long word transfer. The base register for the predecrement mode is handled in a manner similar to the postincrement mode in that the base register is decremented once for byte data, twice for word data, and four times for long word data.

To illustrate the nature of the predecrement mode of operation, consider an example similar to the previous one.

		Memory	
A. MOVE.W -(A0),D0 (3020) Opword		Address	Data
Before execution	After execution	14F5	F5
A0 00001500	A0 000014FE	14F6	F6
D0 00000000	D0 0000FEFF	14F7	F7
B. MOVE.B -(A0),D0 (1020) Opword		14F8	F8
Before execution	After execution	14F9	F9
A0 00001500	A0 000014FF	14FA	FA
D0 00000000	D0 000000FF	14FB	FB
C. MOVE.L -(A0),D0 (2020) Opword		14FC	FC
Before execution	After execution	14FD	FD
A0 00001500	A0 000014FC	14FE	FE
D0 00000000	D0 FCFDFEFF	14FF	FF
		1500	0F

The automatic incrementing and decrementing features of the 68000 make the movement of data tables in the memory of the 68000 a relatively straightforward problem. For example, if you want to move data from a table, A0 would be set to the low address of the original table and A1 would be set to the low address of the new table, an instruction MOVE.W (A0)+, (A1)+ would be executed in a loop until the appropriate number of data words were moved. Note the data of the table can be reordered relatively easily by allowing one of the base address registers to predecrement.

Address Register Indirect with Displacement

Many times it is necessary to retrieve data from a fixed location in a table. The address of beginning or end of the table is loaded into an address register. The fixed displacement, either positive or negative, into the table from the address register is incorporated in the opcode for the address register indirect with displacement.

The displacement is expressed in two's complement form, and thus the effective address can be displaced either up or down from the address established in the base address register. At the end of the execution of this statement the base address register is left unchanged.

The example below illustrates the use of address register indirect with displacement. In this example D0 is to be loaded with data located six locations away from the base address register A0.

		Memory	
		Address	Data
MOVE.W 6(A0),D0 (3028) Opword		1504	04
(0006) Postword		1505	05
Before execution	After execution	1506	06
D0 000000000	D0 00000607	1507	07
A0 00001500	A0 00001500	1508	08

If the data was to be found in a location 6 less than the base register ($1500), the instruction would have been written with the displacement in two's complement form.

		Memory	
		Address	Data
MOVE.W -6(A0),D0 (3028) Opword		14F7	F7
(FFFA)		14F8	F8
Before execution	After execution	14F9	F9
D0 00000000	D0 0000FAFB	14FA	FA
A0 00001500	A0 00001500	14FB	FB
		14FC	FC
		14FD	FD

The offset cannot place the memory at an odd address. For example, the offset cannot be $5 in the previous example.

This information was compiled with the assistance of Motorola, Inc.

Addressing Modes will be continued next month.

You may contact Professor Hootman at the University of North Dakota, Dept. of Electrical Engineering, University Station, Grand Forks, ND 58202.

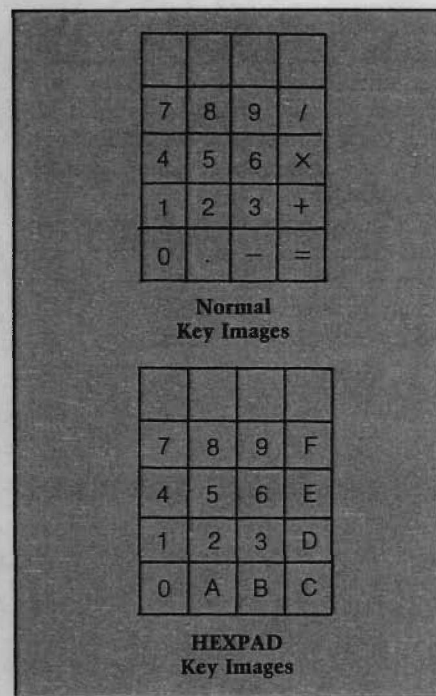
MICRO

HEXPAD: UTILITY FOR MACHINE LANGUAGE KEY-INS

This utility redesigns the PET calculator keypad. Use it when you need to type in a published machine-language program.

by Bob Sullivan

HEXPAD is a fifty-six byte tool that converts your PET calculator pad into a hexadecimal key pad:



This utility is designed to be used while entering data *via* the machine-language monitor. It has been used successfully on 4032 and 2001 PETs to type in and save a relocated version of itself. Published machine-language programs such as Micromon, Basic-Aid, or Face can be more conveniently entered with this handy software tool.

HEXPAD is "wedged" into the hardware interrupt routine. It checks to see if the last key entered is a target key or key image to be changed. Whenever a target key is entered, a cursor left and replacement image are immediately printed.

Entering and Running HEXPAD

Use your monitor or assembler to enter the hexadecimal code for HEXPAD or use your assembler. After you have entered the code, SAVE it to tape or disk. Then check the contents of addresses \$90-\$91.

```
.M 0090 0091
.: 0090 55 E4 XX XX XX XX XX XX
.■
```

If the first two locations don't contain 55 and E4, take note of them, since you must make changes in the HEXPAD.

(Continued on page 92)

Enter 00 and 10 (the starting address of HEXPAD) and press RETURN.

Unless you have a different IRQ address, you are now ready to use HEXPAD. Each time you use the program, you must change the contents of $90,$91 to contain the starting address of the HEXPAD routine. Each time you switch back to BASIC you must disable HEXPAD by replacing the IRQ vector. (If you don't, you'll continue to get 'A' when you type a period, etc.)

If the current IRQ address is not $E455, then line 260 or the object code in line 450 must be changed to aim at the current IRQ address — or else crash:

```
Line 260 IRQ       .DE $E455 (4032)
                   ($E62E-3.0 ROMS)
Line 450 IRQ.JMP   JMP IRQ
                   (4C 55 E4)
```

Locating HEXPAD

Relocating HEXPAD by changing the object code in lines 470 and 480 relative to the new address:

```
Line 470  JSR KEYCHK2  (20 1C 10)
Line 480  JMP IRQ.JMP  (4C 13 10)
```

Or change the beginning assembly address in line 210:

```
Line 210  .BA $1000
```

The Program

Conditions

Lines 330 to 470 include three conditions that must be met before a key image is checked. First, the program checks the column variable in zero page address $C9. If the cursor is not yet in the tenth column then an image change is not needed. Also, this eliminates problems with those periods that the monitor types in column one. Next, if any keys other than 0 to 9 have not been pressed then the program jumps to the KeyCheck subroutine.

KeyCheck

Lines 520 to 580 compare the last key entered (the ASCII code is kept as a variable in zero page address: $D9) with the values in 'Table' (lines 740 to 790). A match causes a branch to the NewKey subroutine.

NewKey

Lines 610 to 670 print a cursor left. Next, the table is set up so that the Y-register increment plus #$3F gives the ASCII value for the desired replacement image. With this value in the accumulator, a JSR to @WRT ($FFD2) will print the replacement image. The program concludes each time by jumping to the normal IRQ address.

Special thanks to Brent Anderson for helping me get started with ASSM/TED and MAE, and to Jim Strasma for initiating ATUG, which provides assembly-language examples.

The author may be contacted at P.O. Box 2247, Oak Park, Illinois, 60301.

MICRO™

Parameter Passing in Assembly Language

Part 2

by Randall Hyde

Passing Parameters *via* the Return Address

The most convenient way to pass constants to a subroutine is to follow the JSR instruction with the parameters. A prime example is the PRINT subroutine, which prints a string to the output device. On the 6502, PRINT is usually called in the following manner:

```
JSR   PRINT
BYT   "PRINT THIS STRING",0
```

The PRINT subroutine would send each character that follows the JSR PRINT statement to the console output device. Normally such a construct would not be allowed in assembly language. After all, on return the 6502 microprocessor would attempt to execute the ASCII character "P" as an instruction code. With a certain amount of trickery this problem can be avoided. Consider the following 6502 subroutine:

```
PRINT    STA ASAVE      ;Save Acc value
         STY YSAVE      ;Save Y reg val
         PLA            ;Get LO byte of
                        ;rtn adr
         STA ZPAGE      ;and save in a
                        ;zero-page loc
         PLA            ;Get HI byte of
                        ;rtn adrs
         STA ZPAGE+1    ;and save in a
                        ;zero-page loc
;
         INC ZPAGE      ;Add one to the
                        ;rtn address
         BNE  1         ;so it points at the
         INC ZPAGE+1    ;true rtn addr
 1       LDY #0
PRTLOOP  LDA (ZPAGE),Y  ;Fetch each
                        ;character until
         BEQ ALLDONE    ;a zero byte
                        ;is found
         JSR PUTC       ;Print character
                        ;character
         INC ZPAGE      ;Move to the next
                        ;character
         BNE PRTLOOP
         INC ZPAGE+
         JMP PRTLOOP
;
; Zero byte detected. Use this address as the
; new return address
;
ALLDONE  LDA ZPAGE+1
         PHA
         LDA ZPAGE
         PHA
         LDA ASAVE      ;Restore acc
         LDY YSAVE      ;Restore Y reg
         RTS            ;Return to loc
                        ;just past zero
                        ;byte
```

Whenever you jump to a 6502 subroutine, the return address left on the stack is the address of the next instruction, *minus one*. By POPping the return address off the stack and adding one to it you have a pointer to the data that follows the JSR statement. In the previous example this pointer was used to fetch the characters one at a time until a zero-terminating byte was encountered. Once the zero byte is encountered the pointer to the zero byte is pushed back onto the 6502 stack to be used as the new return address. Since the 6502 expects the true return address minus one to appear on the stack, upon executing the RTS instruction the 6502 continues processing at the instruction immediately past the zero byte.

The PRINT subroutine uses variable length parameters. In this case the end of the parameter list was specified by a special zero byte. Actually, any value can be used to terminate the parameter list as long as that value doesn't appear in the parameter list. Any time a parameter list contains a variable number of parameters, the subroutine being called must be informed as to how many parameters appear on the list. This can be done in one of several ways: you can use a terminating byte (like the PRINT subroutine does) or you can pass to the subroutine some indication of how many parameters are present on the line.

Sometimes a fixed number of parameters will be passed after a JSR instruction. These types of parameters are handled easily. For example, consider the subroutine used to transfer one zero-page location to another:

```
ZTRANS STA ASAVE
       STY YSAVE
       STX XSAVE
       PLA
       STA ZPAGE
       PLA
       STA ZPAGE+1
;
; Get data pointed at by first parameter
; and store in second.
;
```

```
        LDY #1
        LDA (ZPAGE),Y
        TAX
        INY
        LDA (ZPAGE),Y
        TAY
        LDA $0,X
        STA $0,Y

        CLC
        LDA ZPAGE
        ADC #2
        TAY
        LDA ZPAGE+1
        ADC #0
        PHA            ;Push HI rtn adrs
        TYA
        PHA            ;Push LO rtn adrs
        LDA ASAVE
        LDY YSAVE
        LDX XSAVE
        RTS
```

To use this subroutine simply enter the code:

```
        JSR     ZTRANS
        BYT ZPG1,ZPG2
```

and ZTRANS will copy the zero-page memory location pointed at by ZPG1 into the memory location pointed at by ZPG2. (*Note:* This particular routine is for educational purposes only. The "LDA ZPG1/STA ZPG2" instruction sequence is both faster and shorter and performs the same function.) While ZTRANS isn't a very useful subroutine, it certainly demonstrates how you would pass a fixed number of parameters after the JSR statement.

The 6809, 68000, and 16032 offer additional addressing modes that make picking up parameters after the JSR extremely easy. PRINT coded in 6809 code would look like this:

```
PRINT    PSHS A,X      ;Save 6809 regs
PRTLOOP  LDA  [3,S]
         BEQ  ALLDONE  ;Done yet?
         JSR  PUTC     ;If not, output it
         INC  3,S      ;Increment to
                       ;the next char
         BNE  PRTLOOP
         INC  4,S
         BRA  PRTLOOP
;
ALLDONE  INC  3,S      ;Increment to
         BNE  RTN      ;the true rtn
         INC  4,S      ;adrs
RTN      PULS A,X
         RTS
```

The PRINT subroutine coded in 68000 code is

```
PRINT    MOVEM.L D0/A1,  ;Save D0 and
                 -(SP)   ;A1
         MOVE.L  12(SP), ;Get rtn adrs
                 A1
PRTLOOP  MOVE.B  (A1)+,  ;Get char to
                 D0      ;print
         BEQ     ALLDONE
         JSR     PUTC
         BRA     PRTLOOP
;
ALLDONE  ADDQ.L  #1,A1   ;Fix rtn adrs &
         MOVE.L  A1,12   ;store back on
                 (SP)    ;the stack
         MOVEM.L (A7)+,  ;Restore
                 D0/A1   ;registers
         RTS
```

The 16032's stack architecture makes performing the PRINT subroutine a relatively simple task. The code for the 16032 PRINT routine is

```
PRINT    SAVE    [R0]     ;Save affected
                          ;registers
PRTLOOP  MOVEB   0[4[SP]] ;Get byte to
                 ,R0      ;print
         CMPQB   #0,R0    ;Check for end
                          ;of string
         Beq     ALLDONE
         JSR     PUTC     ;Send the char
                          ;to the console
         ACBD    #1,4     ;Increment
                 [SP],    string
                 PRT-     pointer
;                LOOP     ;and repeat
;                         ;loop
ALLDONE  ADDQD   #1,4     ;Fix return
                 [SP]     ;address
         RESTORE [R0]     ;Restore the
                          ;affected regs
         RTS     0        ;Rtn to caller
```

The MOVB 0[4[SP]],R0 instruction takes the value in the stack pointer and adds four to it. This sum points at the return address for the PRINT subroutine (pushing R0 onto the stack added four bytes to the top of the stack). The data at this address (the PRINT return address) is added to the first value, zero, and the byte at this address is fetched and moved to R0. The ACBD instruction adds one to the return address and then branches to the beginning of the PRTLOOP loop.

Passing parameters of fixed length after the JSR on the 6809, 68000, and 16032 is simple; this exercise will be left to the interested reader.

Different Methods of Passing Data

Once the mechanics of parameter transfer are mastered, learning how to pass different types of parameters becomes important. Most of the examples presented thus far (with the noted exception of passing the address of a parameter block) passed their parameters *by value*; i.e., the actual data to be used was passed to the subroutine. Three other forms of parameter transfer are pass by value returned, pass by reference, and pass by name.

Passing parameters by value is easy, fast, and doesn't affect variables in the calling procedure. Most pass-by-value parameters are stored in a temporary memory location (or some processor register) during execution of the subroutine. After execution of the subroutine the memory location used to store the parameter often is used for some other purpose. The value contained in the parameter after the execution of the subroutine is lost and cannot be passed back to the calling subroutine. When a subroutine needs to return data to the calling procedure within one of the parameter variables passed to it, then pass-by-value parameters are inadequate.

To return data within a parameter variable you must pass the address of the variable to the subroutine instead of the value it contains. In one of the previous examples where the address of a parameter block was passed, we did just that. The address was used as a pointer to access data in the parameter. Some sort of indirect addressing mode was used to read data from the parameter block for usage within the subroutine. By storing data into this parameter block (again using an indirect addressing mode of some type) data can be passed back to the calling procedure.

There is one problem with passing variables by reference. As an example, consider the following Pascal procedure:

```
PROCEDURE MIXEDUP(VAR I,J:INTEGER);
BEGIN

   I  := 5;
   J  := 6;
   WRITELN('I + J = ',I + J);

END'
```

Regardless of the input data, you would expect this routine to print "I+J=11" out to the console device. Since I and J are passed by reference (because of the VAR reserved word in the parameter list) the variables I and J actually contain addresses, not data. So any time you access I or J within the procedure MIXEDUP you are actually accessing the memory location pointed at by I or J.

Now suppose you invoke the procedure MIXEDUP with the call "MIXEDUP(L,L)". The parameter I would contain the address of L in the calling procedure as would the parameter J. Upon executing the statement "I=5;" the procedure MIXEDUP would store the value five in the memory location pointed at by I, which is the variable L in the calling program. Upon executing the statement "J = 6;" the procedure MIXEDUP would store the value six in the memory location pointed at by J, which is also the variable L in the calling procedure. Now the variable L contains the value six.

When the expression "I + J" is evaluated, the Pascal procedure will read the value from the memory location pointed at by I (which is L in the calling program), add in the value contained in the memory location pointed at by J (which is also L in the calling program), and print the sum. Since both I and J point at L and the variable L currently contains six, the sum will be twelve. So "I + J = 12" will be printed instead of "I + J = 11"! This is one of the major drawbacks to passing parameters by reference.

To overcome the problem demonstrated in this example, many high-level languages (such as FORTRAN) use a parameter-passing technique known as *pass by value returned.* This parameter-passing technique is a combination of the pass-by-value and pass-by-reference methods. In a pass by value returned, the address of a parameter is passed to the procedure and then the procedure copies the data at that location into some temporary storage location. During the execution of the procedure, any references to such parameters use the temporary storage locations. After the execution of the procedure (but before control is returned to the calling procedure) the value in the temporary location is copied back into the parameter variable whose address is passed. While this method solves the problem found when passing parameters by reference, it is quite inefficient in terms of storage and execution time. Pass by value returned should be used only when absolutely necessary.

The last method discussed here for passing parameters to a subroutine is the *pass-by-name* method. When using the pass-by-name parameter access method, you pass a string containing the name of the variable you wish to access. To retrieve the data stored in the variable you must look up the variable name in a table. Once the variable name is found, some auxiliary information is obtained that provides you with the data you are interested in or tells you where you can find it.

Pass-by-name parameters are used by assemblers, BASIC interpreters, and compilers for symbol-table manipulation. They rarely would be used in a typical assembly-language program, but the method is presented here for the sake of completeness, since the more advanced assembly-language programmers are likely to come across this type of parameter passing now and then.

Passing Parameters in Your Assembly-Language Programs

As lengthy as this article is, it's only a small treatise on the subject of parameter passing. A considerable amount of time could be spend discussing in more detail the parameter passing techniques mentioned here. Additional topics, like returning function values, could also be discussed. For more information on passing parameters a good reference book of data structures (like Knuth, Vol. 1) is highly recommended. Beyond that, only a lot of experimentation will help you to nail down all the techniques involved in passing parameters to assembly-language subroutines.

You may contact the author at Lazer MicroSystems, Inc., 1791 Capital Unit G, Corona, CA 91720.

MICRO™

Interface Clinic

by Ralph Tenny

An interface component that was mentioned only briefly in an earlier column is the *latch*. A latch was shown in a circuit designed to capture microprocessor bus data "on the fly" — directly from the bus. Latches have certain special features you need to know. First, there are two kinds of latches: *edge-triggered* and *transparent* latches. An edge-triggered latch will capture data only when a positive-going clock signal is applied to the part; transparent latches have a data input and an enable line. When the enable line is active, the latch output copies the logic state on the data line. When the enable line goes inactive, the latch locks up or captures the data present on the data line. The read/write memory in your computer works the same way; when the R/W* line is low, data from the bus is gated through to the memory cells and the data is captured when R/W* goes high.

All latches have certain critical timing parameters in common: *setup time, hold time,* and width of *clock pulse* or *enable strobe.* Setup time specifies how long the data must be present before the clock or enable strobe changes state and ranges from .2 microseconds for CMOS latches to .02 microseconds for TTL latches. Hold time specifies how long the data must remain valid *after* the strobe changes state. CMOS hold times are about .12 microseconds and TTL times are about .005 microseconds. Clock or enable pulse widths range from .2 microseconds for CMOS to .04 microseconds for TTL. Generally, setup and hold times must be taken into account only when parts are driven directly by the microprocessor bus; if the parts are driven by a PIA, timing limits are met easily since the PIA cannot be programmed to change quickly enough to cause timing problems. Figure 1 shows the typical timing waveforms for edge-triggered and transparent latches with the critical timing parameters identified.

Another interface component I have ignored so far is the *shift register.* Figure 2 shows the simplest form of shift register; it is a collection of latches connected in series so that the data output of the first stage drives the

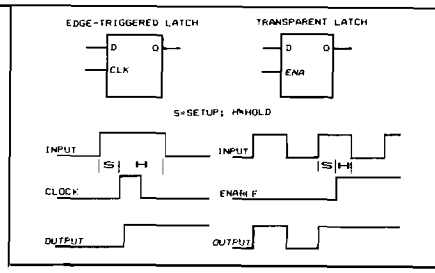


Figure 1: ***A comparison of edge-triggered and transparent latches showing waveforms and critical timing parameters.***

data input of the second stage and so on to the end of the chain. All stages are clocked at the same time, so a logic one clocked into the first stage comes true in the second stage after the second clock pulse. With this kind of logic block it is possible to convert a serial data stream into a collection of parallel bits or data word.

The serial (printer) port on most computers works with a communications protocol known as asynchronous ASCII, which means that the individual groups of bits known as words can be sent and received on an irregular schedule. This is accomplished by the scheme illustrated in figure 3, which defines a single serial word as it might be sent to a printer. In the illustration, a data word has been divided into eleven segments or bits. All words start with the RS-232 Out line low (RS-232 logic one). The first bit is *always* logic zero, and is called the *start* bit. The start bit serves as a timing mark for the receiving circuit so it can know when the data word is coming. The receiving circuit begins counting time and tests the input line 1½ *bit times* later, recording whether this bit is logic one or zero. Seven more tests or samples are made at one bit-time intervals, with the logic value of each being recorded. Bits 10 and 11 are called *stop* bits and are always logic ones. In continuous transmission, the last stop bit will be followed immediately by a start bit for the next word; during asynchronous transmission, the next word can be sent anytime.

What I have discussed is one common version of an asynchronous ASCII word; some versions will have only one stop bit, usually at the higher transmission rates. In order for an asynchronous scheme to work properly, both the sending and receiving circuits must be set for the same transmission rate, which normally is expressed as a *baud* rate. The definition of baud rate is that one baud is equal to one bit per second. Typical transmission rates are 300 baud, 600 baud, and 1200 baud; the Radio Shack standard is 600 baud to match Radio Shack printers.

In contrast to asynchronous transmission, some computer communication uses synchronous transmission, where special data-bit patterns signal start of data, end of data, etc. Inherent in synchronous transmission is a clock signal that signals the receiving circuit when to sample to read a bit's logic value.

Last month I used the printer port to send and receive single-bit information. The output level was held steady

Figure 2: ***A simple shift register can be constructed by connecting a series of edge-triggered latches and using a common clock input.***

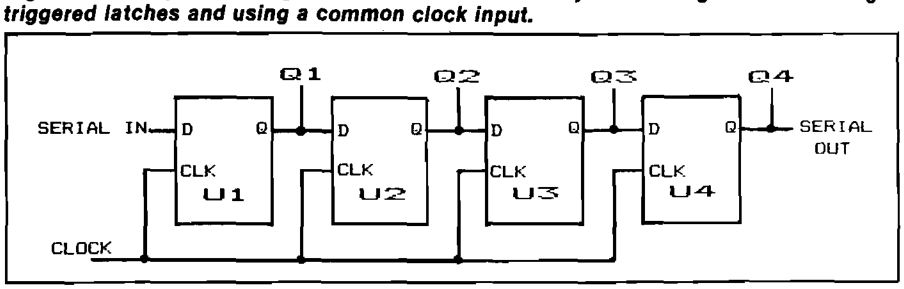
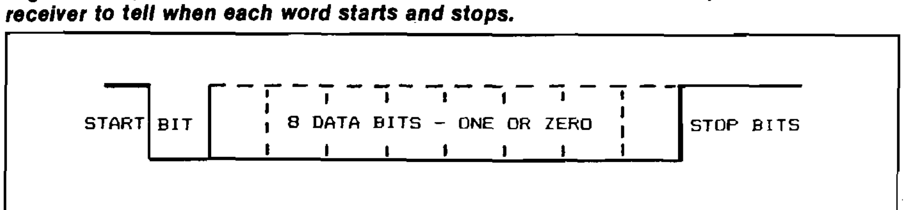


Figure 3: ***Asynchronous serial transmission must use start and stop bits to allow the receiver to tell when each word starts and stops.***



to indicate a logic one or zero, and the computer sampled the input line to determine if an external switch was open or closed. Since not much can be accomplished with single input and output bits, I have designed a scheme to handle more bits.

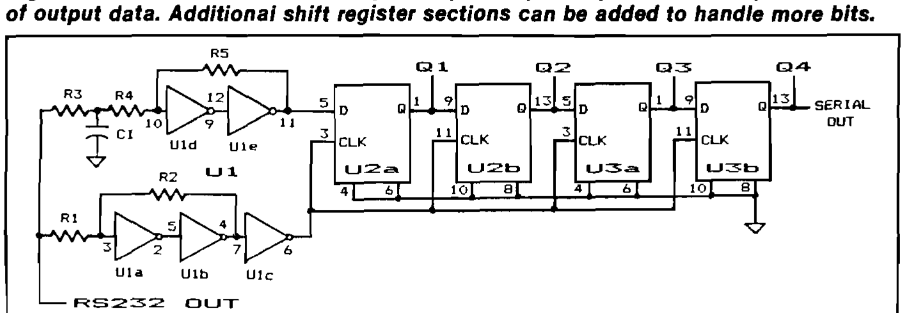
For this month's experiment I developed a scheme to output multiples of four bits, depending on how many sections I want to hook up. The first hurdle to face on the Color Computer is that there is only one output line; this limitation will probably apply on any computer that uses software to generate a serial data stream. Although it is possible to generate "home brew" circuitry that will receive standard asynchronous signals, I use somewhat simpler circuitry to implement a self-clocking scheme working in four-bit data blocks.

The circuit in figure 4 uses a delay scheme to encode both data and clock on the RS-232 Out signal, and a four-stage shift register captures the data. The circuit shows how to make a shift register from the CD4042 4-bit latch.

U1a and U1b work together to produce very fast signal transitions from the somewhat slower RS-232 Out signal as shown in figure 5. With the input low, U1a's output is high and U1b's output is low. As the input signal rises, R1 and R2 reduce the signal change at the input of U1a, but eventually U1a's input will reach the IC's signal threshold and U1a's output will begin to change, forcing U1b to change also. As the input continues to rise, U1b's output will go high until U2 is helping pull up on U1a's input rather than slowing the input rise. The graph in figure 5 shows the result — a slow level change at the input gives a fast change at the output. This type of circuit is called a Schmitt Trigger and is useful for interfacing slow signals to computers. U1c inverts the clock signal to provide the proper clock timing.

The network consisting of R3 and C1 acts to delay signal transitions reaching U1d and U1e, which are also connected as a Schmitt Trigger. If the RS-232 line goes high for only a short time, the RS/C1 delay into U1d and U1e prevents the data line from changing, but when the signal stays high long enough, the data line goes high. In either case, when the RS-232 line goes low again, C1 is discharged by R3 and a new cycle can begin. If a latch is connected to both the data and clock lines as shown in figure 4, it will capture either a logic one or a logic zero, depending on whether the RS-232 line stays high a long or short time. This is shown in figure 6, which shows the response of the circuitry in figure 4. The top line is an input from the RS-232 port, and the responses of the data and clock circuits are shown in the next two lines. Only the second input pulse is wide enough for U1d/U1e to change state, and this is reflected by the short period of logic one in the data signal. Meanwhile, the transitions of the clock signal marked with an arrow show when U2 will sample the data line. Assuming that Q1 of the shift register was high at the start, the logic zero of the input signal causes a transition to zero on the first clock pulse. The next data bit sent was a one (long positive pulse), and the second clock pulse captures a one. The third bit sent was a zero, which is captured by the third clock pulse. In order for this circuit to be useful, the RS-232 line must send four bits and stop until the next time data has to be sent. Also, you may need driver circuitry such as the LED driver used last month, one driver for each bit stored in the shift register.

Figure 4: ***This circuit can be driven by a computer's printer port and will capture four bits of output data. Additional shift register sections can be added to handle more bits.***



Parts List for Experiment #2

U1 — CD4049 (Radio Shack #276-2449)
U2, U3 — CD4013 (Radio Shack #276-2442)
R1, R4 — 82K ohm, ¼-watt resistor
R2, R4 — 330K ohm, ¼-watt resistor
R3 — 22K ohm, ¼-watt resistor
C1 — 1 μF, 16-volt capacitor (Radio Shack #272-1419)
Power Supply — +5 volts to +10 volts (battery-suitable)

The circuitry in figure 4 can be exercised with the short subroutines shown below. These routines should be called by a program that breaks an output pattern into individual bits, counts the bits as they are output, and then stops transmission.

After a bit of reflection, I renege on one comment I made last month: I said that I would specify parts and materials to be used in projects only as the need arose. I now believe our purposes can be better served by listing materials that you can watch for, possibly saving money by finding items on sale. In particular, Radio Shack often has sales that allow good savings if you can anticipate future needs. When I choose parts for an experiment, you might already have the needed parts and can proceed immediately. The following listed items will be useful for various hardware experiments. You can collect these items gradually or get them as needed (Radio Shack part numbers shown).

Discrete Components

Aluminum Electrolytic assortment: 272-604 or 272-605
¼-watt resistor assortment: 271-602
Prototyping boards: 276-170, 276-158, 276-153
Transistor assortments: 276-1603, 276-1604
LED assortment: 276-1622
Silicon signal diodes: 276-1620 or 276-1122
Silicon rectifiers: 276-1101

(continued)

Interface Clinic *(continued)*

Measurement Instruments (Optional, but useful)

Volt-ohmmeter: 22-204, or Digital
VOM: 22-197
Logic probe: 22-301

Figure 5: ***Two resistors and two CMOS inverters make a circuit that generates fast logic transitions from a slow input signal.***

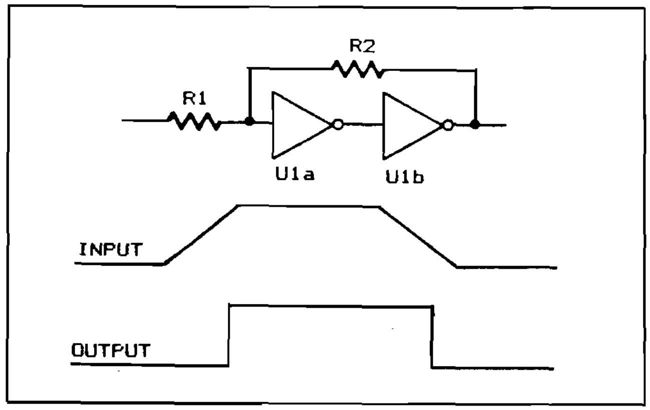
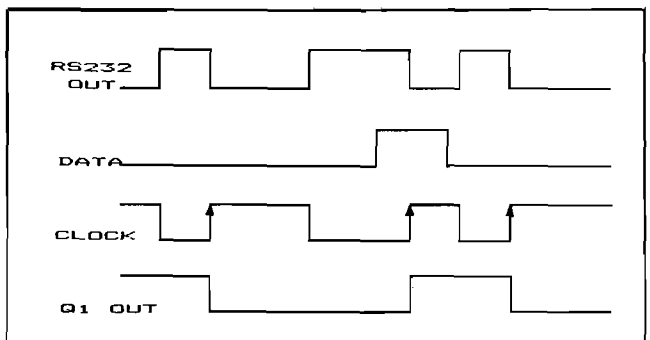


Figure 6: ***Timing diagram for circuit in figure 4. The input signal encodes both data and clock by varying width of output pulses.***



Interface Clinic Listing

```
                       * THIS PROGRAM WILL DRIVE A SERIAL PORT ADAPTER.
                       * CALL THAT SUBROUTINE WHICH OUTPUTS EITHER A ONE
                       * OR ZERO AS NEEDED.  THE CALLING ROUTINE MUST COUNT
                       * OUTPUT BITS AND HANDLE THE OPERATOR INTERFACE.
                       * THE USER MUST DEFINE THE ORIGIN; $2800 USED AS
                       * AN EXAMPLE.
                       * EQUATES
                  FF20 RS2320  EQU   $FF20     RS-232 OUTPUT BIT = BIT 1
                  0002 BITHI   EQU   $02       SET BIT 1 HIGH
                  00FD BITLO   EQU   $FD       SET BIT 1 LOW
                  0500 LONG    EQU   $0500     TIME CONSTANT FOR LOGIC ON OUTPUT
                  0280 SHORT   EQU   $280      TIME CONSTANT FOR LOGIC ZERO
                       * SUBROUTINES
2800                           ORG   $2800
2800 B6   FF20         ONE     LDA   RS2320    READ PORT BIT PATTERN
2803 8A   02                   ORA   #BITHI    SET RS232 BIT HIGH
2805 B7   FF20                 STA   RS2320
2808 8E   0500                 LDX   #LONG     TIME DELAY FOR LOGIC ONE
280B 30   1F           COUNT1  LEAX  -1,X      COUNT DOWN
280D 26   FC                   BNE   COUNT1
280F B6   FF20                 LDA   RS2320    READ BIT PATTERN AGAIN
2812 84   FD                   ANDA  #BITLO    SET RS232 BIT LOW
2814 B7   FF20                 STA   RS2320
2817 8E   0280                 LDX   #SHORT    REMAINING BIT CELL TIME
281A 30   1F           CNTDN1  LEAX  -1,X      COUNT DOWN
281C 26   FC                   BNE   CNTDN1
281E 39                        RTS
281F B6   FF20         ZERO    LDA   RS2320    READ PORT BIT PATTERN
2822 8A   02                   ORA   #BITHI    SET RS232 BIT HIGH
2824 B7   FF20                 STA   RS2320
2827 8E   0280                 LDX   #SHORT    TIME CONSTANT FOR LOGIC ZERO
282A 30   1F           COUNT2  LEAX  -1,X      COUNT DOWN
282C 26   FC                   BNE   COUNT2
282E B6   FF20                 LDA   RS2320    SET RS232 BIT LOW
2831 84   FD                   ANDA  #BITLO
2833 B7   FF20                 STA   RS2320
2836 8E   0500                 LDX   #LONG     FINSIH BIT CELL TIME
2839 30   1F           CNTDN2  LEAX  -1,X      COUNT DOWN
283B 26   FC                   BNE   CNTDN2
283D 39                        RTS
                               END
```

Please forward questions and suggestions for discussion topics to Mr. Tenny at P.O. Box 545, Richardson, TX 75080.

MICRO

MICRO™

Annual Index

The following two pages are a continuation of MICRO's Annual Index (see MICRO 60:105 for the first installment). The list is comprised of articles that have appeared in MICRO over the past year and are placed under specific headings for easy reference. The first number indicates the issue and the second number the page of that issue.

(continued)

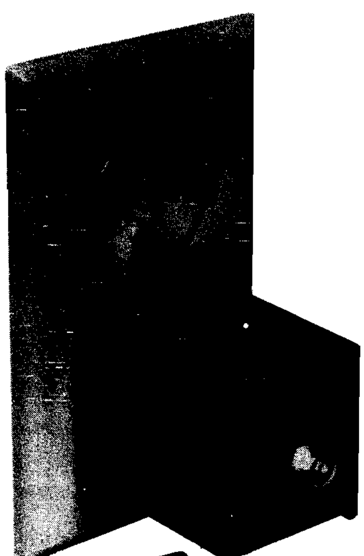
Annual Index *(continued)*

UTILITIES

MICRO

another **FLEXIDUCT**® OFFICE SAFETY PRODUCT

When you least expect it, ZAP!!!



In a few millionths of a second, common electrical surges and spikes can enter your data processing equipment and cause memory loss, false logic and misregistration. Surges very often do permanent damage to microcircuitry.

FLEXIDUCT Surge Suppressors catch surges and spikes before they have a chance to enter your equipment. In billionths of a second (Nanoseconds), **FLEXIDUCT** Surge Suppressors dissipate surges and spikes from any side of the line (most protect only one side).

Model FS-P plugs into the wall outlet to protect that outlet **and all other outlets on that circuit.** For safety, it is fused to protect from overloads.

No computer should be without the protection of a **FLEXIDUCT** Surge Suppressor...**especially yours!** Write or call for further information. Available from office products retailers.

FLEXIDUCT® Surge Suppressors

a product of Winders & Geist, Inc. P.O. Box 83088 Lincoln, NE 68501 402/474-3400

Circle No. 73

Commodore

Five new programs are available for CBM/PET and Commodore 64 computers. **Script Ease ($39.50)** is a 40-column word processor designed for all levels of word processing needs. Text can be output to the printer without any format commands; These commands can be added in subsequent lessons. Requires disk drive, printer, and 32K in the CBM/PET. A $5.00 demo is available (refundable).

Datalog ($39.50) lets you define up to 12 of your own fields and creates up to 1000 records on a 4040 disk, each record being over 200 characters long. Interfaces to popular word processors for printing form letters or labels. Multiple sort and search capabilities. Same requirements as Script Ease. A $5.00 demo is available (refundable).

Date Due ($39.50) manages overdue items in libraries. It prints reports by name, date, call number, title, or room number. Operates in 16K, 32K, or the "64". Requires disk drive and a printer is preferred. **Multiple Choice ($29.50)** creates up to 150 question and answer sets per disk file. Any number of questions can be selected from the bank and randomized if desired. The answer choices are always in random order. A test can be taken on screen or printed on paper. An answer key is printed down the left side of the paper. Supplied on disk only but can be saved and used on tape. Provides high security for teacher made tests. Runs in 16K, 32K, and the "64".

Ledger ($29.50) is a financial package for personal finance or school revolving accounts. It prints reports sorted by journal, ledger, or check register and handles up to 300 transactions in any number of accounts. It runs in 16K, 32K, or on the C64.

For more information contact Midwest Software, Box 214, Farmington, MI 48024; 313-477-0897.

Atari 800 Utility Cartridge

Monkey Wrench II is a ROM cartridge that plugs into the right-hand slot of your Atari 800. It provides 18 new BASIC commands and 16 machine-language commands. The new BASIC commands include - auto line numbering, renumber lines, delete range of lines, display all BASIC variables and values, scrolling up or down, find string, exchange string, copy lines, move lines, display disk directory, cursor exchange (no shift key needed), upper case lock, hex conversion, decimal conversion, and format program list (separates and prints to the screen or printer each command appearing on a BASIC multiple command line).

Price is **$59.95.** For more information contact Eastern House Software, 3239 Linda Drive, Winston-Salem, NC 27106; (919) 924-2889.

Inventory Management for the Commodore 8032

INFOTORY, an inventory management system now available for the Commodore 8032 Professional Computer, offers an item capacity of 4,000 inventory items with 38 fields of information for each item, including three pricing levels. INFOTORY provides users with information reporting capabilities through its data management sub-system, ANYREPORT. ANYREPORT allows users to design reports based on information associated with their inventory and the option to save those reports for future use.

Suggested retail price is **$425.00.** For more information contact S.S.R. Corp., 1600 Lyell Ave., Rochester, NY 14606; (716) 254-3200.

Programmer Utilities for Atari

AiDE (Absolute Disk Editor) for the Atari is a menu-driven system to simplify the disk managemant task. **Telecom** is a communications package that allows the Atari 400/800 to communicate with any and all other computers.

AiDE is **$29.95**; Telecom is **$69.95**. Both systems run on the Atari 400/800 home computer. Available from Roklan Corporation, 3335 N. Arlington Heights Road, Arlington Heights, IL 60004.

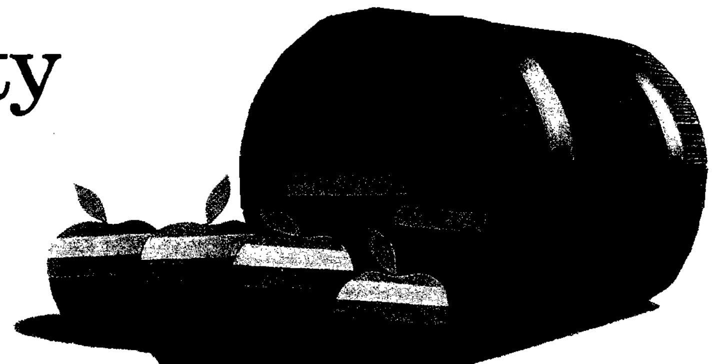
In Search of the Most Amazing Thing

In Search of the Most Amazing Thing, a combination adventure, strategy and arcade game, is designed to create an environment in which children are experiment and discover. The game places the child into a fantasy adventure where he/she must employ strategy and learning skills to discover the location and identity of The Most Amazing Thing.

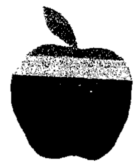
Children will learn decision making and organizing, note-taking and writing, map reading, trading and bargaining, music writing and drawing, and exploration.

In Search of the Most Amazing Thing is compatible with Apple, Atari, IBM, and Commodore-64 computers. Price is $39.95.

(Continued on page 111)



2
4



APPLE POSTERS
ORDER FORM

Electronic Report Card

The Report Card, for the Apple, tracks the progress of up to 300 students per diskette. The program calculates student and class averages and ranks students within their class. Exercises, quizzes and tests can be individually weighted for their effect on the final grade. The manual includes a reference section and a tutorial for ease of use.

Price is **$60.00** available from Sensible Software, 6619 Perham Drive, W. Bloomfield, MI 48033; (313) 399-8877.

Physics Packages for the Apple

Harmonic Motion Workshop, Projectile Motion Workshop, and Charged Particle Workshop are the first three in a series of packages being developed by a former college Physics professor. The Harmonic Motion Workshop visually presents the concepts associated with harmonic motion by use of high-resolution graphics. The student can alter such variables as the phase, amplitude, and damping factor and immediately see the effect. Projectile Motion Workshop is designed to illustrate projectile motion under the influence of a uniform force of gravity. In this program, the user may control the projectile's initial velocity and its angle of fire. Charged Particle Workshop simulates the motion of a charged particle under the influence of various combinations of electric and magnetic fields: uniform electric field, uniform magnetic field and crossed electric-magnetic field are illustrated.

The packages require a 48K Apple with Applesoft in ROM and Disk II. Price is $75.00 each. Available from your local computer retailer or High Technology Software Products, Inc., 1611 N.W. 23rd Street, P.O. Box 60406, Oklahoma City, OK 73106; (405) 525-4359.

Serpent Game for Atari

In **Serpentine** for the Atari 400/800, giant serpents set forth to slay their slithery cousins. The snakes lay eggs and fight ferociously to protect their young. A fast arcade-style game with five maze configurations and many levels of play.

Price is **$39.95.** Available from Broderbund Software, Inc., 1938 4th St., San Rafael, CA 94901; (415) 456-6424.

Apple Tax Planner

Tax Planner helps minimize personal tax liability through year-round planning. The pro-

(continued on page 112)

gram is designed for tax practitioners, estate planners and attorneys, plus banks and trusts who do income tax projections for clients. The Planner helps build client data bases for use during subsequent tax seasons and in serving more clients. Individuals who do their own financial planning will also find the Planner helpful.

User-modifiable parameter files for each year permit long-range tax-planning options. Up to four client data files can be handled at once — the first to hold current data, additional files to reflect assumed data changes. Up to four different assumption data files for a given year and tax calculations for four different years can be programmed.

Tax Planner cost is **$300.00** available from CPAids, 1061 Fraternity Circle Drive, Kent, OH 44240; (216) 678-9015.

Commodore Products

The **64-RABBIT** is a high-speed cassette interface on ROM cartridge that adds 12 commands to BASIC. The new RABBIT save, load, and verify commands operate five times faster than the normal BASIC commands. Other commands include load/run, test memory, decimal to hexidecimal conversion, hex to decimal conversion, and more.

RABBITs are also available for the VIC-20 and PET 4001,8001, and 8032. Price is **$39.95.**

MAE (Macro Assembler and Text Editor) is a professional development tool based on the well-known Apple-, PET-, and ATARI-MAE. Features include: 38 error codes, 27 commands, 26 pseudo ops, and 5 conditional assembly operators; built-in software UART (110-9600 baud; and macro, conditional assembly, and interactive assembly capability.

Text editor features include autoline numbering, find, exchange string, copy, move, renumber, delete lines, and append. Word processor features include right and left justify, center, set margins, define text body shape, headers, and footers.

All MAEs are priced at **$99.95.**

Our Software and hardware catalogs list the newest and most pertinent products on the market. The easy-to-read entries offer quick information for our readers. Use these catalogs to present your products. Send your press releases to:
MICRO INK
P.O. Box 6502
Amherst, NH 03031

MICRO™

New Publications

The Small Computer Connection, by Neil L. Shapiro. A Micro Text/McGraw-Hill Copublication, 1221 Avenue of the Americas, New York, NY 10020, 1983, 190 pages, paperback.
ISBN 0-07-056412-4 $15.95

Intermediate-Level Apple II Handbook, by David L. Heiserman. Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, IN 46268, 1983, 324 pages, paperback.
ISBN 0-672-21889-5 $16.95

The World Connection, by Timothy Orr Knight. Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, IN 46268, 1983, 142 pages, paperback.
ISBN 0-672-22042-3 $9.95

The McGraw-Hill Computer Handbook, by Harry Helms. McGraw-Hill Book Company, 1221 Avenue of the Americas, New York, NY 10020, 1983, 992 pages, Hard cover.
ISBN 0-7-027972-1 $79.50

6502 Assembly Language Programming, by Judi N. Fernandez, Donna N. Tabler, & Ruth Ashley. John Wiley & Sons, Inc., 605 Third Ave., New York, NY 10158, 1983, 277 pages, paperback.
ISBN 0-471-86120-0 $12.95

BASIC Exercised for the Atari, by J.P. Lamoitier. SYBEX, 2344 Sixth Street, Berkeley, CA 94710, 1983, 251 pages, paperback.
ISBN 0-89599-101-2 $12.95
plus $1.50 S/H

Doing Business with PASCAL, by Richard Hergert & Douglas Hergert. SYBEX, 2344 Sixth Street, Berkeley, CA 94710, 1983, 371 pages, paperback.
ISBN 0-89599-091-1 $15.95
plus $1.50 S/H

MICRO

MICRO™

Hardware Catalog

Atari Speech Synthesizer

VOICE BOX II for ATARI 400/800 computers requires a 32K disk system. It offers many features, including the abilities to speak with inflection, to speak in foreign languages with correct foreign spelling as input, and to sing with voice and 3-part music.

VOICE BOX II contains a library of 30 famous songs and a comprehensive music system that allows you to enter new songs easily.

Software provided can convert the bottom two rows of the Atari keyboard into a piano with a range of 3½ octaves using the shift and control keys. You can also have programmable musical sound effects such as tremolo, vibrato, and glissando.

Price is **$169.00.** For more information, contact The Alien Group, 27 West 23rd Street, New York, NY 10010; (212) 741-1770.

Voice Box II ***for*** **ATARI**
Speech Synthesizer with Singing and Inflection

Z80 Card for the Apple IIe

The new **Premium SoftCard IIe** from Microsoft Corporation fits into the video slot of the Apple IIe and apparently will double many of the standard features of this new 8-bit personal computer.

The Z-80-based card comes with the CP/M 80 operating system, giving users access to the broad range of applications programs available for the 8-bit microcomputer world. It will also give access to software development tools not developed specifically to work with Apple-DOS. Once installed, the user will specify which operating system by booting the appropriate Z80 or 6502A disk.

The card features 64K of random access memory, doubling the amount of memory standard with the Apple IIe. In addition, the card allows for an 80-column display.

Two versions of Microsoft's BASIC language interpreter are included. MBASIC supports the low-resolution graphics and GBASIC supports both the low- and high-resolution graphics.

Retail price for the card is **$495.** It comes complete with 400 pages of documentation including CP/M and Microsoft BASIC Quick Reference Guides, and the Osborne tutorial on CP/M.

For more information contact Microsoft, 10700 Northupway, Bellevue, WA 98006; (206) 828-8080.

Color Computer Keyboard

SUPER-PRO replacement keyboard kit for TDP-100 and Radio Shack Color Computers offers complete compatibility and operator convenience.

Price is **$69.95.** An additional plug adapter, priced at **$4.95,** is required when upgrading computers manufactured after approximately October, 1982.

Available from dealer or Mark Data Products, 24001 Alicia Parkway, No. 226, Mission Viejo, CA 92691.

Epson FX-80 Printer

The Epson FX-80, a high-performance bi-directional printer, combines a printing speed of 160 characters-per-second with features that provide sophisticated printing for a wide range of applications.

The FX-80 offers a software-selectable choice of elite (12 cpi) or pica (10 cpi) print spacing. In addition, users can send their own special fonts from their computer system to the printer, downloading the font into the printer's memory.

Other features include a one-to-one graphics ratio and a 2K-byte buffer, which allows buffered printing on longer productions. The FX-80 provides 9x9 dot-matrix characters with full descenders and is downward compatible with the Epson MX Series of printers. It offers proportional spacing, pin- and friction-feeds, and a standard parallel communications interface, with serial or IEEE 488 interfaces also available as options.

Normal, emphasized, bold and double-emphasized printing densities provide flexibility.

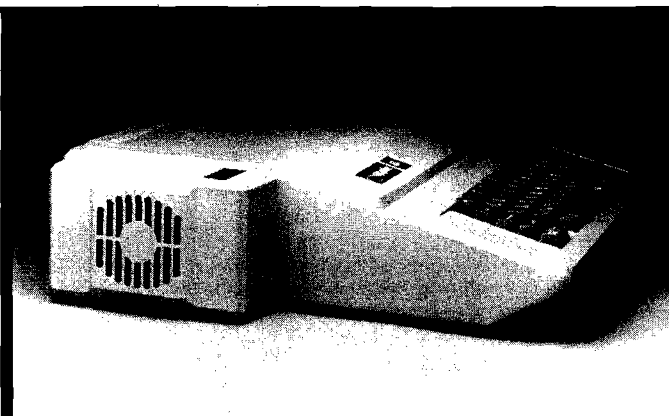
Price is **$699.00.** Available through Epson retailers. For more information contact Epson America, 3415 Kashiwa Street, Torrance, CA 90505; (213) 539-9140.

Atari I/O Package

The MOSAIC **I/O Package** can help give the ATARI computer direct ties to the real world. The four ports on the front of the ATARI computer connect directly to a PIA for use as output as well as input ports. Now ATARI owners can build custom program controllers, interface to home control circuits, or use any hardware the imagination can devise.

The I/O Package comes with four, nine-pin connectors, four, twenty-four-inch lengths of nine conductor ribbon cable, and complete documentation for their use.

Order number H-309, Price **$18.** Available from Mosaic Electronics, P.O. Box 748, Oregon City, OR 97045.



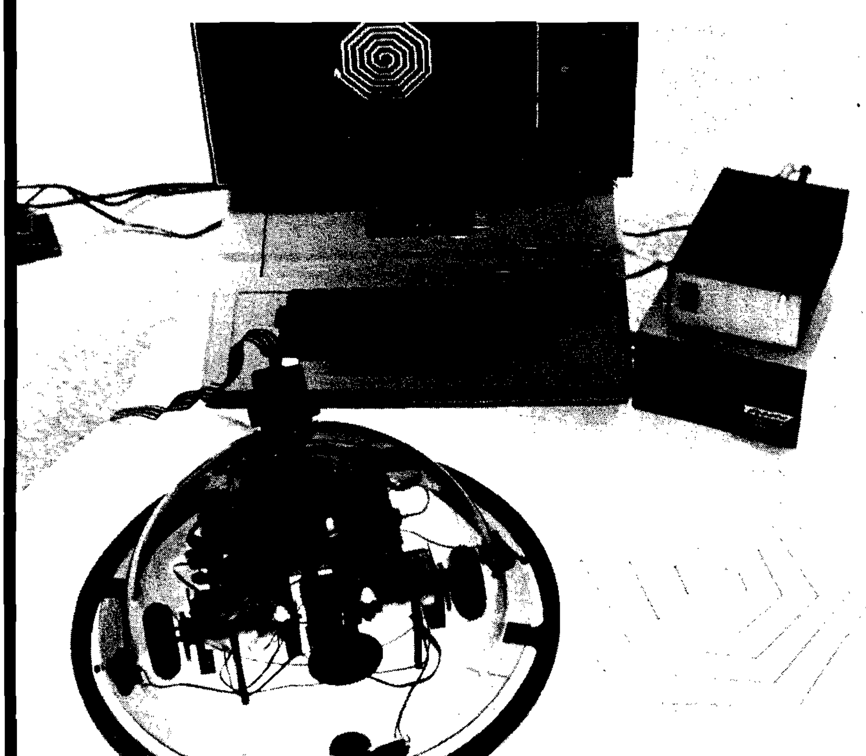
The Kensington Microware System Saver Power Control

Power Control for the Apple

Apple owners' high-tech power line problems are being solved a design approach that most electrical engineers would view as low tech using technology that has existed for decades. But the circuitry used in the company's UL-listed **System Saver** power control and protection accessory is effective and reliable, and inexpensive.

System Saver attaches to any Apple II, Apple II+, or Apple IIe, with or without an Apple monitor stand in place. It replaces the Apple power cord, providing simultaneous control and protection to the Apple computer itself, to one or two accessories plugged into auxiliary outlets (rated at 8 Amps total) on the back of the System Saver case, and to its built-in cooling fan.

Price is **$89.95**. For more information contact Kensington Microware Ltd., 919 third Ave., New York, NY 10022; (212) 486-7707.



The **Tasman Turtle,** a programmable robot, is available for the Apple II or other parallel interface micros as well as those with an RS-232 interface. All turtles can move and turn, toot their horn, blink their eyes, draw with a pen and "feel" through touch sensors.

A talking version of Tasman comes with a basic vocabulary of 150 words, expandable to more than 600 words. Another option is an electronic compass that indicates directional change.

Prices begin at **$999.95.** For more information contact Harvard Associates, Inc., 260 Beacon Street, Somerville, MA 02143; (617) 492-0660.

The Tasman Turtle interfaced to an Apple II

MICRObits

Enhanced OS65D 3.3 C1P-MF

Many new functions such as system commands for catalog control, 10 active files, end of file/end of volume processing, background printing, file append, dynamic file buffers/sectors command file processing, long string read command. Many more! $30. Write for details. Ray Lydon, 20 Eastwood Dr., Grafton, OH 44044.

OSI C1P Word Processor

Fast, Assembly Language Word Processor for 8K, Cassette-Based C1P or Superboard. Document storage on tape. Full screen edit. 9 Screens of text in 8K! Price: $30. 16K version supports 8-32K with additional features. Price: $35. Requires C1S Monitor ROM. Micro Chip's Software, 8104 NE 92nd Circle, Vancouver, WA 98662.

The Cheap Assembler

Includes unlimited length labels, free field programming, two pass RAM/disk based assembly, ten command text editor, interactive operation, tutorial manual and demonstration routines. APPLE II +, 48K, DOS 3.3 required. Send $20.00 + $4.00 P&H to: Thunder Software, P.O. Box 31501, Houston, TX 77231, 713-728-5501.

Franklin Color Adapter

There is no E-Z-ER method. Our assembled boards are an absolute must for your Franklin Ace 1000. No cutting, wiring, or soldering required. $50.00 + $2.00 S/H. N.Y. residents + 7%. DISPLAY SYSTEMS RESEARCH, 444 Front Street, Owego, NY 13827.

OSI Programs

Quality programs for all OSI systems. Games, Full Screen Editor, Machine Code Tracer. On tape or 5¼ in. disk for C1, C2, C4 systems. Write for catalog. Available from DMP Systems or our many distributors. DMP Systems, 319 Hampton Blvd., Rochester, NY 14612.

Payroll

Compatible with 64K Apple computers. Features: 100 employees per disk, Time card adding, automatic overtime, User changeable taxtables, five miscellaneous deductions - fixed or percentage amounts, tips, automatic printing of checks, W-2 forms, and reports. Bill Martin, 6485 Diana Drive #2, Poland, OH 44514, (216) 757-2143.

OSI - Game Set I

Two arcade games. Machine language (disk only) and Basic. Lunar Pro (Lunar Lander) with multiple screens, craft rotation, excellent graphics. Squeeze (ZAP) - defend your star-castle. Frustrating. Arcade graphics. C1pmf, C4pmf, C1p. 6.5K. $5 cassette/$12 disk ppd. Flxware, Suite 15c, 1150 Fifth Ave., New York, NY 10028.

AIM-65 Real Time Clock

Provides hour, minute, second, day of week, day, month, year. 12 or 24 hour format. Pin compatible with AIM expansion connector (also SYM, KIM). Four switch selectable interrupts. Nicad battery backup. Industrial quality board 4.5 x 6. All IC's socketed. Single 5V supply. 22 page manual. All software included. Bare board $29. Complete A&T $93, includes batteries. Add $4 ship and handling. Calif. residents add 6%. DATA DESIGN GROUP, P.O. Box 3318, La Jolla, CA 92038, (619) 265-6940.

MICRO

MICRO™

Reviews in Brief

Product Name: **Disk Data Handler**
Equip. req'd: TRS-80 Color Computer 32K Disk
Price: $44.95 + $1.00 Shipping
Manufacturer: Custom Software Engineering, Inc. (D-8)
807 Minutemen Causeway
Cocoa Beach, FL 32931

Description: *Disk Data Handler* is a database management system for the Color Computer. On-screen editing and high-speed record sort and selection is featured. The BASIC program with machine-language subroutines allows you to create fields and records that will fit your specific needs.

Pluses: The program is easy to learn and has powerful sort and select features. Both quick and extended files are available. Extended files are stored on disk and are retrieved by keyed quick files. Reports are easy to generate and report command files can be read that will format and print any desired report. The program will create files of selected data for additional processing by your BASIC programs.

Minuses: The disk-handling routines are not error trapped. Care must be used to specify correct file specifications otherwise the program will crash. The documentation does warn where this will happen. The report feature has many powerful aspects, but it lacks the ability to generate headings or pagination. No computation is possible in files.

Documentation: A 12-page instruction sheet is included that explains the operation of the program. Techniques of accessing and computing numeric data are included as is a Stock Tracker program.

Skill level required: A knowledge of data handling and file-creation techniques is helpful but not required.

Reviewer: John Steiner

Product Name: **Ultra 80CC**
Equip. Req'd: Color Computer with 32K memory
Price: $49.95 (Disk only)
Manufacturer: Spectral Associates
141 Harvard Ave.
Tacoma, WA 98466

Description: *Ultra 80CC* is an old friend in disguise for many 6809 users; it is TSC's editor and 6809 mnemonic assembler, adapted for the Color Computer. This software is used on most 6809 systems running Flex, and this adaptation makes fully professional software available for Color Computer owners. The adaptation includes tape read and write from the editor and printer output for both modules. (I/O is normally furnished by Flex.) Both modules operate from command lines similar to those used by most DOS packages, so the user who later upgrades to a DOS will already be familiar with this type of operator communication. The Editor is an exceptionally powerful content-oriented line editor with a full complement of edit, search, copy, and delete functions.. Full mastery of the editor's capabilities will take some time, but simple editing is quickly learned. The assembler has full macro and conditional assembly capability and ten assembly-time options including print-format options. Source files should be produced by the editor, but tape files probably can be read by the editor and written to disk. Object code is returned to disk and then the file can be accessed by the Radio Shack DOS.

Pluses: Exceptional quality at a very low cost. Short learning curve to get started, with reserve power as the user learns more about the programs. Very smooth operation with Radio Shack DOS and various debugger programs.

Minuses: None noted; some similar products offer a debug monitor at the same price, but the quality in this package makes it a bargain.

Documentation: The manual includes over 100 pages devoted to use of the two programs, 6809 assembly-languages procedures, and addenda detailing the special features in the CoCo adaptation.

Experience level required: Some assembly language experience will ease the learning process, but a diligent beginner should be able to use this package to good advantage.

Reviewer: Ralph Tenny

Product Name:	**Apple Mechanic**
Equip. req'd:	Apple II, 48K
Price:	$29.50
Manufacturer:	Beagle Bros. 4315 Sierra Vista San Diego, CA 92103

Description: This new Apple utility contains a shape-defining/manipulating program, a disk zap program, and a collection of "two liners."

Pluses: The shape program includes several character set fonts that do not require any extra drawing code. The Applesoft DRAW command is entirely sufficient. The program for modifying fonts is without question the best available.

Minuses: Not many. Perhaps the manufacturer should have used examples of greater educational value in the "Byte Zap" program user's manual.

Documentation: Well written, informative, and entertaining.

(continued)

Skill level required: Both the novice and the expert can benefit from this product.

Reviewer: Chris Williams

Product Name:	**Semi-Draw**
Equip. req'd:	TRS-80C 32K Extended BASIC
Price:	$21.95
Manufacturer:	Computerware Box 668 Encinitas, CA 92024

Description: *Semi-Draw* is a graphics development and sketching program for the Color Computer. Pictures can be drawn in three resolutions — Semigraphics 8, 12, or 24. Text can be placed anywhere on the graphics screen. Up to eight colors plus black are allowed, and drawings can be animated by paging through up to six available screens.

Pluses: Allows drawings to be made using the joystick or keyboard. Pictures can be stored or retrieved on tape for use with other software or transferred to a Line Printer VII, Line printer VIII, or NEC 8023 printer. Colors are simulated by dot-pattern densities.

Minuses: The program is not compatible with the RS disk system.

Documentation: A six-page manual is included that describes the operation of the program.

Skill level requires: No special skills required.

Reviewer: John Steiner

Product Name:	**Earl's Word Power: Horrible Homonyms**
Equip. req'd:	Apple II or Apple II+
Price:	$29.95
Manufacturer:	George Earl 1302 South General McMullen San Antonio, TX 78237
Author:	Karen Knudson

Description: This is an educational program that enables the student to practice the most abused homonyms (their/there/they're, its/it's, too/to/two, etc.. Each homonym is defined and used in an example. Then a sentence missing a word is shown, with the homonyms listed below. Paraphrased Shakespearian plays are used for review tests.

Pluses: The program provides instant feedback with the score shown after each problem. When a mistake is made, the program reviews the material and then presents the problem again. The program is completely error-trapped (ignores spurious key entry), and the screen shows large, easy-to-read type. I would recommend this program for school and home.

Minuses: The 13 sets of homonyms are fixed; so a teacher cannot insert her own words. The program does not automatically send low scorers back for another review. Also, there is no provision for saving student scores on disk.

Documentation: One sheet on loading the disk; all necessary instructions are built into the program.

(continued)

Skill level required: This program is good for anyone with fourth grade-to adult-level English. No programming required.

Reviewer: Mary Gasiorowski

Product Name:	**HEXDOS 4.0**
Equip. req'd:	OSI C1P or Superboard with BASIC-in-ROM and disk drive
Price:	$49.50 includes diskette and manual
Manufacturer:	The 6502 Program Exchange 2920 West Moana Reno, NV 89509
Author:	Steven P. Hendrix

Description: *HEXDOS* is a disk-operating system for the OSI C1P system that requires only one disk track and 2K of memory for the DOS. It uses BASIC-in-ROM routines where possible to keep the DOS small.

Pluses: A real DOS is possible with as little as 8K of RAM. The user has 38 disk tracks available for storage and approximately 12K more memory than OS65D. The USR routine has been substantially expanded to do many things in addition to calling user-written routines. The "SAVE" command will create a disk file if none exists. File names may be of any length and are allowed to contain embedded blanks. Assembly-language files are uniquely identified and stored with load and execution addresses. Data files may be written and read from BASIC — up to 22 files may be open simultaneously. Opening the file automatically creates the necessary buffer space. The rubout key is now a non-destructive back space. *HEXDOS* will operate with CEGMON as well as the standard OSI monitor chip. Several hardware enhancements are supported (real time clock, tone generator, etc.). The program appears to be well supported by its author, with updates offered from time to time. Consultation is provided willingly in case of user problems.

Minuses: Lack of compatibility with OS65D. Disks created by *HEXDOS* and OS65D are mutually unreadable. I have an assembly-language routine to read OS65D sectors under HEXDOS, and a BASIC program for copying OS65D files to HEXDOS disks, that I will publish in the next issue of the HEXDOS Newsletter for current users, and include with future purchases of HEXDOS. The changes in the USR mentioned above require that any existing program using USR must be revised to run with *HEXDOS*. *HEXDOS* retains most of the features of BASIC-in-ROM, including 7-digit floating-point precision (OS65D now has 9½-digit f.p., - but 7-digit f.p. is faster and uses less memory if the precision is adequate for your work). The notorious garbage collection bug has not been fixed. (In fairness I must point out that a corrected BASIC ROM3 chip is available elsewhere at modest cost.) The smallest unit on disk is a track (i.e., one sector of 2K bytes per track).

Documentation: Newly revised 40-page manual, including demonstration programs and appendices. Terse, but apparently complete and error-free.

Skill level required: Ability to program in BASIC. No assembly-language experience is needed.

Reviewer: Rolf B. Johannesen

MICRO

6809 Bibliography

136. Color Computer News, Issue 14 (November, 1982)

Sias, Bill, "REMarks," pg. 3.
Remarks on the 6809-based color computer of Tandy *versus* other Tandy micros.

Kelment, George E., "HEX/DEC Conversion," pg. 10.
A listing for the TRS-80 Color Computer.

Benenson, Alexander, "Screen Print Program for the Color Computer," pg. 13.
A listing and notes on a utility for the Color Computer.

Perry, Thomas, "CW Send/Receive Program," pg. 14-21.
An amateur radio utility for the Color Computer.

Degler, Roger L., "Flex Corner," pg. 24-27.
A discussion of operating systems for the 6809-based color computer.

Steiner, John, "Morse Code Instruction: Part 2," pg. 30-32.
Programmed CW instruction for the TRS-80 Color Computer.

Hogg, Frank, "64K Korner," pg. 34-36.
Telewriter and FLEX and use of an external terminal.

Norris, Danny, "Chromaledger," pg. 38-45.
Chromaledger is an easy-to-use expense account program written for the 32K Extended BASIC Color Computer.

Pakerski, Andrew, "Tumble," pg. 46-48.
A game program for the Color Computer.

McGarry, Donald L., "Slither," pg. 49-52.
A programming game for the 6809-based Color Computer.

Aker, Jack L., "Care and Feeding of RS Disk Drives," pg. 54-56.
Discussion and diagnostic test program for Radio Shack Color Computer disk drives.

Kelty, John R., "A 'Cheap Talker' for the Radio Shack Color Computer," pg. 58-63.
Hardware and software for a speech program for the Color Computer.

Trepal, George, "POKE and String Graphics for the 4K Color Computer," pg. 64-67.
POKE and string graphics are much faster than SET and RESET graphics.

Sullivan, Steve, "Slope and Linear Graphing," pg. 68-70.
A program for graphing various algebraic equations.

Giovanoni, Richard, "Estimating On My Color Computer," pg. 73-77.
A program for proposal estimates.

Anon., "Learning ASCII Codes," pg. 82-83.
A learning program intended to burn the ASCII codes into your subconscious.

Knight, Glen B., "How I learned to Soundex Code and Love My Color Computer," pg. 89-91.
Family names are often spelled various ways. A Soundex code assists in retrieving these names in a computer file.

137. System-68, No. 2 (August, 1982)

Pass, Edgar M., "A Comparison of FLEX and UNIFLEX," pg. 7-19.
A detailed discussion and comparison of two operating systems for 6800 and 6809 systems.

Hughes, James, "Dynamic Address Translator," pg. 21-26.
An explanation of how the 6809 Dynamic Address Translation and Extended Memory Addressing software (SWTPC) work in order to be able to program them for correct operation.

138. System-68, No. 4 (November, 1982)

Pentecost, Joe, "Beginning Assembly Language Programming," pg. 8-28.
A tutorial to machine-language for the 6800- and 6809-based systems.

139. '68' Micro Journal 4 Issue 12 (December, 1982)

Reitzel, Norman L., "6809 Problems," pg. 10-11.
Discussion of some bugs in the 6809 with low mask numbers, and fixes available.

Anderson, Ronald W., "FLEX User Notes," pg. 11-14.
Discussion of the use of 6800 and 6809 programs.

Nay, Robert L., "Color User Notes," pg. 14-18.
Discussion of needs filled by the TRS-80 Color Computer; as an advanced system, as an excellent educational tool, and as a very good tool for control systems.

Kelty, John R., "Cheap Talker," pg. 18-19.
A speech program for the 6809-based Color Computer.

Hunt, Thomas H., "Hi-Res Color Graphics," pg. 19-22.
Arcade graphics by adding a special TMS-9918A and some other chips to your 6809-based system.

Lyon, Stewart D., "Adapting the Microworks SDS80C to FLEX09 DOS," pg. 22-24.
Detailed instructions for system modification, including an assembly-language listing.

140. The Rainbow 1, No. 6 (December, 1982)

Nolan, Sara, "Micro-Maestro," pg. 10-28.
Using the 6809-based Color Computer's PLAY statement. How to program music on the micro.

Schrag, Roger, "Patch EDTASMplus to Disk to End Those Cassette Blues," pg. 29-32.
How to modify the EDTASMplus package to support disk drives on the Color Computer.

Stumpf, Peter, "You'll Log This Program for Holiday Merriment" pg. 34-36.
Graphics for a fireplace complete with glowing flame and stockings hung with care.

Inman, Don, "High-Resolution Graphics Techniques Are Explained," pg. 41-48.
A tutorial with several demo listings for the 6809- based Color Computer.

Nolan, Bill, "Pressed for Time? Paint a Dungeon!" pg. 50-52.
How to speed up the actual play of a fantasy game by properly using the 6809 graphics ability.

Hands, Lester, "Memory Exam: Where Does it Start?" pg. 54-55.
A machine-language utility routine that allows you to rapidly examine memory and determine the address of items of interest.

Blyn, Steve, "Sustain Children's Interest By Expanding Relevance," pg. 56-60.
Motivating the young computer user to go beyond games and learning programs. A sample listing is included.

Kolar, Joseph, "On Printing Alphanumerics in Eight Directions," pg. 62-64.
A program to allow printing in any of the eight motion directions as used in the Color Computer's DRAW command.

Benenson, Alexander, "Screen Print Program for the Color Computer," pg. 13.
A listing and notes on a utility for the Color Computer.

Konecky, Larry, "This Is Just Like Music to Your Eyes," pg. 66-72.
In this Color Computer program music can be written and presented on a music staff on a black background.

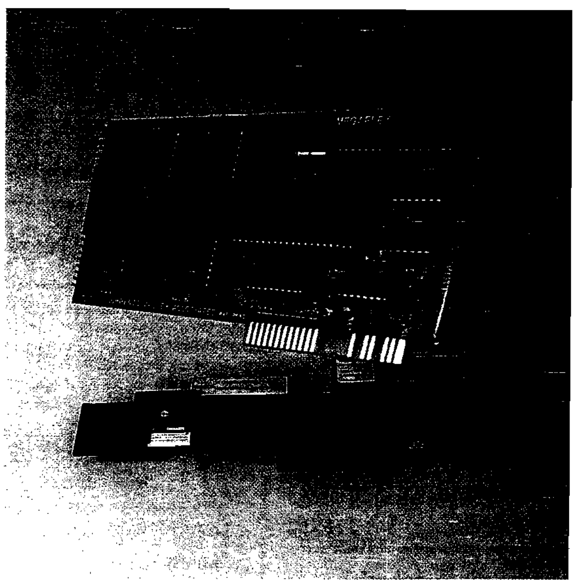
Bennett, J.E. and Laidlaw, "From Out of the Blue Comes This Dexterity Test," pg. 76-80.
A parachute drop game for the Color Computer.

Roslund, Charles J., "Format Your LLISTings with FLIST," pg. 86.
A formatting utility machine-language routine for the 6809- based Color Computer.

Wells, Geoff, "Go Adventuring with GAPAD," pg. 98-104.
A game-writing utility for the TRS-80 Color Computer.

MEGAFLEX® ABILITY

You Pick The Disk System, MegaFlex Controls It!



WITH SOFTDRIVERS FOR A FLEXIBLE FUTURE!

MEGAFLEX—a universal floppy disk controller and modern alternative to the Apple drive system offering increased storage, improved reliability and . . . FLEXIBILITY.

Enjoy megabytes of online storage with your choice of micro, mini, or maxi drives—or even 6Mb with the Amlyn cartridge pack! Ideal for high-capacity storage now, winchester-disk backup later.

The MEGAFLEX secret is to autoboot softdrivers that match the needs of your drive system. All hardware functions are software-controlled. MEGAFLEX can match new drive capabilities without hardware changes. Drive-dependent ROMs have been eliminated.

APPLE *///*? OF COURSE!!

MEGAFLEX is compatible with BASIC, CP/M, Pascal, VISICALC, SOS and DOS-emulation on the Apple III, Apple II, Franklin Ace and Basis. All language features and operating system commands (LOAD, BRUN, etc.) are standard. If you can operate Apple drives you can operate MEGAFLEX! Your Apple software will run without modification too.

BRIDGE THE APPLE FORMAT BARRIER!

The MEGAFLEX diskette does what Apple's cannot—read and write diskettes from other computers! Software-controlled industry-standard IBM 3740 or System 34 type formats allow the MEGAFLEX library of reformatting software to read and write Altos, Radio Shack, Osborne, and IBM PC diskettes. (Call for the latest software details.)

MORE STORAGE, MORE UNIVERSAL FEATURES, LOWEST COST

MEGAFLEX with 8" maxi or high density 5.25" minis gives you 1.2 Megabyte of formatted data per diskette for 8 times the file and data size!

MEGAFLEX offers flexible software choices:

- data rate (250/500 Kbits per second),
- single and double density recording, and
- single/double sided drive operation (max 4 drives).

MEGAFLEX has the lowest chip count of any controller today! This means less power, a cooler Apple and better reliability.

Lowest price, highest performance, that's MEGAFLEX!

MEGABYTER™
A Division of SVA

11722 SORRENTO VALLEY ROAD
SAN DIEGO, CA
(619) 452-0101
TWX 910-335-2047 APPLE TWO SDG

TRADEMARKS CP/M—Digital Research

Circle No. 102

Advertiser's Index

MICRO INK is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

National Advertising Representatives

Middle Atlantic and Southeastern States:

Dick Busch Inc.
Richard V. Busch
6 Douglass Dr., R.D. #4,
Princeton, NJ 08540 (201) 329-2424

Dick Busch, Inc.
Eleanor M. Angone
74 Brookline,
E. Atlantic Beach, NY 11561 (516) 432-1955

serving: New York, Pennsylvania, New Jersey, Delaware, Maryland, West Virginia, Virginia, D.C., North Carolina, South Carolina, Louisianna, Tennessee, Mississippi, Alabama, Georgia, and Florida

West Coast:

The R.W. Walker Co., Inc.
Gordon Carnie
2716 Ocean Park Boulevard, Suite 1010,
Santa Monica, California 90405 (213) 450-9001

serving: Washington, Oregon, Idaho, Montana, Wyoming, Colorado, New Mexico, Arizona, Utah, Nevada, California, Alaska, and Hawaii (also British Columbia and Alberta, Canada)

Mid-West Territory:

Thomas Knorr & Associates
Thomas H. Knorr, Jr.
33 N. Michigan Avenue, Suite 403
Chicago, Illinois 60601 (312) 726-2633

serving: Ohio, Oklahoma, Arkansas, Texas, North Dakota, South Dakota, Nebraska, Kansas, Missouri, Indiana, Illinois, Iowa, Michigan, Wisconsin, and Minnesota.

VistaDaVinci
GREAT INNOVATORS

15th century renaissance man, Leonardo Da Vinci, contributed an inexhaustible collection of inventions and ideas to solve the world's problems. These ideas were centuries ahead of their time. The studies on this page deal with experiments in manned flight.

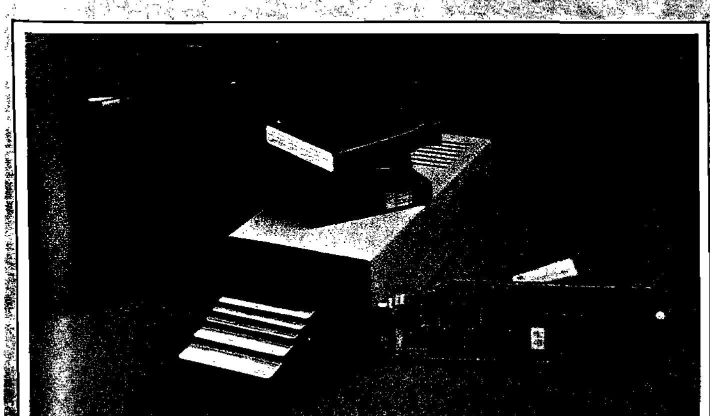
Like Leonardo, Vista Computer is answering many of today's complex computer storage problems. The Vista V1200 is a great solution to Apple II™ storage. Mass storage for your Apple II™ Computer has always been a problem. On one hand, there were the exotic, expensive hard disks with no cost efficient means of backup. On the other hand, the Apple floppy drive lacked the speed and storage demanded by today's professionals.

Vista's V1200 offers both at an incredibly attractive price. The removable VistaPak cartridges offer 6 Megebytes of removable storage each and can be backed up like a floppy.

Now hard disk storage and speed can be yours with the added capability of interchangeable media. The V1200 eliminates the worries of head crashes, drive alignments, lost data or backup with a new application of field-proven floppy technology.

The VistaPak cartridges hold 6MB of formatted data each. The removable cartridge allows you to keep duplicates of your valuable data as well as to keep separate paks for your accounting, word processing, spread sheet and other applications. No other storage device offers more in flexibility and capability.

• Microprocessor controlled drive • DMA Controller • Removable Data Cartridges • CP/M, DOS & Pascal compatible • Quickcharge™ DOS enhancement included • Includes 1 VistaPak cartridge • Vista 120 Day Warranty



Vista V1200

™Apple II is a registered trademark of Apple Computer Co.

Contact Your Local Vista Dealer or Call our Vista Hotline

Vista COMPUTER COMPANY, INC.

[illegible]

(714) 953-[illegible] [illegible] 8017

DISTRIBUTORS / REPRESENTATIVES

[illegible]

South Central [illegible]

(214) 385-8885