NO. 38 $2.50 JULY 1981

MICRO™

THE 6502/6809 JOURNAL

Two 16-page bonus sections for OSI and Apple users

A Bit Pad for Your Micro

AIM Memory Maps

Expressions Revealed

6809 Super Features

MICRO™

THE 6502/6809 JOURNAL

ARTICLES

APPLE BONUS

OSI BONUS

STAFF

Editor/Publisher
ROBERT M. TRIPP

Associate Publisher
RICHARD RETTIG

Associate Editor
MARY ANN CURTIS

Special Projects Editor
MARJORIE MORSE

Art Director
GARY W. FISH

Production Assistant
LINDA GOULD

Typesetting
EMMALYN H. BENTLEY

Advertising Manager
CATHI BLAND

Circulation Manager
CAROL A. STARK

Dealer Orders
LINDA HENSDILL

MICRO Specialists
APPLE: FORD CAVALLARI
PET: LOREN WRIGHT
OSI: PAUL GEFFEN

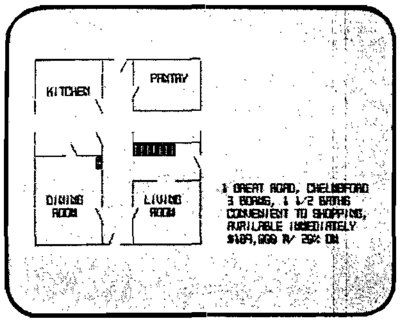
Comptroller
DONNA M. TRIPP

Bookkeeper
KAY COLLINS

Sales Representative
KEVIN B. RUSHALKO

DEPARTMENTS

About the Cover



Real Estate

Pictured on this month's cover is the historic Fiske House in downtown Chelmsford, just down the block from MICRO. The display shows one of the ways that a microcomputer might be used in the real estate business: to present listings to potential buyers. Instead of requiring the buyer to look at dozens or even hundreds of houses, many of which are of absolutely no interest, the buyer could answer a short questionnaire detailing the type of house, location, price range, bedrooms, and other significant features desired. This material then could be used to match the houses on file and to present only those houses for consideration which had a reasonably high correlation. In addition to listing the basic facts normally found, the file could contain a floor plan, as in the cover example; a map showing the location of the house; a simulated "tour" of the house; and other pertinent information.

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and Avon, MA 02322
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
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

Subscription rates	*Per Year*
U.S.	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

MICRO Editorial

An Important 18 Cent Investment

A frustration in publishing MICRO arises from the fact that the information flow is essentially uni-directional. While a tremendous volume of material goes out, only a trickle of information comes back in. There is very little feedback from the MICRO readership to let us know how we are doing. The letters we get from individuals tend to focus only on one or two points that are of immediate importance to the author of the letter. There is no regular channel for us to obtain a broad-base understanding of who our readers are, what interests them, what they do with their computers, what they would like to do with their computers, and so forth. To help remedy this, we are taking a reader survey. You will find the Reader Survey Form inserted between pages 96 and 97 of this issue. The information received in this survey will have a major influence on the directions which MICRO takes in the near future. Therefore, those readers who do take the time to complete the questionnaire and spend the 18 cents to return it will have a great influence on the magazine.

More on the 6809

It was with some uncertainty that MICRO decided to cover the 6809. I thought that some readers might be upset that MICRO would have anything to do with any microprocessor other than the 6502. So far, all of the response has been positive. Several long-time subscribers have contacted me to say that they discovered the 6809 over the past year, are very happy with it, and are glad to see MICRO cover it. A number of people at the recent Applefest in Boston expressed interest in the 6809 and wondered how it might affect the Apple. A couple of 6809 experts have contacted me about providing articles for MICRO, so there should be a significant increase in the quality and quantity of material in future issues.

I freely admit that I am a novice on the 6809. To date I have written only one minor program, hand assembled, for the 6809. Therefore, the material that I am presenting in my series is only to be taken as a basic introduction to the device, as seen through the eyes of a 6502 devotee. The material from the 6809 experts in future issues will cover a wider variety of topics in greater depth. If you are knowledgeable of the 6809, please consider sharing your knowledge with us. I would be happy to discuss possible articles with you by letter or phone.

The more I investigate the 6809, the more I like it. There are little things such as the two-byte addressing which is the natural high-byte/low-byte form (12 34) instead of the reversed form used by the 6502 (34 12). There are more significant improvements such as the 16-bit operations. And, there are major effects, such as greatly increased transportability of code. Since the 6809 does not make special use of page zero or page one, it eliminates one of the major areas of contention that one encounters when trying to make 6502 code general. When I wrote a program to support a video board on the AIM, SYM and KIM, I kept running into problems of page zero and page one usage. Since each machine had allocated different sections of these limited memory resources, it became impossible to find any locations which were universally free. This type of memory contention would simply not occur on the 6809.

Of even greater significance to making code transportable is the 6809's inherent position-independent code capability. There are several companies which offer complete disk operating systems for the 6809 which can be fairly easily adapted to any 6809-based system. Once the particular 6809-based operating system is installed, a large number of packages are commercially available. These include BASIC, Pascal, FORTH and other languages; word processors, assemblers, editors and other "tools;" and a variety of business-oriented applications. This means that many new 6809-based computers can be designed and built that can take advantage of common software. This should encourage programmers to write truly universal software packages for the 6809 and perhaps eliminate the "Tower of Babel" that has evolved within the 6502 world, where almost every program is specific to a single microcomputer.

Robert M. Tripp

MICRO

Letterbox

Dear Editor:

I have both good news and bad news for MICRO readers. The good news is that the 6516 will shortly be available for purchase by the public. The bad news is that it is a 16K CMOS RAM made by Harris.

Rats!

Hal W. Hardenbergh, President
Digital Acoustics, Inc.
1415 E. McFadden, Suite F
Santa Ana, California 92705

Dear Editor:

This is a reply to the anonymous letter in the May issue of MICRO (36:16). I am one of those "skinflint," "bare-board" KIM-1 users and I think this is a typical reply from all of us "unintelligent," "not-so-serious," "impoverished single-board" users who read MICRO.

Since purchasing my KIM-1 a few years back, for a paltry two hundred and fifty dollars, I have added the following:

Three Memory Plus boards with PROM and RAM
One Mother Plus board
One case for the KIM-1 (no longer a "bare-board")
Three power supplies
One Micro-Ade package (assembler-disassembler-editor)
One Microsoft 9K BASIC package
One Tiny BASIC package
One printer
Two cassette drives
One ASCII keyboard
One video terminal board
One video monitor
Twelve EPROM chips at $50 each
One extended monitor package
One information retrieval package
One logic probe
One stringy floppy or regular floppy (tentative)
One 4800 baud tape interface board
One tape management system package
One subscription to MICRO magazine
One subscription to COMPUTE magazine
One EPROM eraser

I think the Editor of this magazine will recognize a lot of "familiar" products in this list.

My point is this. Before you Johnnie "Appleseeds" and the like shoot off your mouths about us "impoverished, bare-board users," it would do well for you to investigate just who supports the small-user industry.

The products on my list came from *various* manufacturers, not just *one*, who all advertise in magazines such as MICRO.

If you want the "Black Box" concept (it doesn't take a lot of intelligence or sophistication to operate a "black box") that is your business, but don't force your snobbish attitudes on everyone else....

I work with black boxes at work all day long (Data Generals, Harris Slash/7, MACSY M-2, etc.), but after work I want to delve into something a little more challenging and rewarding. In other words, I like to do it "my way."

A "skinflint KIM-1 user"
from St. Louis, Missouri

Dear Editor:

Enclosed is an Apple tip that I think might be of interest to the readers of your magazine. In order to make some types of programs easier to find in your catalog, the type name can be changed to another character. For example, the 'B' in binary programs may be changed to a 'flashing B'. The 'T', 'I' and 'A' may also be changed to any ASCII character. Refer to the Apple manual, page 15, for a table of ASCII characters. Here are the POKEs.

```
POKE 45191,?
(Change T in text files)
POKE 45992,?
(Change I in integer files)
POKE 45993,?
(Change A in Applesoft files)
POKE 45994,?
(Change B in binary files)
```

Example:

```
POKE45994,66 Changes 'B' in
binary file to 'flashing B'
```

If you initialize any disk after making these POKEs they will have the changes written in their DOS permanently. For a 32K system subtract 16384 from the above POKEs.

Dean Kay
P.O.Box 3984
Irving, Texas 75061

Dear Editor:

Allow me to relate my experiences with a genuine software thief and his immediate victims. An ad appeared locally offering Apple PIE or Easy-writer for $50 (vs. $130 and $100 list price). I called the number given and asked the man if he had VisiCalc, too. He did indeed... for $40 (vs. $150 list)! He went blatently on to tell me that it was a copy, that I could make my own backup disks and that the documentation was photocopied. "Do you realize," I asked him, "that you're a thief?" A pause... "Yeah," he said. I hung up in his ear.

If you look out your window and see someone picking the lock of your neighbor's car, would you turn away? If you feel a pickpocket's hand in your own pocket, do you just stand there? A software thief is no better than a car thief or a pickpocket. If we, the users and producers of software, prove unable to police ourselves there will surely be someone happy to do it for us. Uncle Sam will have his heavy finger on your keyboard and his beady eye on your disks. We'll all be saddled with yet more Big Brother government, empowered to watch our every software purchase and sale. And who will pay for this watchdog bureaucracy? You will. I will. Every person and company in the United States will pay for it with their taxes. Is that what this thief wanted? Or was he just too stupid to think?

So I phoned Personal Software, Inc., (about VisiCalc) and Programma International (about Apple PIE). (I would have called Easy-writer's manufacturer but I had no company name or phone.) I talked to the highest-ranking managers there and told them of the thief. Both men were shocked. Perhaps these calls

(Continued on page 18)

AIM Memory Map

This article describes how a ROM-based assembler works, with detailed instructions for getting at several useful, but undocumented features, including new .OPT functions for the AIM.

Greg Paris
11-2A English Village
Cranford, New Jersey 07016

The AIM 65 assembler was designed by Compas Microsystems (the makers of the AIM monitor) to be a subset of its larger, RAM-based A/65 assembler. In fitting the AIM assembler into a 4K ROM, several features of the A/65 assembler had to be dropped. What remains, however, is an extremely useful program to be resident in one's AIM, even if it doesn't list a sorted symbol table or count lines of program listing.

I wanted to see if I could extend the AIM assembler's command set through a conveniently-placed zero-page RAM hook or vector. I found out quickly that I could not. But in the process of line-by-line decoding, I found many other things of interest — some useful subroutines which can be called from outside the assembler, and several hidden shortcuts and undocumented functions. This article will provide a memory map of the AIM 65 assembler ROM, describe its operation and use of RAM, and detail these undocumented features.

The Assembler Disassembled

Table 1 shows how the assembler is organized into a 4K block of memory which starts at $D000. Most of the look-up tables are found near the upper end of this block, which allows the majority of the program from $D000 to $DD4A to be disassembled continuously by use of the AIM monitor command "K". If you do it for yourself, it's best to disassemble only 1 to 2 pages of memory at a time, to prevent your power supply from overheating any more than it usually does.

***Table 1:* Assembler ROM Memory Map**

D000 - D0DF	initialize RAM and setup for PASS 1
D0E0 - D0E8	loop to process lines of source code; stack reset each time
D0E9 - D66E	SBR - PROCESS a line... includes:
D104	..get a line from AID; echo to display
D128	..separate labels from mnemonics and operands
D1DB	..reassign program counter or PC (* =)
D1E8	..process an equate (=)
D259	..directive (.XXX) decoding; then jump-indirect to do it
D299	..encode data as per .BYT, .WOR, .DBY instructions
D346	..check and assign .BYT data in ASCII literal format
D396	..decode .OPT XXX; then jump-indirect to do it
D3B3	..set up directive flag variable ($37)
D3CC	..do .OPT SYM, NOS, NOC, CNT, and COU: i.e., nothing!
D3D4	..perform .SKI
D3DE	..perform .END; setup for PASS 2
D414	..toggle tape recorders while waiting for PASS 2
D43E	..set up FNAME for tape file for PASS 2
D454	..encode mnemonic/symbolic address into opcode/operand
D66F - D68F	SBR - do list of line and preset ERROR statement; then NEW line
D690	execute .FIL if AID = T or U
D69D	perform .PAG
D6CA -	SBR - get beginning-of-line pointer, then
D6CE -	SBR - increment line pointer, then
D6D0 - D6E7	SBR - get first non-space character to begin string
D6E8 - D71F	SBR - get last character in a string; ignore between quotes
D720 - D74A	SBR - look for), comma, space or end-of-line (EOL)
D74B - D75B	SBR - output the buffer to LIST-AOD until quote or EOL
D75C - D767	SBR - carry set if alphabetic character
D768 - D773	SBR - carry set if numeric character
D774 -	SBR - set A = 3, then
D776 -	SBR - store A as number of characters, then
D778 - D796	SBR - transfer characters from text buffer to SEARCH buffer
D797 - D8AC	SBR - EVALUATE an expression..., includes:
D7B9	..select low byte of symbol (<)
D7C1	..select high byte of symbol (>)
D7D4	..decimal number string
D7DA	..hex number string ($)
D7E0	..octal number string (@)
D7E6	..binary number string (%)
(D7D4 - D7E6)	set up to convert to a hex number (.DBY format)
D7E8	..get symbol value with SEARCH
D81D	..evaluate current pointer or PC (*)
D858	..perform 2-byte addition (+)
D886	..perform 2-byte subtraction (−)
D8AF - D8C2	SBR - test flag from EVALUATE for arithmetic error and overflow

(Continued)

There are several directives and "list" options which are supported by the assembler. The recognition process requires that a list of these commands (in ASCII) be present in ROM to be scanned as necessary. This list, and the action address for each command, are shown in table 2. I noticed that there were more options listed in ROM than I had ever seen described. As I will detail later, there is a new pair of options which *are* supported — .OPT MEM and .OPT NOM — and several which are recognized (i.e., not rejected outright with "**ERROR 14") but simply ignored.

A memory map of any program is only of limited usefulness if its constants and variables are not well-documented. Table 3 shows how the assembler utilizes zero page RAM, and the functions of most of these addresses, or their contents. In addition to this zero page use, a section of page one, just below the stack area, is reserved for the temporary storage of compiled opcodes and data. Several addresses vie for the most-used-zero-page-address award, but the winners are $46+ (the text input buffer starting address), $35 (the length of the current line in said buffer), and $29 (the pointer to the active character in this buffer, a single byte usually stored here from the X register.)

How It Works

The following description will be most informative if the disassembled object code is available, if for no other reason than to see how some of the tricks are accomplished with minimal coding. But it's not absolutely necessary.

All the real work of assembly is directed from the subroutine at $D0E9 - $D66E, which I've labeled PROCESS. The section immediately preceding this (from $D0E0 - $D0E8) is a small loop which calls PROCESS each time a new line is to be processed. This loop does only two things: resets the stack pointer, and calls PROCESS. All other subroutines are called from PROCESS.

If it becomes necessary to leave PROCESS because of some fatal processing error, even if the stack pointer is randomly set, there is no problem because exit always occurs after the stack pointer is partially reset. This allows an RTS instruction to return control to the small loop. (See $D686 -$D688 for how this is done.)

The assembler itself has very few functions: get some text; try to assemble it; check for errors; and output the results. The actual processing is almost as simple as the statement.

Table 1 *(Continued)*

D8C3 - D8DA	SBR - get current character with X register as pointer; also check for end-of-symbol
D8DB - D8EC	SBR - get opcode addend from table
D8ED - D94E	SBR - base conversion
D94F - D955	SBR - test for carry from previously performed add/subtract
D956 - D95D	TABLE - constants for base conversion
D95E - D9A1	SBR - SEARCH symbol table for entry
D9A2 - D9D3	SBR - STORE symbol and value in table
D9D4 - D9E9	SBR - if string = mnemonic, get opcode data
D9EA - DA0B	SBR - find mnemonic
DA0C -	SBR - set flag for no-error/list-line-only, then
DA0F - DA5D	SBR - decode error number, select LIST or not
DA5E - DBC6	SBR - LIST a line to LIST-AOD and output OBJ code to OBJ-AOD, followed by **ERROR XX, if needed..., includes:
DA7E	..determine if PC needs to be output
DA90	..output PC at beginning of line, then
DAA0	..output label if one is present
DAC3	..recalculate when next PC announcement is due
DAD0	..output opcode/operand or data
DB19	..output rest of line
DB62	..format quotated strings
DBB2	..finish output line; return for more data if .OPT GEN selected
DBC7 - DBEC	SBR - output an error message and number; increment error count
DBED -	SBR - set A = 1, then
DBEF -	SBR - add A to PC, then
DBF8 -	SBR - zero A, then
DBFA - DC05	SBR - add PC to A storing result as memory deposit pointer
DC06	SBR - output single byte to OBJ-AOD
DC09 - DC28	SBR - output byte as 2 ASCII hex numbers to OBJ-AOD
DC29 -	SBR - add opcode to A, then
DC2E - DC4D	SBR - output A to memory, or to OBJ-OUT intermediate buffer
DC4E -	SBR - move from intermediate buffer to OBJ-OUT buffer, then
DCA9 - DCB7	SBR - clear OBJ-OUT intermediate buffer
DCB8 -	SBR - zero and start OBJ-checksum calculation, then
DCC8 - DCD1	SBR - add A to OBJ-checksum
DCD2 -	SBR - format and output an OBJ-code record, then
DD02 - DD0C	SBR - CRLF to OBJ-AOD
DD0D - DD4A	SBR - format and do last OBJ-record; close tape file
DD4B - DD74	TABLE - assembler directive action addresses (.WOR format)
DD75 - DDB3	TABLE - assembler directives and .OPT list, in ASCII
DDB4 - DE5B	TABLE - mnemonic list, in ASCII, in alphabetic order
DE5C - DE65	TABLE - allowed opcode addends
DE66 - DE74	TABLE - look-up index to reference table $DE75
DE75 - DEDD	TABLE - look-up legal operand format
DEDE - DF15	TABLE - opcode classification list
DF16 - DF4D	TABLE - basal opcodes; in same order as mnemonics
DF4E - DFA2	TABLE - messages, in ASCII; each one ends with a semicolon
DFA3 - DFA7	TABLE - reserved labels, in ASCII: "AXYSP"
DFA8 -	SBR - set up display and monitor with FNAME of .FIL, then
DFCC - DFDC	SBR - go get file if AID = T or U
DFDD - DFE8	SBR - print a message; input in X = offset of beginning of message from $DF4E
DFE9 -	SBR - output a blank space, then
DFEC - DFF5	SBR - output a CRLF to AOD
DFF6 - DFF9	??TABLE?? - four unidentified bytes...
DFFA - DFFE	SBR - output space to AOD
DFFF	"N" in ASCII: the monitor command to jump to the Assembler

***Table 2:* Assembler Directive and Option Mnemonics**

Location of First Byte	Mnemonic	Action Address (hex)
DD75	BYT	D299
DD78	WOR	D2A1
DD7B	DBY	D29D
DD7E	SKI	D3D4
DD81	PAG	D69D
DD84	END	D3DE
DD87	OPT	D39D
DD8A	FIL	D690
DD8D	GEN	D3B3
DD90	NOG	D3B7
DD93	SYM	D3CC (unsupported)
DD96	NOS	D3CC (unsupported)
DD99	NOC	D3CC (unsupported)
DD9C	CNT	D3CC (unsupported)
DD9F	COU	D3CC (unsupported)
DDA2	ERR	D3BB
DDA5	NOE	D3BF
DDA8	MEM	D3C8
DDAB	NOM	D3C4
DDAE	LIS	D3BF
DDB1	NOL	D3BB

Input text is obtained from the AID as specified by the monitor variable INFLG (which also allows input directly from memory) in a loop from $D104 - $D127. Output, on the other hand, can be two-fold: actual object code (the real reason for using this program, after all) and a formatted assembly listing. These must go to two different devices, and a significant portion of the assembler is devoted to the proper formatting of the listing ($DA5E - $DBEC) and to the production of a formatted standard object code ($DBED - $DD4A). If the object code is to go directly to memory, no formatting into a record is performed, and the code is merely deposited (at step $DC3C) as per the pointer in $09/0A.

The assembly itself is done as follows. The input line is first parsed into labels, mnemonics or assembly directives. Any string that does not meet these criteria is rejected with error numbers 3, 8, 9, 10, or 20. Directives are processed by the section which starts at $D259; the jump-indirect to the specific address is taken only after the directive in the text is compared with those commands supported (see table 2) and the proper action address is obtained from the table at $DD4B. Any errors in this process are called "undefined assembler directives." When a directive has been performed and listed (if desired), exit to the small loop at $D0E0 occurs.

Those strings which are used as symbolic constants or address labels are differentiated from mnemonics by length, or by a mnemonic scan called from $D167. Labels may be associated with equates, or with the current program counter address (PC). On the first pass, if the string is legal and not a mnemonic, it is assigned a value and placed in the symbol table with this value by the subroutine called from $D1CF. If the string is found to be a mnemonic, a branch occurs to that section of the assembler which performs the actual opcode assembly calculations.

The opcode compiler starts at $D454 and is the heart of the assembler. First the mnemonic is checked against a list in ROM, which starts at $DDB4. Like the directive list, this list is in ASCII, and is conveniently arranged alphabetically. Then, two new bytes of information are obtained using the position of the mnemonic in the list as an index. The table which starts at $DF16 yields the "basal opcode." This is a single byte which represents the lowest numeric value of the opcodes allowed for a given instruction, to which a constant determined by the assembler may be added. And the table at $DEDE yields the opcode classification type. How do these two bytes determine the actual opcode?

If you look at the allowed instruction set for the 6502, you will see that not only does it contain holes (not all instructions use all addressing modes) but there is some pattern to these holes. Various mnemonics can be grouped together by considering which modes are allowed for each. Table 4 shows how this classification scheme is implemented. What the assembler does in the opcode compiling section is to sort out the requested mode, and give errors if this disagrees with those allowable modes obtained from table $DEDE. Then it evaluates the expression which is the operand (if any) and does the following calculation (more or less):

basal opcode + (addend from table $DE5C × factor Q) = opcode for the desired addressing mode.

"Factor Q" is determined when the syntax of the operand is checked. It takes into account such things as whether the address is page zero, or whether the mode is implied, indirect, indexed, etc. If your source code can run this gantlet, it is assembled.

One concept simplifies the control of much of the operation of the assembler — flag variables. Several page zero locations store information which is used repeatedly to direct operations: locations $21 - $23, and $36 - $38. Of central importance is the directive flag, $37. Three of its bits are used to store the status of various selected options and allow this status to be tested frequently during assembly. Table 5 details how the bits of this variable are understood by the assembler. This variable will also be of importance later in the discussion of the undocumented .OPT MEM/NOM functions.

There are few differences between PASS 1 and PASS 2. During the first pass, any output is swallowed by the program instead of being directed to the printer or OBJ-OUT device. The symbol table is compiled during the first pass, and is used extensively in the second pass to evaluate expressions. The distinction between each pass is signaled by the PASS 1/2 flag — $23.

Undocumented Features

This is probably the section you turned to first! Here I'll describe those assembler functions which haven't been detailed in the AIM manual, including a few shorthand notations, a built-in routine which allows the user to toggle tape recorders on and off while waiting for PASS 2, and several undocumented .OPT functions, especially two which are supported but not described in the manual.

1. I found three shorthand techniques that are allowed by the assembler. First, the indexed indirect addressing mode can be written either as LDA (VAR,X) or LDA (VAR,X with no closing parenthesis. Second, the indirect indexed addressing mode can be written either as LDA (VAR),Y or LDA (VAR)Y with no separating comma. Third, single-byte ASCII literal operands may be denoted in two ways: CMP #'X' or CMP #'X with no closing quotation mark. This last shorthand is not explicitly stated in the AIM manual, but it is used as an example on pg. 5-19 (rev 3/79). These shorthand methods save one shifted keystroke per operand. Note, however, that .BYT 'XXXXXXX' still requires a closing quotation mark.

2. If you have ever assembled from a source file on a tape cassette under remote control, you will have noticed one inconvenient operating detail: while the assembler waits to do PASS 2, the remote line shuts off your recorder! Before the tape can be rewound, you have to manually override this control, and, for example, disconnect the remote plug. But no more! The capability to toggle the tape remote control is already a part of the assembler. Here is how it works.

Table 3: Assembler RAM Usage

(00→03)	(not used)
04	number of bytes in data or opcode/operand at SBR $DA0F
(05)	(not used)
06/07	.WOR—temporary storage of program counter (PC)
08	error index at SBR $DA0F
09/0A	.WOR—pointer used to store OBJ code in memory
0B/0C	.DBY—number of entries in symbol table
0D/0E	.WOR—directive action address or SEARCH address
0F	basal opcode stored here
10	opcode classification type (see table 4); or $E if branch
11/12	.WOR—symbol counter for SEARCH
13/14	.DBY—value of symbol; or workspace for * assignment
15	+ or − sign for EVALUATE
16	same as 04, but maximum value allowed is $14
17/18	parameters for BASE conversion; loaded from table at $D956
19	number of bytes in completed .BYT ASCII literal string; or flag for formatting quotated material for LIST
1A/1B	.DBY—number of errors in PASS 2
1C	allowable operand coding key
1D	expression OK/NOK flag
(1C, 1D)	used in opcode processing
1E	error number (in decimal) for to print **ERROR XX
1F	output line counter for LIST formatting
20	flag: "this line contains a label"
21	flag: "* ="
22	flag: used to select .DBY, .WOR, .BYT notation
23	pass counter: PASS 1 = 0; PASS 2 = 1
24	pointer to next non-space character in buffer
25	pointer to last character of string in buffer
26	number of characters in string
27/28	.DBY—output of EVALUATE = value of expression
29	pointer to active character in buffer
2A→2F	string storage for comparison by SEARCH
30	number of bytes compiled at SBR $D66F et al.
31	stored error number at SBR $D683
32/33	.WOR—program counter or PC
34	display buffer pointer
35	number of characters in current line in buffer
36	flag: for > or < operations
37	flag: directive/option status (see table 5)
38	flag: arithmetic over- or under-flow from EVALUATE
39	number of bytes (.BYT = 1; .WOR and .DBY = 2)
3A/3B	.WOR—symbol table start
3C/3D	.WOR—last active symbol
3E/3F	.WOR—symbol table upper limit
40/41	.WOR—OBJ output record counter
42/43	.DBY—OBJ record checksum
44/45	.WOR—address at which PC is next due to be LISTed
46→81	input buffer; usually uses X as index/pointer
82/83	workspace... various uses
84	index/pointer for OBJ intermediate buffer
85/86	used in OBJ output processing: absolute address of where data would be deposited if not stored in intermediate buffer
87	OBJ-OUTFLG, if defined
88	LIST-OUTFLG stored here when OBJ is being output
89→A6	record assembly space for OBJ output... includes:
89	number of bytes in record
8A/8B	starting address of data
8C→A2	data
A3-A6	checksum
A7→AB	AID input FNAME stored here
0170-0183	intermediate storage buffer of compiled object code

Assume that PASS 2 has been displayed, and that the assembler is patiently waiting for you to press "space" to initiate the second pass. Instead of "space", press "1" or "2", depending on which line is connected to your recorder. *Voila*, your recorder is now running. Rewind to the start of the file, toggle "1" (or "2") again if you wish, start the recorder, and *then* press "space" on the keyboard. It's as easy as that.

3. Now to the undocumented options. You may have noticed in table 2 that several assembler mnemonics were unfamiliar. Indeed, MEM and NOM are supported, and I'll discuss them in the next paragraph. But the options SYM, NOS, NOC, CNT, and COU, while recognized, are not supported. Their action addresses direct processing to null place in the program so their inclu sion doesn't crash the assembly, bu merely is ignored. I assume that thes are fossils which remain from the com mand set of Compas Microsystem' larger A/65 assembler. With tha assumption, some of their functions ca be guessed at: SYM/NOS toggled th printing of a sorted symbol table NOC/CNT probably determine whether each line of the formatte assembly listing was sequentiall numbered; and COU probably set th number of lines per page. Note tha there is room in the directive fla variable for, at most, 5 more statu toggles than are used by the AIN Assembler.

4. .OPT MEM / .OPT NOM *doe* work, however. Its syntax is like that o other .OPT commands, and the optio determines the status of bit 3 in th directive flag. (See table 5.) This optio allows the user, for whatever reasons, t choose exactly when and where th object code will be directed durin assembly. As with other options, use o an .OPT command overrides thos parameters determined during th initialization dialog. But this mean that if .OPT NOM is to be use somewhere in the source text, the use *must* reply "Y" to "OBJ?" during th dialog, and then specify the OBJ-OUT device to insure that the OBJ-OUTFLC will be determined before it is needed Thereafter, .OPT MEM and .OPT NOM will allow object code to be directed t this device as desired during assembly o the source program.

I have even found a few usefu subroutines that can be called from out side the assembler. Some of these ar described in detail in table 6. I especiall like the subroutine which converts fron multiple base systems to hex notation Although it cannot be incorporate directly into a USR function and calle from a BASIC program because of zer page RAM conflicts, the concept can b used by anyone to provide a simple bas conversion function in BASIC.

Finally, a word of warning to an reader who may want to relocate th assembler. Disassembling this progran into a source file cannot be done blindly Various changes must be made manual ly. These are summarized in table 7. I these suggestions are followed, an planned reassembly should procee smoothly.

***Table 4:* Opcode classifications from table $D9DF**

Table Entry	Class of Opcodes
01	widest variety of operand type allowed (as for ADC, LDA, etc.)
02	STA
03	JMP, direct or indirect
04	JSR
05	accumulator mode allowed (as in LSR)
06	CPX,CPY
07	BIT
08	LDY
09	STX
0A	STY
0B	LDX
0C	DEC
14	single bytes (accumulator mode not allowed) (as in SEC or TAY)
15	all branches

Greg Paris has been doing postdoctoral research in neurobiology, and has turned his hobby into a job — as Senior Applications Specialist at Merck Pharmaceutical Co. He interfaces between the research scientists and the programming and design staff.

MICRO™

***Table 5:* Directive Flag Variable ($37)**

Bit Number	Used For	.OPT If Bit Is SET	.OPT If Bit Is CLR
7	generate complete data for .BYT command?	NOG (no)	GEN (yes)
6, 5	(not used)		
4	output a complete assembly listing or errors only?	ERR / NOL (errors only)	NOE / LIS (complete)
3	object code to memory	NOM (no)	MEM (yes)
2, 1, 0	(not used)		

***Table 6:* Useful Subroutines: I/O formats, RAM and register usage.**

SBR entry address	Function	Input	Output	Flags upon exit	Registers altered	RAM used, including that of called SBR's
D797	EVALUATE an expression	pointer to beginning of expr in 46,X	value in 27/28 (if done)	test $38 .and. Y=0, 1 or 2 0: not done 1: no symbol found 2: OK	AXY	13/14 15 16 17/18 27/28 32/33 35 36 38 82/83
D8ED	BASE conversion	pointer to beginning of string in 46,X	hex value in 13/14	SEC if OK CLC if not possible .also. test $38	AXY	13/14 16 17/18 35 82/83 38
D95E	SEARCH for symbol table entry	label in $2A+	value in 13/14, if found	SEC if OK CLC if not found	A Y	0B/0C 11/12 13/14 2A+ 3A/3B 3C/3D
D9A2	STORE symbol and value in table	value in A/MSB and Y/LSB symbol in $2A+	none	if no room, Assembler automatically restarts	A Y	0B/0C 13/14 3C/3D 3E/3F

***Table 7:* Disassembly Precautions**

Location (Hex)	Content	Status
D956-D95D DD75-DFA7 DFF6-DFF9	position-independent data	no change necessary
D000-D955 D95E-DD4A DFA8-DFF5 DFFA-DFFE	program segments	although relative branches remain intact, all absolute addresses in the range $D000-DFFF must be changed
DD4B-DD74	action addresses for directives (.WOR)	all must be changed
D27C-D27F D3AA-D3AD D9D4-D9D7	these are MSB/LSB bytes of position-dependent address used as input to SBR $D9EA in registers A and Y	change LDA#_____ and LDY#_____ operands to reflect new addresses

Function Input Routine for Applesoft

Applesoft permits the identification of a function through the use of the DEF FN command. This article describes a self-modifying subroutine which allows function input during program execution.

Roy E. Myers
William G. Miller III
The Pennsylvania State University
New Kensington, PA 15068

Software which accepts user-defined functions frequently receives them by giving the user the instructions such as

TYPE

10 DEF FN F(X) =
(YOUR FUNCTION)
(RETURN)

THEN TYPE

RUN 10 (RETURN)

This procedure is made necessary by the fact that Applesoft makes no provision for function input. How much simpler for the novice user to be asked:

ENTER F(X) = _

The program below allows this approach. The procedure receives the function as a string, then "transfers" the string to a line at the end of the program (line 5330), which initially reads

5330 DEF FN F(X) =

The "transfer" must take into account the following:

1. In a string, the characters *, +, −, /, =, ∧ are represented by the ASCII character codes 42, 43, 45, 47, 61, 94 (decimal). But, in a function the arithmetic operators *, +, −, /, =, ∧ are represented by the decimal codes 202, 200, 201, 203, 204. (See the *Applesoft Reference Manual*, pages 121, 138, 139.)
2. In a string the characters SIN are stored as 83, 73, 78 (decimal), whereas in a function SIN is represented by the decimal 233. A similar state of affairs exists for LOG, SQR, TAN, etc.

These cases are handled in lines 5080-5230. After translation, the appropriate code is POKEd into the function definition by line 5260. When the entire string has been transferred, line 5290 POKEs the code for ":" and the code for "RETURN".

```
10  LOMEM:  PEEK (176) * 256 +  PEEK (175) + 256
20  INPUT "ENTER F(X) = ";F$
30  GOSUB 5000
100  REM
200  REM
300  REM     PROGRAM
400  REM     BODY
500  REM     GOES
600  REM     HERE
700  REM
800  REM
4999  END
5000 FINI =  PEEK (176) * 256 +  PEEK (175) - 4
5010 FOLD = FINI
5020 L =  LEN (F$)
5030 STR =  PEEK (112) * 256 +  PEEK (111)
5040  FOR Q = 1 TO L
5050 A =  PEEK (STR + Q - 1)
5060 B =  PEEK (STR + Q)
5070 C =  PEEK (STR + Q + 1)
5080  IF A = 42 THEN A = 202
5090  IF A = 43 THEN A = 200
5100  IF A = 45 THEN A = 201
5110  IF A = 47 THEN A = 203
5120  IF A = 61 THEN A = 208
5130  IF A = 94 THEN A = 204
5140  IF A = 83 AND B = 71 AND C = 78 THEN A = 210: GOTO 5250
5150  IF A = 73 AND B = 78 AND C = 84 THEN A = 211: GOTO 5250
5160  IF A = 65 AND B = 66 AND C = 83 THEN A = 212: GOTO 5250
5170  IF A = 83 AND B = 81 AND C = 82 THEN A = 218: GOTO 5250
5180  IF A = 76 AND B = 79 AND C = 71 THEN A = 220: GOTO 5250
5190  IF A = 69 AND B = 88 AND C = 80 THEN A = 221: GOTO 5250
5200  IF A = 67 AND B = 79 AND C = 83 THEN A = 222: GOTO 5250
5210  IF A = 83 AND B = 73 AND C = 78 THEN A = 223: GOTO 5250
5220  IF A = 84 AND B = 65 AND C = 78 THEN A = 224: GOTO 5250
5230  IF A = 65 AND B = 84 AND C = 78 THEN A = 225: GOTO 5250
5240  GOTO 5260
5250 Q = Q + 2
5260  POKE FINI,A
5270 FINI = FINI + 1
5280  NEXT
5290  POKE FINI,58: POKE FINI + 1,177
5300  POKE FINI + 2,0: POKE FINI + 3,0: POKE FINI + 4,0: POKE FINI + 5,10

5310  POKE FOLD - 10,(FINI + 3) / 256
5320  POKE FOLD - 11,FINI + 3 - 256 *  PEEK (FOLD - 10)
5330  DEF  FN F(X) =
```

Before a user identifies a function, line 5330 reads:

```
5330 DEF FN F(X) =
```

If a user defines the function to be 2*X* SIN[X], the program changes line 5330 to read:

```
5330 DEF FN F(X) = 2*X*
   SIN(X) : RETURN
```

The remainder of the program consists of housekeeping chores. Set LOMEM high enough to allow room to input the function [line 10]. Since an input line is no more than 256 characters, LOMEM could be set to end-of-program + 256.

The function is transferred from string storage to the DEF FN F[X] = statement. Line 5030 identifies the beginning of string storage. The most recently defined string will begin at this location. The DEF FN F[X] = statement is at the end of the program and it is there that the program will POKE the code for the function. Line 5000 identifies the end-of-program memory location. It is necessary to subtract 4 from the actual end-of-program, in order to write over the end-of-program and end-of-line code. Line 5300 replaces the code.

In the memory locations preceding a program line Applesoft inserts a pointer to the beginning of the next line. Since additional code is being POKEd at the end of line 5140, the pointer preceding the line is incorrect. Lines 5310, 5320 reset the pointer so that it points to the end-of-program code.

The program segment 5000-5140 may be re-used several times within a program to re-enter the function, since the end-of-program pointer stored at locations 175 and 176 are not changed by the program.

Since the user of a program which includes this procedure may mis-type the function [e.g. leave out a "*" for multiply], the programmer may wish to have an appropriate ONERR GOTO statement before the first usage of the function.

Roy E. Myers is Associate Professor of Mathematics at The Pennsylvania State University, New Kensington, PA. His work with the Apple II is primarily concerned with computer graphics as an instructional tool in mathematics.

William G. Miller III is currently a programmer at Penn State, writing accounting programs for classroom instruction. He is also investigating the possibilities of opening a computer services business.

MICRO

Vector Calculations with a Microcomputer

Many physics and engineering problems involve the use of vectors. Unfortunately the required calculations are often tedious and susceptible to errors. This microcomputer program, compatible with PET, OSI, and Apple systems, speeds the process, and avoids costly errors.

Peter A. Koski
144 Delaware Avenue
Apartment F
Troy, New York 12180

At an engineering school, a myriad of problems are continually being solved. Most are examples of real world situations. Whether they be differential equations expressing some complex rate of change (world population growth, for example), or the moment of an applied force on a supporting member (engineering design), these are real problems. In solving these, the computer can be used as a very powerful tool. Programs used for problem-solving don't need to be masterpieces of structured programming, they only need to speed arrival at an answer.

In many cases, answers are only good approximations — very good when using the computer. For example, when trying to find a root of a polynomial equation, Newton's method is often used. This method involves refining an "educated" guess. Using a small program, many iterations may be made in a small fraction of the time it would take to manually make one refinement.

Definite integral problems in mathematics may be very well approximated by giving dx a very small finite dimension and summing along the given interval. Without the machine, this couldn't be done, as many hundreds of calculations must be made.

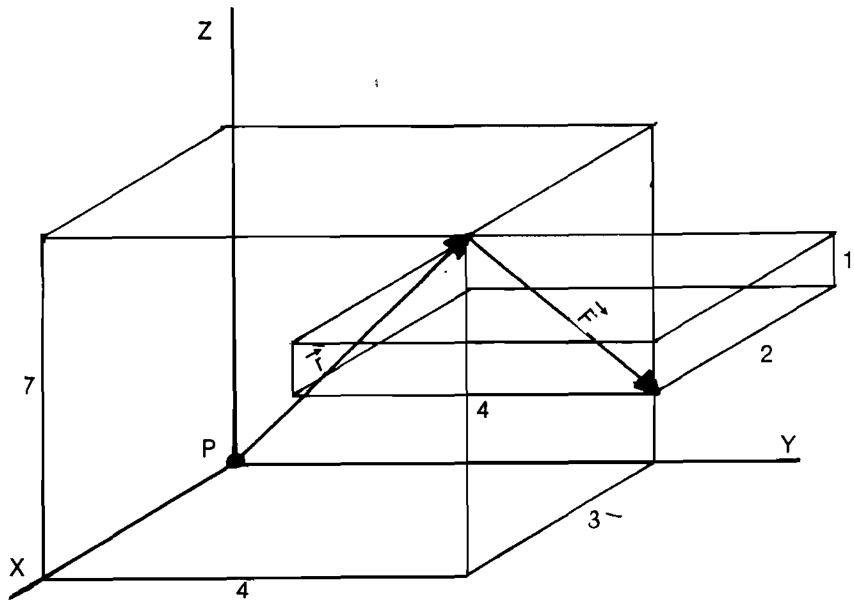


Figure 1: **Showing $\vec{F}$ and $\vec{r}$ as their components. (Note: not to scale.)**

```
1080 PRINT"      VECTOR CALCULATIONS"
1090 PRINT"      ==================="
1100 PRINT"      BY PETER ALAN KOSKI"
1105 PRINT
1110 PRINT"VECTORS USED BY THIS PROGRAM ARE"
1115 PRINT"REFERRED TO BY USER-DEFINED"
1120 PRINT"NAMES. PROVISION HAS BEEN MADE"
1130 PRINT"FOR 15 UNIQUE VECTORS."
1140 PRINT
1150 PRINT"VECTORS MUST BE DEFINED TO THE"
1160 PRINT"PROGRAM PRIOR TO ANY CALCULATIONS"
1165 PRINT"INVOLVING THEM. DEFINED VECTORS"
1170 PRINT"MAY BE REDEFINED IMPLICITLY OR"
1180 PRINT"EXPLICITLY."
1190 PRINT
1191 PRINT"KEY WORDS/SYMBOLS ARE RESERVED"
1192 PRINT"FOR PROGRAM USE AND THEREFORE"
1193 PRINT"MAY NOT APPEAR EMBEDDED OR ALONE"
1194 PRINT"IN A VECTOR LABEL:LIST, DELETE,"
1195 PRINT" X, .(PERIOD), /, +, -, =."
1196 PRINT:PRINT"PRESS ANY KEY TO CONTINUE"
1197 GETZ$:IFZ$=""THEN1197
1198 PRINT"□";:REM CLEAR SCREEN
1200 PRINT"OPERATIONS SUPPORTED / FORMAT :"
1210 PRINT
1220 PRINT"*VECTOR DEFINITION -- LABEL=I/J/K"
```

In all branches of science and engineering, vectors are often used in problem solving. A vector is a three-dimensional line of force, having both magnitude and direction. By defining forces, velocities, displacements, etc., as vectors, certain relationships may be easily developed and solved. Vectors are most often expressed in terms of their x, y, and z components.

Often, developing the vectors and vector equations can be time consuming enough without having to grind through the arithmetic to the final solution. That is the purpose of the program presented here.

VECTOR is a command-line processor which allows the user to define and operate on vectors. Program commands allow the user to DEFINE (enter vector and its label), DELETE (remove a vector from the work file), LIST (print a list of all vectors in work file), or CLEAR all vector definitions from the work file.

Operations available are addition, subtraction, dot products and cross products. Operations producing a resultant vector add the new vector's definition to the working file. If a previously-defined vector is specified as the resultant label, the vector will be re-defined and its previous value is lost, but the program will inform you of the redefinition.

Looking at an example, consider finding the moment (torque) of a force acting on a point. From mechanics, the moment, M, about point, P, is equal to the vector locating the force, crossed with the vector defining the force: $\vec{M} = \vec{r} \text{ x } \vec{F}$. Referring to figure 1, r may be expressed as (3,4,7) and F as (2,4,−1). The solution is arrived at, long-hand, by establishing a matrix and solving it. Alternately, the VECTOR program may be employed as follows (see sample run):

1. R=3, 4, 7 (define vector $\vec{r}$)
2. F=2,4,−1 (define vector $\vec{F}$)
3. M=RXF ($\vec{M}$ is defined as $\vec{r}$ cross $\vec{F}$)

As is seen, the output produced is the desired moment vector as well as the angle between the two original vectors.

Many time-consuming mistakes are eliminated by avoiding the long-hand arithmetic solutions.

Peter Koski is a sophomore at Rensselaer Polytechnic Institute majoring in Biomedical engineering and minoring in Computer Systems engineering. Most of his work is on an OSI Challenger 2-4P mini floppy system. Pete enjoys integrating hardware and software in optimizing his system.

```
1230 PRINT
1240 PRINT"*LIST DEFINED VECTORS -- LIST"
1250 PRINT
1254 PRINT"*DELETE VECTOR -- DELETE LABEL"
1256 PRINT
1258 PRINT"*CLEAR ALL VECTORS -- CLEAR"
1259 PRINT
1260 PRINT"*DOT PRODUCT -- LABEL1.LABEL2"
1270 PRINT
1280 PRINT"*CROSS PRODUCT -- RESULT=LABEL1XLABEL2"
1290 PRINT
1293 PRINT"*ADDITION -- RESULT=LABEL1+LABEL2"
1294 PRINT
1300 PRINT"*SUBTRACTION -- RESULT=LABEL1-LABEL2"
1310 PRINT
1315 PRINT"NO EMBEDDED BLANKS ARE PERMITTED IN"
1320 PRINT"COMMAND LINES (EXCEPT FOR DELETE)"
1322 PRINT
1324 PRINT"LABEL, LABEL1, LABEL2, RESULT"
1326 PRINT"REFER TO USER-DEFINED VECTOR NAMES."
1330 REM
1340 DIM LBL$(15),I(15),J(15),K(15)
1350 LBL=0
1360 DEF FNT(X)=INT(100*X)/100
1370 DEF FNC(X)=ATN(SQR(1-X↑2)/X)
1375 DEF FNS(X)=ATN(X/SQR(1-X↑2))
1380 DEF FND(X)=57.2957795*X
1400 REM
1410 REM        PROCESS COMMAND LINE
1420 REM
1440 PRINT:INPUT LN$
1450 IF LN$="" THEN PRINT"J":CLR:END
1460 REM
1470 REM      CHECK FOR LIST / CLEAR / DELETE COMMANDS
1480 REM
1490 IF LN$="LIST" THEN 5000
1500 IF LN$="CLEAR" THEN CLR: GOTO 1330
1510 IF LEFT$(LN$,6)="DELETE"THENT1$=RIGHT$(LN$,LEN(LN$)-7):
     GOTO6000
1520 REM
1530 REM      SCAN FOR IMPLICIT OR EXPLICIT DEFINITON
1540 REM                    OF  VECTOR
1550 REM
1560 FORI=1 TO LEN(LN$)
1570 T$=MID$(LN$,I,1)
1580 IF T$="/" THEN 1600
1585 NEXT I: GOTO 1700
1590 REM
1600 REM   EXPLICIT DECLARATION OF VECTOR / DOT PRODUCT
1610 REM
1620 T1$=""
1630 FOR I=1 TO LEN(LN$)
1640 T$=MID$(LN$,I,1)
1650 IF (T$="=")OR(T$=".")THEN OP$=T$: GOTO 1670
1655 T1$=T1$+T$
1660 NEXT I
1665 GOTO 9030
1670 T2$=RIGHT$(LN$,LEN(LN$)-I)
1680 GOTO 1900
1700 REM
1710 REM     IMPLICIT DECLARATION OF VECTOR
1720 REM
1730 RVL$=""
1740 FORI=1 TO LEN(LN$)
1750 T$=MID$(LN$,I,1)
1760 IF (T$="=")THEN 1810
1770 RVL$=RVL$+T$
1780 NEXT I
1790 GOTO 9030
1800 REM
1810 REM     ASSIMILATE T1$
1820 REM
1830 T1$=""
1840 FOR J=(I+1) TO LEN(LN$)
1845 T$=MID$(LN$,J,1)
```

(Continued)

MICRO

```
1850 IF (T$="+"ORT$="-"ORT$="."ORT$="X") THEN OP$=T$:GOTO 1895
1855 T1$=T1$+T$
1860 NEXT J
1865 PRINT"* ERROR IN COMMAND LINE *": GOTO 1440
1895 T2$=RIGHT$(LN$,LEN(LN$)-J)
1900 REM
1910 REM        JUMP TO ROUTINE FOR REQUIRED OPERATION
1920 REM
1930 IF OP$="=" THEN 2000
1940 IF OP$="." THEN 3000
1950 IF OP$="X" THEN 4000
1960 IF OP$="+" THEN 7000
1970 IF OP$="-" THEN 8000
2000 REM
2010 REM    STORE LABEL AND CORRESPONDING I/J/K VALUES
2020 REM
2030 FOR I=1 TO LBL
2035 IF LBL$(I)<>T1$ THEN 2050
2040 PRINT"* ";T1$;" RE-DEFINED *"
2045 GOTO 2100
2050 NEXT I
2052 IF LBL<15 THEN 2060
2055 GOTO 9040
2057 GOTO 1440
2060 LBL=LBL+1: I=LBL
2100 T$="": X$="": Y$=""
2110 FOR J=1 TO LEN(T2$)
2120 T$=MID$(T2$,J,1)
2130 IF T$="/" THEN X=VAL(X$): GOTO 2160
2140 X$=X$+T$
2150 NEXT J
2155 PRINTJ
2160 FOR K=(J+1) TO LEN(T2$)
2170 T$=MID$(T2$,K,1)
2180 IF T$="/" THEN Y=VAL(Y$): GOTO 2210
2190 Y$=Y$+T$
2200 NEXT K
2210 Z=VAL(RIGHT$(T2$,LEN(T2$)-K))
2220 REM
2230 REM     DEFINE VECTOR
2240 REM
2250 LBL$(I)=T1$: I(I)=X: J(I)=Y: K(I)=Z
2260 GOTO 1440
3000 REM
3010 REM       DOT PRODUCT CALCULATION
3020 REM
3030 FOR I=1 TO LBL
3040 IF LBL$(I)=T1$ THEN 3060
3050 NEXT I
3055 T0$=T1$:GOTO9060
3060 U1=I(I): U2=J(I): U3=K(I)
3070 FOR J=1 TO LBL
3080 IF LBL$(J)=T2$ THEN 3110
3090 NEXT J
3100 T0$=T2$:GOTO9060
3110 V1=I(J): V2=J(J): V3=K(J)
3130 UV=(U1*V1+U2*V2+U3*V3)
3140 U=SQR(U1↑2+U2↑2+U3↑2)
3150 V=SQR(V1↑2+V2↑2+V3↑2)
3160 PRINT
3170 PRINT T1$;" DOT ";T2$;" = ";FNT(UV)
3180 PRINT"COS(THETA) = ";FNT(UV/(U*V))
3190 PRINT"THETA = ";FNT(FNC(UV/(U*V)));
3192 PRINT" (";FNT(FND(FNC(UV/(U*V))));" DEGREES )"
3200 GOTO 1440
4000 REM
4010 REM     CROSS PRODUCT CALCULATION
4020 REM
4030 FOR I=1 TO LBL
4040 IF LBL$(I)=T1$ THEN 4060
4050 NEXT I
4055 T0$=T1$:GOTO9060
4060 U1=I(I): U2=J(I): U3=K(I)
4070 FOR J=1 TO LBL
```

(Continued)

(Continued from page 6)

were a first. Then they were pleased, very pleased. They thanked me profusely and said they'd do something about the thief immediately. Good! One pirate down (perhaps) and hundreds, at least, to go.

How many people, however, are afflicted with an ethical standard that makes them pay $125 (the lowest, legitimate discount price I've seen for VisiCalc) when they could get the program for $40? How many moral decisions can be bought for $85 plus tax? As long as a conscience can be bought for that or less, there will be software thieves popping up like spiders in spring.

I offer a proposal, then, to cut the feet from under the pirates. I challenge software manufacturers to stop the thieves as they start, before "protection" is forced upon us all. They can do it. I can't. Let each software manufacturer reward the first person reporting a software thief with a free, legitimate copy of the program being stolen or another of equivalent value. Then let the manufacturer's lawyer obtain a court injunction, at the least, against the thief's sales. A software buyer would then have a real incentive to keep the business honest. A software manufacturer would make a profit if he could prevent the thief from selling but one or two pirated copies. A software magazine would be able to devote its editorial page to technical rather than legal problems. A software thief would have to find a way to turn an honest buck and sleep better for it. Above all, each and every one of us would keep our taxes from going up still more and would retain a free-market economy in computer software; that, my friends, would keep *all* our costs down.

Let us not forget the user while we're protecting the manufacturer. Yes, we do need better service and support. Yes, we do need backup copies for our personal use. Yes, we do need the information to customize our programs. Yes, we do need lower cost software. But software piracy will cost us all more in the long run, both in dollars and in freedoms. We can stop it here. And now.

I have asked this magazine not to print my name or location. This is not because I don't sign up to what I say. Instead, I fear reprisals from thieves. If you feel that you must deal with a software thief, remember this advice offered me by a police detective. *All* thieves, when thwarted, readily turn to murder.

Anonymous

(Continued on next page)

```
4080 IF LBL$(J)=T2$ THEN 4110
4090 NEXT J
4100 PRINT"* ";T2$;" NOT IN WORKING FILE *":GOTO 1440
4110 V1=I(J): V2=J(J): V3=K(J)
4130 FOR I=1 TO LBL
4140 IF LBL$(I)<>RVL$ THEN 4160
4145 GOSUB9070
4150 GOTO4250
4160 NEXT I
4170 IF LBL<15 THEN 4240
4180 GOTO 9040
4190 GOTO 1440
4240 LBL=LBL+1: I=LBL: LBL$(I)=RVL$
4250 I(I)=(U2*V3)-(V2*U3)
4260 J(I)=(V1*U3)-(U1*V3)
4270 K(I)=(U1*V2)-(V1*U2)
4280 UV=SQR(I(I)↑2+J(I)↑2+K(I)↑2)
4290 U=SQR(U1↑2+U2↑2+U3↑2)
4300 V=SQR(V1↑2+V2↑2+V3↑2)
4310 PRINT
4320 PRINTT1$;" CROSS ";T2$;" = <";I(I);"I,";J(I);"J,";K(I);"K >
4330 PRINT"SIN (THETA) = ";FNT(UV/(U*V))
4340 PRINT"THETA = ";FNT(FNS(UV/(U*V)));"<";
4350 PRINTFNT(FND(FNS(UV/(U*V))));" DEGREES >"
4360 GOTO 1440
5000 REM
5010 REM    LIST VECTORS PRESENTLY ON FILE
5020 REM
5030 PRINT
5040 PRINT"LABEL";TAB(8);"I";TAB(14);"J";TAB(20);
5045 PRINT"K";TAB(24);"MAGNITUDE"
5050 PRINT"=====";TAB(8);"=";TAB(14);"=";TAB(20);
5055 PRINT"=";TAB(24);"========="
5060 PRINT
5070 FOR I=1 TO LBL
5075 MAG=SQR(I(I)↑2+J(I)↑2+K(I)↑2)
5080 PRINTLBL$(I);TAB(8);FNT(I(I));TAB(14);FNT(J(I));
5085 PRINTTAB(20);FNT(K(I));TAB(24);FNT(MAG)
5090 PRINT
5100 NEXT I
5120 GOTO 1440
6000 REM
6010 REM    DELETE LABEL T1$ FROM WORKING FILE
6020 REM
6030 FOR I=1 TO LBL
6040 IF LBL$(I)=T1$ THEN 6100
6050 NEXT I
6060 T0$=T1$:GOTO 9060
6070 GOTO 1440
6100 FOR J=I TO (LBL-1)
6110 LBL$(J)=LBL$(J+1)
6120 I(J)=I(J+1): J(J)=J(J+1): K(J)=K(J+1)
6130 NEXT J
6140 LBL=LBL-1
6150 GOTO 1440
7000 REM
7010 REM     VECTOR ADDITION
7020 REM
7100 FOR J=1 TO LBL
7110 IF LBL$(J)=T1$ THEN 7130
7120 NEXT J
7125 T0$=T1$:GOTO 9060
7130 U1=I(J): U2=J(J): U3=K(J)
7140 FOR K=1 TO LBL
7150 IF LBL$(K)=T2$ THEN 7180
7160 NEXT K
7170 T0$=T2$:GOTO 9060
7180 V1=I(K): V2=J(K): V3=K(K)
7200 FOR I=1 TO LBL
7210 IF LBL$(I)<>RVL$ THEN 7240
7220 GOSUB9070
7230 GOTO 7300
7240 NEXT I
7250 IF LBL<15 THEN 7295
```

(Continue

```
7260 GOTO 9040
7270 GOTO 1440
7295 LBL=LBL+1: I=LBL: LBL$(I)=RVL$
7300 I(I)=U1+V1
7310 J(I)=U2+V2
7320 K(I)=U3+V3
7330 PRINT
7340 PRINT T1$;" + ";T2$;" = <";I(I);"I,";J(I);"J,";K(I);"K >"
7350 GOTO 1440
8000 REM
8010 REM       VECTOR SUBTRACTION
8020 REM
8030 FOR J=1 TO LBL
8040 IF LBL$(J)=T1$ THEN 8080
8050 NEXT J
8060 T0$=T1$:GOTO 9060
8080 U1=I(J): U2=J(J): U3=K(J)
8090 FOR K=1 TO LBL
8100 IF LBL$(K)=T2$ THEN 8130
8110 NEXT K
8120 T0$=T2$:GOTO 9060
8130 V1=I(K): V2=J(K): V3=K(K)
8150 FORI=1 TO LBL
8160 IF LBL$(I)<>RVL$ THEN 8190
8170 GOSUB 9070
8180 GOTO 8250
8190 NEXT I
8200 IF LBL<15 THEN 8240
8210 GOTO 9040
8220 GOTO 1440
8240 LBL=LBL+1: I=LBL: LBL4(I)=RVL$
8250 I(I)=U1-V1
8260 J(I)=U2-V2
8270 K(I)=U3-V3
8280 PRINT
8290 PRINTT1$;" - ";T2$;" = <";I(I);"I,";J(I);"J,";K(I);"K >"
8300 GOTO 1440
9000 REM
9010 REM MESSAGES
9020 REM
9030 PRINT"* ERROR IN COMMAND LINE *":GOTO 1440
9040 PRINT"* DEFINITION SPACE EXCEEDED *"
9050 PRINT"* DELETION REQUIRED *":GOTO 1440
9060 PRINT"* ";T0$;" NOT IN WORKING FILE *":GOTO 1440
9070 PRINT"* ";RVL$;" REDEFINED *":RETURN
```

```
OSI C1P MODIFICATIONS
1197 POKE 57088,0: IF PEEK(57088)=255 THEN 1197
9130 FOR I=1 TO 24: PRINT: NEXT I
OSI C2-4P MODIFICATIONS
1000 GOSUB 9130
1197 POKE 57088,255: IF PEEK(57088)=1 THEN 1197
1198 GOSUB 9130
1450 IF LN$="" THEN GOSUB 9130: CLEAR: END
9100 REM CLEAR SCREEN--
9110 REM YOU MAY WISH TO USE YOUR
9120 REM OWN MACHINE LANGUAGE ROUTINE
9130 FOR I=1 TO 32: PRINT: NEXT I
9140 RETURN

APPLE MODIFICATION
1000 CALL -936: REM CLEAR SCREEN
1198 CALL -936: REM CLEAR SCREEN
```

(Letterbox continued)

Dear Editor:

I would like to relate a problem I encountered servicing an early KIM-1 computer. The 6502 uP had died for reasons unknown. The uP, when it was working, was of early enough vintage so that it did not have the rotate right ROR instruction. When a replacement uP was put in, the system still did not work. (The original had to be unsoldered and was replaced with a new one in a socket.) The problem was the crystal oscillator circuit. The original consisted of only a crystal across 6502 pins 3 and 37. When the uP was replaced, apparently the uP internal clock circuitry did not have enough gain in the updated process to sustain oscillation. I was able to modify the oscillator circuit by removing one side of the crystal from the circuit board, and adding 4 parts and wiring so that the circuit matched later-production KIM-1's. No circuit board cuts had to be made and the uP oscillator now works. Figure 1 shows the modification.

I would like to hear other readers' experiences servicing 6502-based uP systems. We could all learn about unusual problems which may be common to many different systems.

Eric R. Bean
927 S. 26 St.
South Bend, Indiana 46615

Write to MICRO

Do you have any comments, gripes or suggestions that might be valuable for other readers? Send your letters to Letterbox, MICRO, P.O. Box 6502, Chelmsford, MA 01824. If you've found bugs in any of our programs, or have discovered a better technique, write to Microbes and Updates, at the same address. We need to hear from you!

Phone Search

This program cross-links a customer's phone number with the actual record number of the customer file so that his phone number in effect becomes his computer account number.

Horst K. Schneider
5341 West Bayaud Ave.
Denver, Colorado 80226

Is this the age of numbers? It appears to be. Wherever I go I seem to need a social security number, an account number, a customer number, a subscriber number, ad nauseum.

Our modern data processing equipment has had a great deal to do with this trend. But is it really necessary to dehumanize relationships between humans by insisting that Bill is #68542 and Judy is #68671?

I am a businessman who "went computer" in 1979 with an Apple II with 48K, a printer and two 5" disk drives. While writing my programs for invoicing, statements, and so forth, I soon came to grips with the problem of assigning each customer a number. While I recognized the necessity of doing this I still could not suppress my feelings of aversion.

I decided to use a number my customers were almost as familiar with as their names — their telephone numbers. Asking customers for their phone numbers did not carry any stigma — in fact, I hoped it created in their minds the picture of an efficient office. Mail orders posed no problem either; very few business letterheads lack the phone number.

Now we all know that a customer file on a diskette stores the information in records numbered sequentially. That meant I needed a program to match a phone number with the actual customer number — or rather the record number of the customer file. So much for the reason this program came to be.

Applesoft BASIC is a fine tool for programming in general and I use it extensively, but there are cases when any BASIC is just too slow for the business environment. And you don't have to be a mathematical genius to realize that a program for this problem, written entirely in BASIC, would be agonizingly slow while the machine language routine would search through a list of 500 phone numbers in less than a second. But read on — all you need is BASIC. The assembly language listing is for those who enjoy assembly programming or for those who wish to get into it.

Writing the search and compare routine in machine language saves considerable memory space since we can nicely dispense with all the extra bytes that Applesoft tacks on when storing such a list of numbers as variables or strings.

There are actually three parts to this program. The main part, written in Applesoft BASIC allows you to add to the list, change the list, and search the list. Then there is a short machine language routine which the program invokes with CALL 38332. It then does the actual work of looking for the phone number in a list of numbers. Finally, there is a binary file containing all the phone numbers.

Enter the program exactly as shown, then type RUN 980. The last part of the program you typed in creates your machine language routine and saves it to your disks in Drive 1 and Drive 2. (You had a disk in each drive, didn't you?)

```
100  HIMEM: 36825
110  REM
120  REM         PHONE SEARCH
130  REM      BY HORST K. SCHNEIDER
140  REM
220 D$ =  CHR$ (4)
230  PRINT D$"BLOAD PH-95"
240  TEXT : HOME : VTAB 3: HTAB 8: PRINT "*   *   PHONE SEARCH  *  *"
250  VTAB 8: HTAB 10: PRINT "1 - SEARCH LIST"
260  VTAB 10: HTAB 10: PRINT "2 - ADD TO LIST"
270  VTAB 12: HTAB 10: PRINT "3 - CHANGE LIST"
280  VTAB 14: HTAB 10: PRINT "4 - SAVE ALL CHANGES"
290  VTAB 16: HTAB 10: PRINT "5 - RETURN TO MAIN"
300  VTAB 20: PRINT "YOUR CHOICE, PLEASE? -": VTAB 20: HTAB 24: GET Q$: PRINT
     :A =  VAL (Q$): IF A < 1 OR A > 5 THEN  GOSUB 740: GOTO 300
310  VTAB 23: PRINT "(- RESPOND WITH 'X' TO RETURN TO START)": POKE 35,22

320  ON A GOTO 330,420,530,830,800
330  HOME : VTAB 5: HTAB 4: PRINT "*  *  SEARCH PHONE LIST  *  *"
340 Y = 1: VTAB 10: INPUT " - PHONE NO.: ";A$: IF A$ = "X" THEN 240
350  GOSUB 700: GOSUB 690: IF  NOT Y THEN  GOSUB 740: GOTO 340
360  POKE 38331,A: POKE 38330,B: POKE 38329,C: CALL 38332
370 Y = 1: GOSUB 770: IF  NOT Y THEN 400
380 A =  PEEK (6) +  PEEK (7) * 256
390  VTAB 14: PRINT "CUSTOMER NO.: ";A / 3: GOTO 400
400  VTAB 19: PRINT " - ANOTHER SEARCH? - Y/N ": VTAB 19: HTAB 28: GET Q$
     : IF Q$ = "Y" THEN 330
410  GOTO 240
420  HOME : VTAB 3: HTAB 8: PRINT "*  *  ADD PHONE NO.  *  *"
430 F =  PEEK (38327) +  PEEK (38328) * 256: GOSUB 730
440 Y = 1: VTAB 12: INPUT " - NEW PHONE NO.: ";A$: IF A$ = "X" THEN 240
450  GOSUB 700: IF  NOT Y THEN  GOSUB 740: GOTO 440
460  IF F < 36827 THEN  GOSUB 750: GOTO 760
```

Now you save your program on disk and it's ready to go to work for you.

I have purposely not compressed the code to make it easy to change or relocate. It is also easy to increase the list size by multiplying by three the number of additional phone numbers you wish to store and subtracting this number from 36825 in line 100, from 36827 in line 460 and from 36826 in line 870, and adding it to 1574 in line 870.

If you operate with only one disk drive (and in a business application that is courting disaster) you should delete the references to "J$" at the end of the main program.

When entering a phone number you may or may not use a hyphen (either 256-5515 or 2565515 is acceptable).

The program will tell you how many phone numbers you have stored and will also alert you to a 'LIST-FULL' condition. In my business we delete a customer by changing his phone number to 0000000. When adding a customer we always first search for a zero string and use that spot for our new entry.

As shown, it is a stand-alone program but can easily be incorporated into a larger one by using a hook after line 900, setting HIMEM: at the beginning of the main program, and deleting line 100.

The program is only a part of a larger program that handles pricing, billing, inventory control and statements, making the customer number available directly to the appropriate routines.

One last comment: All REM line numbers end with a '5' (except starting lines) for easier identification, even at 'List' speeds, in case you want to remove them from your WORKING program.

Horst K. Schneider is a businessman (both wholesale and retail) who enjoys the challenge that programming provides. His first programming effort was fairly ambitious. That program did all his pricing, invoicing, inventory control and monthly statements as well as other tasks such as printing mailing labels. He recently sold his business and has retired into writing software.

MICRO

```
470  GOSUB 690: POKE F,A: POKE F - 1,B: POKE F - 2,C: POKE F - 3,255
480 F = F - 3:H =  INT (F / 256):L = F - H * 256
490  POKE 38327,L: POKE 38328,H: GOSUB 730
500  VTAB 16: PRINT " - CUSTOMER NO.: ";(38326 - F) / 3
510  VTAB 19: PRINT " - ANOTHER ENTRY? - Y/N": VTAB 19: HTAB 26: GET Q$: PF
     : IF Q$ = "Y" THEN 420
520  GOTO 240
530  HOME : VTAB 3: HTAB 6: PRINT "*  *  CHANGE PHONE NO.  *  *"
540  VTAB 12: PRINT " - CUSTOMER NO. : ": VTAB 13: PRINT "(OR OLD PH. NO.
     )": VTAB 12: HTAB 20: INPUT "";A$: IF A$ = "X" THEN 240
550  IF  LEN (A$) < 5 THEN N = 3 *  VAL (A$): GOTO 600
560 Y = 1: GOSUB 700: IF  NOT Y THEN  GOSUB 740: GOTO 540
570  GOSUB 690: POKE 38331,A: POKE 38330,B: POKE 38329,C: CALL 38332
580 Y = 1: GOSUB 770: IF  NOT Y THEN 670
590 N =  PEEK (6) +  PEEK (7) * 256: GOTO 610
600 A =  PEEK (38329 - N) * 65536 +  PEEK (38328 - N) * 256 +  PEEK (3832
     7 - N):A$ =  STR$ (A)
610 A$ =  LEFT$ (A$,3) + "-" +  RIGHT$ (A$,4)
620  VTAB 16: PRINT "OLD: ";A$
630  PRINT "NEW: ";A$: VTAB 17: HTAB 6: INPUT "";A$
640 Y = 1: GOSUB 700: IF  NOT Y THEN  GOSUB 740: GOTO 630
650  GOSUB 700: IF  NOT Y THEN  GOSUB 740: GOTO 640
660  GOSUB 690: POKE 38329 - N,A: POKE 38328 - N,B: POKE 38327 - N,C
670  VTAB 19: PRINT " - ANOTHER CHANGE? - Y/N": VTAB 19: HTAB 28: GET Q$:
      PRINT : IF Q$ = "Y" THEN 530
680  GOTO 240
685 :: REM ::CONVERT TO MODULO
690 A =  INT (X / 65536):B =  INT (X / 256) - A * 256:C = X - A * 65536 -
     B * 256: RETURN
700  IF  MID$ (A$,4,1) = "-" THEN A$ =  LEFT$ (A$,3) +  RIGHT$ (A$,4)
710  IF  LEN (A$) <  > 7 THEN Y = 0
720 X =  VAL (A$): RETURN
730  VTAB 5: HTAB 1: CALL  - 958: VTAB 5: PRINT "TOTAL LISTINGS: ";(38326
      - F) / 3: RETURN
735 :: REM ::ILL. ENTRY WARNING
740  VTAB 21: PRINT " - ILLEGAL ENTRY - PLEASE REENTER": FOR I = 1 TO 120
     0: NEXT : VTAB 21: CALL  - 958: RETURN
745 :: REM ::AUDIO WARNING
750  FOR I = 1 TO 3: FOR J = 1 TO 15:X =  PEEK ( - 16336):: NEXT : FOR K =
     1 TO 10: NEXT K,I: RETURN
760  TEXT : HOME : VTAB 16: PRINT " - OOPS - PAST PRESENT STORAGE CAPACIT
     Y": VTAB 18: HTAB 30: PRINT "SORRY -": VTAB 23: GET Q$: GOTO 240
770  IF  PEEK (38331) = 255 THEN  VTAB 14: PRINT " - NO SUCH NO. ON RECOR
     D -":Y = 0
780  RETURN
790  TEXT : HOME : VTAB 12: PRINT "- DO YOU WISH TO RETURN TO MAIN"
800  VTAB 14: PRINT "WITHOUT SAVING CHANGES - ? - Y/N:": VTAB 14: HTAB 39
     : GET Q$
810  IF Q$ <  > "Y" THEN 240
820  END :: REM ::DELETE 'END' IF RETURN HOOK IN 905 IS USED
830  TEXT : HOME : VTAB 12: HTAB 8: PRINT "*  *  BUSY  *  *"
840 J$ = ",D2"
860  PRINT D$"UNLOCK PH-95";J$
870  PRINT D$"BSAVE PH-95 ,A36826,L1574"
880  PRINT D$"LOCK PH-95"
890  IF J$ = ",D2" THEN J$ = ",D1": GOTO 860
900  TEXT : HOME : VTAB 14: HTAB 12: PRINT "*  *  END  *  * ": POKE 37,22
     : PRINT
905 :: REM ::INSERT HOOK HERE
910  DEL 905,1070
915 ::
920 :: REM ::THIS PROGRAM WILL
930 :: REM ::ENTER THE
940 :: REM ::MACHINE LANGUAGE
950 :: REM ::PORTION AND THEN
960 :: REM ::DELETE ITSELF.
970 ::
980  DIM A(73)
990  FOR I = 0 TO 72: READ A(I)
1000  POKE 38327 + I,A(I): NEXT
1010  DATA 182,149,0,0,0,169,179,133,6,169,149,133,7,169,184,133,8,169,14
      9,133
1020  DATA 9,160,3,208,6,169,255,209,6,240,38,177,8,209,6,240,15,56,165,6

1030  DATA 233,3,133,6,160,3,176,233,198,7,208,229,136,208,232,56,169,182
      ,229,6
1040  DATA  133,6,169,149,229,7,133,7,96,141,187,149,96
1050 D$ =  CHR$ (4)
1060  PRINT D$"BSAVE PH-95,A1000,L10,D1"
1070  PRINT D$"BSAVE PH-95,A1000,L10,D2": GOTO 840
```

MICRO Club Circuit

Apple Bit 'N Pieces Educators Group
This group of math teachers meets on the first Thursday of each month except during summer. Purpose of group is to exchange ideas and programs. For more information please contact:

Pat Calabrese, Dept. Chairman
JS Wilson Middle School
Apple Bit'N Pieces
Educators Group
901 West 54th Street
Erie, PA 16509

Toronto PET Users Group
Membership in this fast-growing club now totals 430. Members receive a subscription to *The Target*, as well as access to all programs (1400) in the disk library. Regular dues are $20, and student and associate dues are $10 per year. For more information contact:

Chris Bennett, Secretary
Toronto PET Users Group
381 Laurence Avenue West
Toronto, Ontario
Canada M5M 1B9

Computer Programs for Investment Management
As an investor, you can obtain professional-quality management programs by belonging to a professional, nonprofit group called the Micro-Computer Investors Association (MCIA). Since 1977 the Association has published a journal, *The MicroComputer Investor*. The journal has a wealth of information and programs for investors who use microcomputers. For membership application and an index of all programs and articles published to date, send $3.00 for an information packet to:

Jack Williams, MCIA
902 Anderson Drive
Fredericksburg, Virginia 22401

Forth Interest Group
This group meets the fourth Saturday of the month at noon and has a membership of over 1200. The club puts out a publication called "Forth Dimensions." For further information, contact:

Jim Flournay Ancon
17370 Hawking Lane
Morgan Hill, California 95037

Attention Educators
Affiliated with the Cleveland Digital Group, this club's primary objective is the investigation, discovery, and exchange of functional and innovative computer-aided instruction ideas among interested computer, minicomputer, or microcomputer users and/or owners. Monthly meetings are held every third Sunday at the Cleveland Heights-University Heights main library, 2345 Lee Road, Cleveland Heights, Ohio. If interested, send a self-addressed stamped business envelope to:

Joyce Townsend
P.O. Box 18431
Cleveland Heights, Ohio 44118
or call (216) 932-6799

Dental Computer Newsletter
For medical and dental professionals using micro- and minicomputers for treatment and office purposes. Membership is over 1500. Meetings are held at the address below. For more information, contact:

E.J. Neiburger, DDS, President
1000 North Avenue
Waukegan, Illinois 60085

OSI — MUG
Ohio Scientific
Michigan User's Group
This group has a membership of approximately 130 people. It is interested in contacting other user groups and anyone wishing to become a member. For information write:

Ralph V. Johnson, Sec.
OSI — MUG
3247 Lakewood Avenue
Ann Arbor, Michigan 48105

Apple Power Users Group
This group meets the second or third Wednesday of every month (7:00 p.m.) at Syosset High School, Syosset, Long Island, New York. Jim Lyons is president of the club, whose membership is now 110 and expanding. There is a bimonthly newsletter, "The Pits," and yearly dues are $20 which includes a free subscription to the newsletter, computer hardware and software discounts, feature demonstrations and presentations at all meetings and an extensive program library. For information concerning membership, library program exchanges, newsletter exchanges, etc., please contact:

Apple Power, c/o m. Lack
8 Division Street
Holtsville, Long Island,
New York 11742

MICRO (East Brunswick Junior Computer Club)
This group whose members are in grades 7-12 meets twice a month at the East Brunswick Public Library. The main purpose of the group is to teach beginners about computers. For additional information, please contact:

Larry Kaplan, Secretary
28 Green Hills Road
East Brunswick, NJ 08816

Microcomputer Users International
This club meets on the third Tuesday of each month. *Northern Bytes* is the group's monthly newsletter. For more club information, or to arrange for a newsletter exchange, contact:

Jack Decker, Newsletter Editor
1804 West 18th St., Lot 155
Sault Ste. Marie, MI 49783

The Apple Guild
The Apple Guild is an organization whose purpose is to promote the interchange of information and applications among Apple microcomputer users. In addition to holding monthly meetings, The Guild supports a sophisticated, computerized, telecommunication system (617-767-1303); maintains a collection of hardcopy material and software at its Apple Resource Center located at Massasoit Community College (Brockton, MA); and plans to publish a quarterly journal. Membership requests and other inquiries should be sent to:

The Apple Guild
P.O. Box 371
Weymouth, MA 02188

Wondai Apple Users Group (W.A.U.G.)
This group of 20 members meets twice a month, and publishes a monthly newsletter called *Waug-Waug*. The group aims to exchange and promote Apple ideas and reviews. Contact:

Dr. P. Lip
P.O. Box 19
Wondai Old 4606
Australia

OSI Users Group Wellington
This group of 30 people meets on the 3rd Thursday of each month at 7:30 p.m. at Computer Consultants Ltd., Wingate Lower Hutt. The club arranges a guest speaker, and provides an OSI microcomputer for members to use. Aims include exchange of ideas and information, plus tuition of machine code. Membership is $5 annually. Contact:

Derryl Cocks (Vice Pres.)
27 Tawa Terrace
Tawa, Wellington, New Zealand

It's Time to Stop Dreaming, Part 2

Robert M. Tripp
Editor/Publisher
MICRO

Part 1 (MICRO 37:9) presented the Motorola 6809 microprocessor — a candidate for serious consideration as a successor to the 6502. The four major points made were:

1. No manufacturer has announced plans to develop an improved 6502;
2. The 6809 is closely related to the 6502 in basic architecture, philosophy and instruction set;
3. The 6809 has a number of improvements which make it very powerful and a worthy successor to the 6502; and,
4. While the 6809 is relatively new, there are already a large number of hardware and software products available. These include upgrades for existing 6502 systems — the SYM and Apple for example — as well as totally new products, such as Commodore's brand new "Micro-Mainframe," the Radio Shack Color Computer, and others.

This article, part 2, will concentrate on describing some of the improvements which make the 6809 a rather remarkable device.

The 6502, 6800 and 8080 microprocessors, were designed to be process controllers, not microcomputer building blocks. Therefore, while they could be used as the "brains" of microcomputers, the many design trade-offs that had been made based on their intended use as relatively simple, ROM-oriented process controllers resulted in limitations when used in microcomputers. The designers of the 6809 had a totally different charter. They set out from the start to build a new device which would be used primarily as the intelligence of a microcomputer. Many of the individual new features work together to provide important new capabilities.

Position-Independent Code

In a dedicated microprocessor controller application there may not be any reason to write position-independent code. After all, the program is probably in ROM and is unique to the application. There are, however, many good reasons to write position-independent code in a general-purpose microcomputer. Different hardware configurations may require that the program reside in different address spaces. In a disk-based system, various software modules may want to be resident in numerous combinations. If each module can only run in a specific address space, then there are severe restrictions on which modules may co-exist. Given a sufficiently well-defined set of interfacing rules, it will even be possible to write software modules which can operate on a variety of microcomputers.

There are four major improvements the 6809 offers which directly affect its capability to support position-independent code. These include:

1. Long Branches which permit relative branching to any location;
2. A Branch to Subroutine instruction which permits relative branching to a subroutine;
3. Addressing relative to the Program Counter;
4. The Load Effective Address instruction which permits the address calculated by many complex addressing modes to be directly accessed.

Long Branch. (This does not refer to the saloon which was so popular in Gunsmoke.) As anyone who has worked in assembly level programming on the 6502 can testify, the limitation of the Branch instructions to plus/minus only 128 locations (decimal) can be a real nuisance as well as a real restriction. The 6809 instruction set includes two addressing modes for all of the Branch instructions.

Short — identical to the 6502 with one byte of offset requiring the target address to be within 128 bytes of the current program counter; and,

Long — which has two bytes of offset permitting the target address to be anywhere in the normal 64K memory.

The Long Branch obviously makes life easier by eliminating the need for branches to branch to branches, etc., to accomplish a branch to an address outside the one byte addressing range. Since it is program-counter-relative, it provides most of the solution to the problem of transferring control to other addresses in a relative way, which makes it position-independent. The 6502 "can" branch to any relative location in memory by having one branch go to another branch to another branch until the target is reached, but this can get so complicated and difficult to maintain that it is generally not practical. The Long Branch improvement in the 6809 is significant.

Branch to Subroutine. The 6502 does not have any direct method for making a relative branch to a subroutine. This is probably the single most serious problem encountered in trying to write position-independent code. There is no simple solution. One can make all subroutine calls via a fixed table, which is itself updated as the code is moved around in memory. Or a special software processor can be written, which traps all subroutine calls and calculates the actual address. Another alternative is that code can be written which will function in a manner similar to a subroutine but will perform some sort of test to determine where to return to so that it may be called via a normal branch. There are other methods as well, but, every technique for getting around the lack of a Branch to Subroutine instruction involves tricky code, additional memory, extra instruction cycles, and can be difficult to maintain and/or debug.

The 6809 does have a straightforward Branch to Subroutine (BSR) which operates exactly as one would expect. It is just like the Jump to Subroutine (JSR) of the 6502 except that it is a branch relative to the Program Counter, not an absolute jump. Like all other Branch instructions on the 6809, it can be short (BSR — one byte offset) or long (LBSR — two byte offset), thereby allowing the Branch to have a target anywhere in memory.

```
        BSR NEWTST
        (control will go to NEWTST)
        (subroutine will return control to
        here)

NEWTST (same code)

        ...

        RTS
        (Return from Subroutine
        instruction)
```

Addressing via the Program Counter. The improved Branch instructions solved one major PIC problem — that of passing program control in a relative fashion throughout the whole memory and to subroutines. The major problems remain: how to address data (individual values, tables, lists, messages, etc.) in a relative way to preserve the PIC. On the 6502 there is no simple way to access data relative to the current value of the program counter. Some tricks, similar to those mentioned to provide relative subroutine calls, can be used, but they all have drawbacks and increase both time and space requirements. The 6809 provides Program Counter Relative Addressing. This form of addressing is almost identical in concept to the Branch addressing. The offset may be either one byte or two bytes, and is added to the current value of the Program Counter Register (PCR) to determine the absolute address. While the Branch operation is normally written in the form

```
        BEQ JUNK
```

it actually adds the signed value of JUNK to the Program Counter Register. The Branch may therefore be considered to be of the form:

```
        BEQ JUNK,PCR
        (add the signed value of JUNK,
        which may be one or two bytes,
        to the Program Counter and set
        the Program Counter to the new
        value)
```

It can then be seen that the Program Counter Relative address is identical since it has the form:

```
        LDA JUNK,PCR
        (add the signed value of JUNK,
        which may be one or two bytes,
        to the Program Counter and load
        the A register from the calculated
        address)
```

This provides the solution for accessing any single memory location in a PIC fashion. The memory at any address may be loaded, stored, incremented, tested, compared, complemented, and so forth with PCR addressing, thereby providing support for PIC.

Loading Effective Addresses. While the Program Counter Relative addressing supports accessing single memory address, it would be very useful to be able to get the absolute address of a table, list or message into an index register so that the whole table could be readily accessed. This is one of the features of a very useful new 6809 instruction: Load Effective Address (LEA). The application of this instruction here is but one of many uses. Other uses will be discussed later. The LEA instruction, in combination with the PCR addressing, allows an index register to be loaded with an absolute address which is calculated relative to the current Program Counter. The form is identical to that discussed for the Branch and Program Relative Addressing:

```
        LEAX TABLE,PCR
        (add one or two byte offset to the
        current Program Counter and
        place this value — the Effective
        Address — in the X index
        register)
```

The X register now contains the absolute address of the location TABLE. Since the 6809 supports a number of indexing modes — Zero Offset Indexed, Constant Offset Indexed, Accumulator Offset Indexed, Auto Increment/Decrement Indexed and Indexed Indirect — this ability to obtain the absolute address relative to the Program Coun solves a lot of the normal proble encountered in generating PIC.

Position-Independent Summa While writing PIC on the 6502 is pos ble, it is not an easy task and alwa adds considerable complexity a overhead. I wrote two versions of video driver to run anywhere in an AI SYM or KIM. In both versions, the p gramming required to provide PIC w more complex than any of the code quired to support the numerous vid functions! The support that the 68 has added would make a similar mod almost trivial to create PIC. The ma improvements of the 6809 which dire ly support PIC are: Long Branch which are relative to any address fr any address; the Branch to Subrouti instruction which permits relative dressing of subroutines; the addressi of locations relative to the Progr Counter; and the Load Effective Addr instruction which can calculate t absolute value of a relative address a make it available for the numer indexed instructions and indexi modes. With all of these added suppo for position-independent coding, there no reason to write position-depend code on a 6809 microprocessor-bas system.

The Versatile Stacks

The Stack plays a very import part in the operation of every signific microprocessor, including the 65(The Stack is a basic part of the hardw interrupt processing, is required for s ing the return address during a s routine call, and can be used as te porary storage, to pass parameters, a so forth. Unfortunately, the 6502 off only limited Stack support. It has o one Stack, which is limited to 256 by and must reside on page one (0100 01FF). There are very few Stack instr tions: TXS (set Stack Pointer from register), TSX (put Stack Pointer int register), PHA (Push A register Stack), PLA (Pull A register from Stac PHP (Push Status on Stack), and P (Pull Status from Stack). Other instr tions such as JSR, RTS and RTI use Stack, but would not normally be c sidered Stack support instructio Although there are many uses would like to make of the Stack, on 6502 the support is limited.

The 6809 makes full use of the St concepts. This is done in a number ways:

1 . There are two Stacks — a Syst Stack and a separate User Sta

2. The Stack Pointers have all of the same indexing modes as the X and Y registers.

3. Any combination of registers may be Pushed/Pulled from either Stack in a single instruction.

4. The Load Effective Address may be used with the Stack registers.

5. Each Stack register is 16-bit, meaning that Stack may be up to 64K bytes and may be located anywhere in memory.

Each of these improvements to the Stack support can have varying degrees of importance, depending upon the application. The overall effect of these improvements is the creation of a whole new facility with new ways of performing many programming tasks. Since the 6502 has limited Stack support it is not surprising that the Stack is not normally used for much beyond its subroutine, interrupt, and occasional short-term storage. With the 6809 features, many new ways of using the Stack become possible.

One difficulty in using the Stack of the 6502 is that it must be "shared" with the hardware. Interrupts and subroutine calls are forever putting things on and taking things off the Stack. The User Stack on the 6809 does not have this problem. All hardware and subroutine service is handled by the System Stack, leaving the User Stack alone. Since all of the indexing operations are available to the two Stack Pointers, which are treated as two additional 16-bit registers, many operations are possible on the Stack that would be too complicated for the 6502.

A number of programming problems may be solved using Stacks. These include position-independent, re-entrant, and recursive coding. Many high level languages can be programmed to be more efficient if there can be free and easy access to Stack operations. An example of the improved 6809 Stack operation is the use of the Load Effective Address instruction to modify the Stack Pointer. Compare the following processes for moving the Stack Pointer forward 20 (decimal) positions on the 6502 and the 6809.

6502:

STX XTEMP		Save X register in some memory location
STA ATEMP		Save A register in some memory location
TSX		Put current Stack Pointer into X register
TXA		Move current Stack Pointer into A register
CLC		Clear carry for addition
ADCIM	#$14	Add 20 (decimal) to the current value
TAX		Put new value into X register
TSX		Put new value into Stack Pointer
LDA	ATEMP	Restore A register
LDX	XTEMP	Restore X register

6809:

LEAS	14,S	Load Effective Address into Stack register = current Stack value + 20 (decimal)

This operation could be used to clean up the Stack after it has been used for temporary storage. It is obviously very simple on the 6809, and probably more trouble than it is worth on the 6502.

The 6809 makes it easy to access data on the Stack. The Transfer instruction can be used to copy the Stack Pointer into any other index register, and then operations can be made relative to the index register without disturbing the Stack Pointer.

```
TFR X,S
Will copy the 16-bit Stack Pointer
to the X register
```

All of the indexed operations may now be performed on the X register without any involvement of the Stack Pointer. Typical applications would be to pass subroutine parameters between the calling program and the subroutine on the Stack with the index register being used to access the various paramaters in any order as required. Then, as the Stack Pointer may be changed due to various operations, the reference pointer can stay fixed.

```
LDA -5,X
to refer to a location five locations
below the position of the Stack
Pointer at subroutine entry
```

The useful programming techniques which depend on stack-type operations are very well supported by the 6809.

Other New Products

The Radio Shack new Color Computer is 6809-based. At this time I do not have enough information to give a full report of its features, but hope to have this information for a column soon.

Commodore has announced the "Micro-Mainframe," a new 6809-based microcomputer with a large body of software developed by Waterloo Computering Systems. This product will be in the $2000 range, complete with micro BASIC, micro PASCAL and other languages, and is supposed to be available by the end of this year.

The Computerist has announced that its new multi-controller board will offer the 6809 as one of its many options. The board will provide controllers for floppy disks, IEEE-488 bus, RS-232 communication, cassette interface, up to 56K memory in any combination of RAM, ROM and EPROM, plus parallel and serial I/O ports. Initial deliveries are scheduled for this summer.

Last month's column mentioned a number of manufacturers of 6809-based hardware and software, but did not give the addresses. A "6809 Resource List" at the end of this installment provides this additional information. If your company has a 6809-based product, send along as much information as possible to me so that you may be covered in future columns. If you have had experience with the 6809, in almost any environment and on any equipment, please consider writing about it for MICRO. Our readers are anxious to keep abreast of the rapid developments in this area and will appreciate hearing from fellow readers.

6809 Resource List

Technical Systems Consultants Inc.
Box 2570
West Lafayette, Indiana 47906

Percom Data Co., Inc.
211 North Kirby
Garland, Texas 75042

Softech Microsystems, Inc.
9494 Blue Mountain Road
San Diego, California 92126

Computer Systems Center
7413 N. Lindbergh Boulevard
St. Louis, Missouri 63132

Ackerman Digital Systems
110 N. York Road 208
Elmhurst, Illinois 60126

Canon USA Inc.
10 Nevada Drive
Lake Success, Long Island
New York 11040

Commodore Business Machines, Inc.
681 Moore Rd.
King of Prussia, Pennsylvania 19406

(Continued)

Motorola Semiconductor Prod. Inc.
P.O. Box 20912
Phoenix, Arizona 85036

Smoke Signal Broadcasting
31336 Via Collinas
Westlake Village, California 91361

Forth Inc.
2309 Pacific Coast Highway
Hermosa Beach, California 90254

Microware Systems Corp.
5835 Grand Avenue
Des Moines, Iowa 50304

Phoenix Digital
2315 North 35th
Phoenix, Arizona 85009

Software Dynamics
211 West Crescent
Anaheim, California 92801

Informer Inc.
P.O. Box 91054
Los Angeles, California 90009

Stellation Two
P.O. Box 2342
Santa Barbara, California 93120

The Computerist Inc.
34 Chelmsford Street
Chelmsford, Massachusetts 01824

MICRO

MICRO

New Publications

Mike Rowe
New Publications
P.O. Box 6502
Chelmsford, MA 01824

General 6809

6809 Microcomputer Programming & Interfacing, With Experiments by Andrew C. Staugaard, Jr. Howard W. Sams & Co., Inc. (4300 West 62nd Street, Indianapolis, Indiana 46268), 1981, 270 pages, diagrams, photos, tables, 5 3/8 × 8½ inches, paperbound.
ISBN: 0-672-21798-8 $13.95

This book is designed as a tutorial type of text or "cookbook" for a first exposure to the 6809, a *high-performance* 8-bit microprocessor, or to high-performance microprocessors in general. According to the author, the 6809 approaches the performance of many 16-bit devices, without the overhead costs required to engineer such a 16-bit system.

CONTENTS: *Fundamental 6809 Concepts and Chip Structure*—Introduction; Objectives; 6809 Evolution and Design Philosophy; 6809 Improvements; 6809 Chip Structure; Review Questions; Answers. *6809 Addressing Modes*—Introduction; Objectives; Inherent, Immediate, and Extended Addressing; Direct Addressing and the Direct Page Register Relative Addressing; Indexed Addressing; Post Byte; Indirect Addressing; Register Addressing; Review Questions; Answers. *6809 Registers and Data Movement Instructions*—Introduction; Objectives; 6809 Internal Register Format; Data Movement Instructions; Review Questions; Answers. *Arithmetic, Logic, and Test Instructions*—Introduction; Objectives; Arithmetic Instructions; Logic Instructions; Test Instructions; Review Questions; Answers. *Branch and Miscellaneous Instructions*—Introduction; Objectives; Branch Instructions; Miscellaneous Instructions; Review Questions; Answers. *6809/6809E Input and Output Signals*—Introduction; Objectives; 6809 Pin-Outs; 6809E Pin-Outs; Review Questions; Answers. *6809/6809E Interfacing and Applications*—Introduction; Objectives; A Minimum 6809 System; An Expanded 6809 System; Multiprocessor Systems; Remote Data Acquisition; The MEK6809D4 Microcomputer Evaluation System. *Appendices A: 6809/6809E Ins tion Set*—Operation Notation; Reg Notation; Definitions of Executable Ins tions. *B. The 6820/6821 Peripheral face Adapter (PIA)*—6821 Funct Description; 6820/6821 Pin Assignm PIA Interfacing and Addressing; PL itialization and Servicing; Review (tions; Answers. *C. Specifica Sheets*—MC6809/MC68A09/MC68 MC6809E / MC68A09E / MC68] MC6829; MC6839; MC6842; MEK68C MEK6809D4/MEK68KPD. *D. MC Instruction Set Summary. Index.*

Pascal

Pascal Primer by David Fox Mitchell Waite. Howard W. San Co., Inc. (4300 West 62nd St Indianapolis, Indiana 46268), 1981 pages plus tear-out UCSD P reference card, line drawings, grams, listings, 8 5/8 × 11 1/8 inc cardstock cover with Wire-O bind
ISBN: 0-672-21793-7 $1

This book was designed for people have dabbled in BASIC and war learn programming in Pascal. authors are committed to he readers master "Pascal without te

CONTENTS: *Introduction: An Overvi Pascal*—Skip This Chapter; How This Is Organized; What Is Not Included; W Pascal?; The Crisis That Gave Bir Pascal; The Rat's Nest Analogy to P Not a Black and White World; Why Is I Special?; The Parts of Pascal; A History of the Language; A Present Da ample: Apple Pascal. *Pascal: Begi Concepts*—Program Structure: PROGI BEGIN, END; WRITELN and WRITE; sor Control: GOTOXY; Quiz. *Variable Inputting*—Variables; Variable T Calculations; Quiz-Variables; REA] READ-Input Without Pressing "Ret Quiz-Inputting; Other Variable T REALs, BOOLEANs, LONG INTE(Quiz-Other Variable Types. *Procedure First Time Around*—Building Bl Global and Local Variables; Proce Calling Procedures; Nested Procec Quiz-Procedures. *Program Control Loops*—The FOR Statement; Variatio FOR; Compound Statements; The Payment Program; Expanding a Pro;

(Continued on pag

Double Barrelled Disassembler

Here is a short utility to make creating disassembly listings easier. This program not only lists from starting to ending addresses, but also formats the listing into two columns for easier reading and less paper usage.

David L. Rosenberg
1706 Ridge Oak Place
Memphis,Tennessee 38119

How many L's are there between $BD00 and $BFFF? What seems at first to be a ridiculous question actually points out one of the few flaws in the Apple II's ROM Monitor. The problem arises because the disassembler routine only prints twenty lines at a time. This can be a major annoyance if you are doing a lot of long listings.

The program presented here attacks this problem and formats the listing into two columns to minimize wasted paper and make the disassembly easier to follow. Once the program has been BRUN the disassembly function is called by typing "beginning address"."ending address" (CTRL-Y) return. This sequence will disassemble the code from the beginning address through the ending address and print it in two column per page format (see listing 1).

How Does it Work?

This program works by dividing the first part of the object code into two segments, each containing the same number of instructions as there are lines on a page. Then taking one instruction from each piece, it calls the Monitor disassembly routine to print them on the same line. Next the pointers to the instructions are incremented and the program loops to the disassembly portion again. When all the instructions in each segment are done, a form-feed is printed and the next portion of the code is segmented, and the process is repeated until the ending address is reached.

```
LINE# LOC CODE   LINE

0002 0000
0003 0000        ; *******************************************************
0004 0000        ; *** THIS PROGRAM PRODUCES A TWO COLUMN DISASSEMBLY ***
0005 0000        ; *** LISTING USING PARTS OF THE MONITOR DISASSEMBLY ***
0006 0000        ; *** ROUTINE. IT PRINTS 60 LINES TO THE PAGE AND    ***
0007 0000        ; *** REQUIRES A 132 COLUMN PRINTER; HOWEVER THIS    ***
0008 0000        ; *** CAN BE MODIFIED IN THE PROGRAM.                ***
0009 0000        ; *** TO INVOKE THE DISASSEMBLER BRUN THE PROGRAM    ***
0010 0000        ; *** AND THEN FROM MONITOR TYPE:                    ***
0011 0000        ; *** BEGINNING ADDRESS.ENDING ADDRESS (CTRL-Y)      ***
0012 0000        ; *******************************************************
0013 0000
0014 0000        CH     = $24                 ; CURSOR HORIZONTAL POSN
0015 0000        LEN    = $2F                 ; INSTRUCTION LENGTH
0016 0000        PC     = $3A                 ; ADDRESS TO DISASSEMBLE
0017 0000        A2     = $3E                 ; ENDING ADDRESS
0018 0000        A3     = $40                 ; ADDRESS TO DISASSEMBLE
0019 0000        A4     = $42                 ; WORK BYTE
0020 0000        A5     = $45                 ; LINE COUNTER
0021 0000        VECTOR = $3F8              ; CTRL-Y VECTOR ADDRESS
0022 0000        NOVID  = $579              ; SERIAL CARD NO VIDEO FLAG
0023 0000        HOOKS  = $AA53             ; DOS 3.2.1 OUTPUT HOOK
0024 0000        INSDS2 = $F88E             ; ROUTINE FOR INSTRUCTION LENGTH
0025 0000        PRINT  = $FDED             ; MONITOR COUT ROUTINE
0026 0000        PR1    = $FD99             ; PART OF DISASSEMBLER (ROM)
0027 0000        PR2    = $F889             ; PART OF DISASSEMBLER (ROM)
0028 0000        PR3    = $F8D3             ; PART OF DISASSEMBLER (ROM)
0029 0000        PR4    = $FE67             ; PART OF DISASSEMBLER (ROM)
0030 0000               *      = $800
0031 0800        ;
0032 0800        ;***********************************************************
0033 0800        ;*** THIS ROUTINE SETS THE APPLE'S CTRL-Y VECTOR ADDRESS  **
0034 0800        ;*** TO POINT TO THE START OF THE DISASSEMBLER CODE       **
0035 0800        ;*** IT IS EXECUTED WHEN THE PROGRAM IS BRUN              **
0036 0800        ;***********************************************************
0037 0800        ;
0038 0800 A94C   INIT   LDA #$4C         ; OP CODE FOR JUMP
0039 0802 8DF803        STA VECTOR       ; STORE AT CTRL-Y VECTOR
0040 0805 A910          LDA #<START      ; GET LOW BYTE OF ENTRY LOCATION
0041 0807 8DF903        STA VECTOR+1     ; STORE AT VECTOR
0042 080A A908          LDA #>START      ; GET HI  BYTE OF ENTRY LOCATION
0043 080C 8DFA03        STA VECTOR+2     ; STORE AT VECTOR
0044 080F 60            RTS
0046 0810        ;
0047 0810        ;***********************************************************
0048 0810        ;***                 START OF DISASSEMBLER                 **
0049 0810        ;***********************************************************
0050 0810        ;
0051 0810 206208 START  JSR STHOOK       ; SET OUTPUT HOOKS FOR PRINTER
0052 0813 208708 MAIN   JSR SETPC        ; SET PC TO A3
0053 0816 209908        JSR SETA5        ; SET A5 TO # OF LINES PER PAGE
0054 0819 20E908        JSR INITA3       ; SET A3 TO START OF COLUMN 2
0055 081C 209E08 LOOP   JSR CMPCA2       ; COMPARE PC TO END ADDRESS
0056 081F 20D608        JSR DISASM       ; DISASSEMBLE INSTRUCTION AT PC
0057 0822 20B708        JSR CMA3A2       ; COMPARE A3 TO END ADDRESS
0058 0825 B012          BCS LOOP2        ; DON'T PRINT SECOND COLUMN IF >
0059 0827 20C408        JSR STORPC       ; SAVE PC AT A4
0060 082A 208708        JSR SETPC        ; SET PC TO A3
0061 082D 204808        JSR TAB          ; SKIP TO MIDDLE OF PAGE
0062 0830 20D608        JSR DISASM       ; DISASSEMBLE INSTRUCTION AT PC(=A3)
0063 0833 209008        JSR SETA3        ; SET A3 TO PC
0064 0836 20CD08        JSR RSTRPC       ; SET PC TO A4
0065 0839 A90D   LOOP2  LDA #$0D
0066 083B 20EDFD        JSR PRINT        ; PRINT CARRIAGE RETURN
0067 083E C645          DEC A5           ; DECREMENT LINE COUNTER
0068 0840 D0DA          BNE LOOP         ; IF NOT END OF PAGE THEN LOOP
0069 0842 205C08        JSR FFEED        ; ADVANCE TO NEXT PAGE
0070 0845 4C1308        JMP MAIN
0072 0848 A942   TAB    LDA #$42         ; SET X-REG TO
0073 084A 38            SEC              ; 66 - CURSOR POSITION
0074 084B E524          SBC CH           ; I.E. # OF SPACES TO PRINT
0075 084D AA            TAX              ; TILL MIDDLE OF PAGE
0076 084E F00B   T1     BEQ TX
0077 0850 3009          BMI TX
0078 0852 A9A0          LDA #$A0
0079 0854 20EDFD        JSR PRINT        ; PRINT SPACES TILL
0080 0857 CA            DEX              ; X-REG = 0
```

The only problem I encountered was that the Monitor disassembly routine prints a carriage return as the first character each time it is called. Obviously this is not desirable after we go to the trouble of positioning the printer to the start of the second column. To circumvent this the disassembler is called in four separate pieces.

PR1 is called to print the address in the Program Counter ($3A,$3B) as four ASCII bytes followed by a dash. PR2 gets the length of the instruction pointed at by PC, and forms an index into the Monitor's op-code mnemonic table. PR3 actually prints the mnemonic along with the appropriate address or hex literal. At this point we must push a $01 onto the stack to indicate that this is the last instruction to disassemble. PR4 increments PC to point to the next instruction then pulls the top value from the stack, decrements it by one and if equal to zero does a return. Since PR4 is jumped to, this return will take us back to the mainline where the program sets up to disassemble the corresponding instruction from column two.

Before calling the Monitor disassembler, PC must contain the address of the instruction to be disassembled. Since we are disassembling and printing two non-sequential instructions on each line, a large part of the program is concerned with swapping instruction addresses in and out of PC. A4 ($42,$43) is used as a work byte to store the column one address when the second column is being disassembled. A3 ($40,$41) serves a similar function when the first column is being disassembled. A2 ($3E,$3F) always contains the ending address of the code to be disassembled.

The subroutine INITA3 is interesting because it calls a Monitor routine at $F88E to return the length of an instruction. The whole purpose of the routine is to find the address of the $nth+1$ instruction, where n is the number of lines per page. This is also the start of column two, and so we want this address to wind up in A3. To accomplish this we will call INSDS2 n times and add the resulting length to the address at A3. Note that the length returned is actually one less than the actual instruction length, and therefore, we must increment LEN before adding it to A3. Invalid op-codes are not flagged, but are returned as one-byte length instructions.

```
0858 4C4E08          JMP T1
085B 60      TX      RTS
085C
085C A90C    FFEED   LDA #$0C           ; PRINT FORM FEED CHARACTER
085E 20EDFD          JSR PRINT
0861 60              RTS
0862
0862 A000    STHOOK  LDY #$00           ; SET THE DOS OUTPUT HOOK
0864 A2C1            LDX #$C1           ; TO $C100 SLOT 1
0866 8E54AA          STX HOOKS+1
0869 8C53AA          STY HOOKS
086C A98D            LDA #$8D           ; PRINT CARRIAGE RETURN TO
086E 20EDFD          JSR PRINT          ; INITIALIZE SERIAL CARD
0871 A980            LDA #$80           ; SET SERIAL CARD TO
0873 8D7905          STA NOVID          ; NO VIDEO MODE
0876 60              RTS
0877
0877 A900    UNHOOK  LDA #$00           ; RESET VIDEO MODE
0879 A0F0            LDY #$F0           ; AND RESTORE OUTPUT
087B A2FD            LDX #$FD           ; HOOKS TO SCREEN
087D 8D7905          STA NOVID
0880 8C53AA          STY HOOKS
0883 8E54AA          STX HOOKS+1
0886 60              RTS
0887
0887 A540    SETPC   LDA A3             ; SET PC TO A3
0889 853A            STA PC
088B A541            LDA A3+1
088D 853B            STA PC+1
088F 60              RTS
0890
0890 A53A    SETA3   LDA PC             ; SET A3 TO PC
0892 8540            STA A3
0894 A53B            LDA PC+1
0896 8541            STA A3+1
0898 60              RTS
0899
0899 A93C    SETA5   LDA #$3C           ; INITIALIZE LINE COUNTER TO
089B 8545            STA A5             ; 60 --- COUNTS DOWN
089D 60              RTS
089E
089E
089E A53B    CMPCA2  LDA PC+1           ; COMPARE HI BYTE OF PC TO
08A0 C53F            CMP A2+1           ; HI BYTE OF A2 (END ADDR)
08A2 9012            BCC C2             ; < RETURN
08A4 F005            BEQ C1             ; = COMPARE LOW BYTES
08A6 68              PLA                ; POP RETURN ADDRESS
08A7 68              PLA                ; OFF THE STACK
08A8 4C7708          JMP UNHOOK         ; RESET HOOKS AND QUIT
08AB A53A    C1      LDA PC             ; COMPARE LOW BYTES
08AD C53E            CMP A2
08AF 9005            BCC C2             ; < RETURN
08B1 68              PLA                ; POP STACK
08B2 68              PLA
08B3 4C7708          JMP UNHOOK         ; RESET AND QUIT
08B6 60      C2      RTS
08B7
08B7 A541    CMA3A2  LDA A3+1           ; COMPARE A3 AND A2
08B9 C53F            CMP A2+1           ; RETURN WITH CARRY BIT
08BB 9006            BCC CMA2           ; SET OR CLEAR TO
08BD D004            BNE CMA2           ; INDICATE STATUS
08BF A540            LDA A3
08C1 C53E            CMP A2
08C3 60      CMA2    RTS
08C4
08C4 A53A    STORPC  LDA PC             ; SAVE CURRENT VALUE OF PC
08C6 8542            STA A4             ; AT A4
08C8 A53B            LDA PC+1
08CA 8543            STA A4+1
08CC 60              RTS
08CD
08CD A542    RSTRPC  LDA A4             ; RESTORE PC FROM CURRENT
08CF 853A            STA PC             ; VALUE OF A4
08D1 A543            LDA A4+1
08D3 853B            STA PC+1
08D5 60              RTS
08D6
08D6 A63A    DISASM  LDX PC             ; DISASSEMBLE 1 INSTRUCTION
08D8 A43B            LDY PC+1           ; AT PC USING MONITOR
08DA 2099FD          JSR PR1            ; DISASSEMBLE ROUTINE IN
08DD 2089F8          JSR PR2            ; FOUR PARTS
08E0 20D3F8          JSR PR3
08E3 A901            LDA #$01           ; SET COUNTER ON STACK FOR
08E5 48              PHA                ; NUMBER OF INSTRUCTIONS
08E6 4C67FE          JMP PR4            ; ROUTINE SUPPLIES RTS
08E9
```

```
0169 08E9
0170 08E9              ;
0171 08E9              ; ***************************************************
0172 08E9              ; *** THIS ROUTINE CALCULATES THE ADDRESS OF THE ***
0173 08E9              ; *** FIRST INSTRUCTION IN COLUMN TWO            ***
0174 08E9              ; ***************************************************
0175 08E9              ;
0176 08E9
0177 08E9 A23C         INITA3 LDX #$3C            ; NUMBER OF INSTRUCTIONS
0178 08EB A000         INIT41 LDY #$00            ; SET INDEX POINTER
0179 08ED 8A                  TXA                 ; SAVE NUMBER OF
0180 08EE 48                  PHA                 ; INSTRUCTIONS ON STACK
0181 08EF B140                LDA (A3),Y          ; GET OP CODE
0182 08F1 208EF8              JSR INSDS2          ; MONITOR ROUTINE FOR LENGTH
0183 08F4 E62F                INC LEN
0184 08F6 A540                LDA A3              ; GET A3 AND
0185 08F8 18                  CLC                 ; INCREMENT BY
0186 08F9 652F                ADC LEN             ; LENGTH OF INSTRUCTION
0187 08FB 8540                STA A3              ; SAVE IN A3
0188 08FD 9002                BCC INIT42          ; INCREMENT HI BYTE
0189 08FF E641                INC A3+1            ; IF NECESSARY
0190 0901 68           INIT42 PLA                 ; GET NUMBER OF INSTRUCTIONS
0191 0902 AA                  TAX
0192 0903 CA                  DEX                 ; SUBTRACT 1
0193 0904 D0E5                BNE INIT41          ; LOOP IF NOT DONE
0194 0906 60                  RTS
0195 0907
0196 0907              ; ************************
0197 0907              ; ** DAVID L. ROSENBERG **
0198 0907              ; ** 1706 RIDGE OAK PL. **
0199 0907              ; ** MEMPHIS TN., 38138 **
0200 0907              ; ************************
0201 0907
0202 0907                     .END
```


In order to end execution, routine CMPCA2 compares the current value of PC to the value of A2 (the end address). If it is equal to, or greater than A2, we pop the last return address from the stack and jump to UNHOOK. This effectively disconnects from the mainline and resets the stack to the condition it was at when the disassembler was first invoked. Because the program is called from monitor, the RTS in UNHOOK will result in a return to monitor.

Making it Work

This program was written for use with an AIO serial card in slot #1 and a Texas Instruments 810 printer. The routine STHOOK sets the DOS output hooks and disables the serial card's video echo. If your interface is in a different slot, change the LDX instruction at line 89. It is of the format C*n*, where *n* is the slot number. For printers with a software-selectable line width this would be the best place to include the code for this function. The routine UNHOOK is always the last one executed, and so is where you should reset the line width.

The first instruction in the routine TAB controls how far over (in print positions) the second column will start. This can be changed to ½ of the line width that you are using (i.e. $28 for an 80-column line). The number of lines per page is set in two places, line 118 and line 177. It can be set to suit your needs, but just be sure it is the same in both places.

If your printer does not recognize $0C as a form-feed character or does not have a formfeed, the routine FFE ED will have to be changed. Its only function is to cause the printer to skip to the top of the next page.

Since the program uses standard Apple output routines it can be used, as is, with any printer card (serial or parallel) that does not require a software driver. If you use a print driver routine, change the JSRs at lines 66, 79, 85 and 93 to go to your driver entry point. The character to be printed will reside in the Accumulator prior to these calls.

David L. Rosenberg is presently employed as an analyst with the Management Sciences department of Holiday Inns, Inc., and has been in the computer field for eight years. He is a founding member of the Apple Core of Memphis and has contributed programs to its "diskette of the month." In addition to working on software and hardware projects for his Apple, which he has owned for a year and a half, he is actively pursuing a Masters degree in Computer Science.

MICRO

Single-drive Disk Back-Ups for Apple

This program allows the owner of a single Disk II drive to back up a disk without worrying about the types of files residing on it. While written for a 48K machine using DOS 3.2, little difficulty should be encountered in converting to DOS 3.3 or to a smaller size machine. Tracks containing DOS are not copied.

Steve Emmett
12816 Tewksbury Drive
Herndon, Virginia 22071

The idea for this single disk drive copy routine was born out of the frustration encountered, and time spent, in doing the many LOAD/SAVEs and BLOAD/BSAVEs necessary to back up disk files. Especially time consuming, and in some cases close to impossible, were the lengthy text files that I encountered on at least one purchased game disk.

The program to be described was the RWTS routine inherent in DOS 3.2 and well documented in *The Do's and Don't's of DOS 3.2.* RWTS permits the reading and/or writing of any specified track/sector combination on a disk. (For an excellent description of the disk format, see pages 123-137 of the DOS 3.2 manual.)

Since I have but one Disk II drive, the philosophy behind the program is to minimize the number of times it is necessary to remove and insert original/backup disks. Of the 35 tracks on a disk, the first 3 are devoted to the DOS 3.2 operating systems. I chose not to incorporate these 3 tracks in the duplication process. There is no program impediment, however, to their incorporation if desired. The remaining 32 tracks were divided into 4 groups, each containing 8 consecutive tracks. Table 1 lists the group number and the track numbers in both decimal and hex. Each track is composed of 13 sectors (numbered 0-12 or $0-$C) with each sector containing 256 bytes. Thus, one track contains 3328 ($CFF) bytes, and each group contains 26624 ($6800) bytes.

Since my Apple II is a 48K machine, there is no problem in temporarily storing the 26K of data from each group in RAM during disk backup. While I have not tried it, I see no reason why appropriate changes in the program cannot be made to allow a 32K machine to accomplish backup using 8 track groups. In addition, with the imminent release of DOS 3.3 and the attending change in sectors per track from 13 to 16, there is only a minimal change to the program that must be made to allow this program to work on 16 sectors per track.

Program Description

The program to accomplish the backup is written in both BASIC and machine language, with operator interface provided by BASIC. The core of the machine language program is the RWTS routine. To use the RWTS routine, two data blocks need to be defined: the Device Characteristics Table (DCT) and the Input/Output Block (IOB). As described in the DOS 3.2 manual, the DCT remains constant, while variables within the IOB are subject to change, depending upon whether a read or write operation is being undertaken. Since RWTS performs a single track/sector operation each time it is called, the rest of the machine language program is used to increment RAM buffer pointers, track and sector counters, and to switch between read and write.

The machine language program starts at $800, and to keep the calculation of RAM buffer ponters simple, it was decided to start the buffer at $1000. Since each sector of the disk contains 256 ($FF) bytes, it is necessary to increment only the high order byte of the buffer pointer. If the low order byte is not zero, the extra programming necessary to implement buffer pointer calculation is eliminated at the expense of the loss of a little flexibility.

***Table 1:* Track Grouping**

Group	Decimal	Hex
1	3-10	$3-$A
2	11-18	$B-$12
3	19-26	$13-$1A
4	27-34	$1B-$22

Prior to discussion of the machine language program, several definitions need to be made: Variable names for the IOB and DCT follow the same scheme as presented in the DOS 3.2 manual. DIO is the number of original disk inserts that will occur. For a 48K machine, it is 4. For a 32K machine it is 8. While it is possible to do the backup in less than 8 inserts on a 32K machine, the increased bookkeeping necessary to count tracks read is not considered worth the effort.

As an example DIO = 6 could be used, but then 5⅓ tracks must be read for each original insert. Or 5 occurrences of 6 tracks per insert need to be read, with a test to insure that the last insert reads only 2 tracks. Either option is possible, but I do not feel that the increased overhead in the software to account for these possibilities is necessary.

The variable TRK is the number of tracks that will be read for each original disk insert. For a 48K machine, it is 8. For a 32K machine, it is 4. SCT is the number of sectors per track that are to be read. Under DOS 3.2 it is 13. With DOS 3.3 it will be 16. As an aside, this is the only change to the program that must be made in order to run under DOS 3.3 (with the possible exception of the RWTS entry point). The increase in the number of bytes read as a result of SCT being 16 (with TRK still being 8 and DIO being 4) causes no data contention between the program located at the low side of the memory and the beginning of the DOS at the high side of memory.

CTRK is simply the number of the track currently being read or written. CSCT is the current sector, and CDIO is the current original disk insert count. NTRK is a local pointer that increments between 1 and 8, and is the current number of tracks processed for the current disk insert.

With these definitions in mind, analysis of the machine language program can begin. (Refer to the listing as needed.)

Locations 800 through 80C (all locations are presumed to be in hex notation as are all variables) are set aside for constant storage. 80D through 812 is set aside as temporary storage of variables. 813 through 823 is the IOB, and 824 through 827 is the DCT. 828 is reserved for the end of operation flag, and is initially set to zero.

Once the constants have been initialized, the RWTS routine is called. After each call, a check is made to determine if 13 sectors have been read. If they have not, CSCT is incremented. The starting address for the next 256 bytes to be delivered by RWTS is entered into the IOB and RWTS is called again. When 13 sectors have been read, a check is made to see if 8 tracks (NTRK) have been processed. If they have not, CTRK and NTRK are incremented, IOB is updated with the new buffer starting address and track/sector to be read, and RWTS is again called. This process continues until 8 tracks have been read. Once this happens, the program then checks to see if RWTS is in the read or write mode.

If it is in the write mode, a check is then made to see if the original disk has been inserted 4 times. If it has, the program branches to the END routine which resets all temporary storage and sets the end flag. A jump is then made back to the BASIC calling routine. If 4 original disk insets have not been made (and RWTS is in the write mode) then IOB is updated by switching to read mode, resetting the buffer to its default to handle the next set of 8 tracks (that the next sequential track has entered), and resetting the sector and track temporary counters. The program then jumps to the BASIC calling routine where operator instructions are given.

Assembly Listing

```
0800           ;*
0800           ;* DISK COPY ROUTINE
0800           ;*  BY STEVE EMMETT
0800           ;*
0800 04        DIO     BYT $04          ;CONSTANTS
0801 08        TRK     BYT $08
0802 0C        SCT     BYT $0C
0803 13        IOBLO   BYT $13
0804 08        IOBHI   BYT $08
0805 24        DCTLO   BYT $24
0806 08        DCTHI   BYT $08
0807 60        CSLOT   BYT $60
0808 01        CDRV    BYT $01
0809 60        PSLOT   BYT $60
080A 01        PDRV    BYT $01
080B 00        BUFLO   BYT $00
080C 10        BUFAB   BYT $10
080D           ;
080D 03        CTRK    BYT $03          ;TEMPORARY
080E 00        CSCT    BYT $00          ;STORAGE
080F 01        CDIO    BYT $01
0810 10        BUFHI   BYT $10
0811 01        NTRK    BYT $01
0812 01        RWS     BYT $01
0813           ;
0813 01        IBTYPE  BYT $01          ;IOB
0814 60        IBSLOT  BYT $60
0815 01        IBDRVN  BYT $01
0816 00        INVOL   BYT $00
0817 03        IBTRK   BYT $03
0818 00        IBSECT  BYT $00
0819 24        IBDCTL  BYT $24
081A 08        IBDCTH  BYT $08
081B 00        IBBUFL  BYT $00
081C 10        IBBUFH  BYT $10
081D 00                BYT $00
081E 00                BYT $00
081F 01        IBCMD   BYT $01
0820 00        IBSTAT  BYT $00
0821 00        IBSMOD  BYT $00
0822 60        IOBPSN  BYT $60
0823 01        IOBPDN  BYT $01
0824 00                BYT $00
0825 01                BYT $01          ;DCT
0826 EF                BYT $EF          ;DCT
0827 D8                BYT $D8          ;DCT
0828 00        FLAG    BYT $00          ;END FLAG
0829           ;
0829 A908      RCALL   LDA #$08
082B A013              LDY #$13
082D 20D903            JSR $03D9        ;RWTS CALL
0830 AD0E08            LDA CSCT
0833 CD0208            CMP SCT          ;13 SECTORS?
0836 F015              BEQ FSECT
0838 EE0E08            INC CSCT
083B EE1008            INC BUFHI
083E AD0E08            LDA CSCT
0841 8D1808            STA IBSECT
0844 AD1008            LDA BUFHI
0847 8D1C08            STA IBBUFH
084A 4C2908            JMP RCALL
084D           ;
084D AD0108    FSECT   LDA TRK
0850 CD1108            CMP NTRK         ;8 TRACKS?
0853 F023              BEQ FTRK
0855 EE1108            INC NTRK
0858 EE0D08            INC CTRK
085B A900              LDA #$00
085D 8D0E08            STA CSCT         ;ZERO SECTOR COUNT
0860 EE1008            INC BUFHI
0863 AD0E08            LDA CSCT
0866 8D1808            STA IBSECT
0869 AD0D08            LDA CTRK
```

```
(Continued)
086C 8D1708          STA IBTRK
086F AD1008          LDA BUFHI
0872 8D1C08          STA IBBUFH
0875 4C2908          JMP RCALL
0878             ;
0878 AD1208     FTRK LDA RWS
087B C901            CMP #$01        ;IN READ MODE?
087D F03C            BEQ RTW
087F AD0F08          LDA CDIO
0882 CD0008          CMP DIO         ;4 ORIGINAL INSERTS?
0885 F069            BEQ END
0887 EE0F08          INC CDIO
088A EE0D08          INC CTRK
088D A900            LDA #$00
088F 8D0E08          STA CSCT        ;ZERO SECTOR COUNT
0892 A901            LDA #$01
0894 8D1108          STA NTRK        ;RESET RELATIVE TRACK COUNT
0897 AD0C08          LDA BUFAB
089A 8D1008          STA BUFHI
089D CE1208          DEC RWS         ;RWS TO READ
08A0 AD0D08          LDA CTRK
08A3 8D1708          STA IBTRK
08A6 AD0E08          LDA CSCT
08A9 8D1808          STA IBSECT
08AC AD1008          LDA BUFHI
08AF 8D1C08          STA IBBUFH
08B2 AD1208          LDA RWS
08B5 8D1F08          STA IBCMD
08B8 4CEF08          JMP RTN
08BB             ;
08BB A901       RTW  LDA #$01
08BD 8D1108          STA NTRK
08C0 A900            LDA #$00
08C2 8D0E08          STA CSCT
08C5 AD0D08          LDA CTRK
08C8 38              SEC
08C9 E908            SBC #$08        ;CTRK=CTRK-8
08CB 8D0D08          STA CTRK
08CE AD0C08          LDA BUFAB
08D1 8D1008          STA BUFHI
```

BASIC Listing

```
10  CALL  - 936
20  CALL 2048
30  PRINT : PRINT : PRINT "   **SINGLE DRIVE DISC COPY** "
40  PRINT : PRINT
50  PRINT : PRINT "THIS PROGRAM WILL COPY TRACKS 3-34."
60  PRINT "DOS TRACKS (0-2) ARE NOT COPIED."
70  PRINT : PRINT
80  INPUT "ENTER THE ORIGINAL DISC AND HIT RETURN";R$
90  CALL 2089
100  IF  PEEK (2088) = 15 THEN  GOTO 140
110  IF  PEEK (2066) = 1 THEN  GOTO 80
120  INPUT "ENTER THE BACKUP DISC AND HIT RETURN";R$
130  GOTO 90
140  POKE 2088,0
150  PRINT : PRINT "BACKUP COMPLETED"
160  END
```

EXEC File Listing

```
10 D$ = "     ": REM D$=CTRLD
20  PRINT D$:"OPEN DISC COPY"
30  PRINT DS;"WRITE DISC COPY"
40  PRINT "INT"
50  PRINT "BLOAD BDISCCOPY"
60  PRINT "LOMEM:2500"
70  PRINT "RUN INTDISCOPY"
80  PRINT D$;"CLOSE DISC COPY"
90  END
```

If, on the other hand, RWTS is in the read mode, the program then decrements the value of CTRK by 8, and resets IOB by switching from read to write, entering the new value for CTRK and resetting the buffer address to its default value. The process ensures that the 8 tracks just read from the original disk can now be written onto the back-up disk. The program then exits to the BASIC routine.

This entire process continues until four original/backup disk insertions have been made. Once the program senses that it is in the write mode and that CDIO = 4, it then branches to the END routine. This routine then exits to the BASIC program declaring that the backup is complete. To back up another disk, all that is necessary is to type RUN.

To facilitate the use of these two routines, the EXEC function of DOS is used. EXEC allows the generation of a text file that is then processed as a series of DOS commands. In order to run the disk copy routines, enter the machine language program and BSAVE BDISCCOPY, A$800, L11F. Enter the BASIC program and

SAVE INTDISCOPY.

Then generate a text file to be EXEC'ed (see listing). Note that the entry on line 40 depends upon whether your system has the language card. If it does not, remove this entry and prior to performing the disk copy, make certain that your system is in Integer BASIC. To perform the disk backup procedure, simply

EXEC DISK COPY

and follow the instructions!

Steve Emmett is a physicist with 15 years in the computer field. Major interests are system security, simulation design and CAI for very young children. He has an Apple II with language card, one drive, and is presently designing a symbolic assembler/linker/loader.

MICRO

Software for the Apple II and Apple II Plus

BENEATH APPLE DOS

A Technical Manual

By Don Worth and Pieter Lechner

Become an expert on the intricacies of Apple's DOS (Disk Operating System). BENEATH APPLE DOS is the perfect companion to Apple's DOS 3.3 Manual. Containing eight chapters, three appendices, a glossary, an index, and over 160 pages, this manual will serve to completely fill in the many gaps left by Apple's DOS 3.3 Manual. Written for Apple users with DOS 3.3, 3.2 or earlier versions, any Apple disk user would welcome having this carefully written manual at his fingertips.

LEARN . . .

- How DOS 3.3 differs from other DOS versions.
- How disks are protected.
- How to reconstruct a damaged diskette CATALOG.
- How tracks are formatted.
- How to use the disk directly, *without* DOS.
- How to call DOS's file manager.
- How *every* routine in DOS works.
- How to customize DOS to your needs.
- How to overcome DISK I/O ERRORS.
- About the "secret" file types — S and R.

INCLUDES . . .

- Large quantities of excellent diagrams and tables.
- Source listings of useful disk utilities.
- Glossary of over 150 technical terms.
- Exhaustive description of DOS program logic.
- Handy reference card.
- Useful patches to DOS.
- Many programming examples.

Book - $19.95

CROSS-REF by Jim Aalto

Applesoft programmers will be delighted to have this cross reference utility program in their 'tool kit' of software aids. What can CROSS-REF do to speed and facilitate your Applesoft program development? Consider these functions:

VARIABLE CROSS REFERENCE	LINE CROSS REFERENCE
FIND VARIABLE	FIND LINE NUMBER
REPLACE VARIABLE	VARIABLE ONLY LISTING

LINE ONLY LISTING

Features that make CROSS-REF easy to use include:

- Written in machine language, occupies less than 3K.
- Resides passively in memory while DOS or Applesoft is active.
- Can be loaded with your Applesoft program already resident.
- Very fast — a VARIABLE CROSS REFERENCE for a 16K Applesoft program can start printing in 5 seconds.
- Contains printer format controls and headers for documentation.
- Prints English language error messages.

Cassette - $22.95 Diskette - $24.95

L I N K E R

LINKER by Don Worth.

Turn your Apple II or Apple II Plus into a powerful and productive software development machine with this superb linking loader/editor package. LINKER does the following and much more:

- Dynamically loads and relocates suitably prepared machine language programs anywhere in RAM.
- Combines a main program with subroutines. You can assemble a subroutine once and then use it with as many main programs as you wish.
- Produces a map of all loaded routines, giving their location and the total length of the resulting module.
- Contains a library of subroutines including binary multiplication and division, print text strings, delay, tone generator, and random number generator.

Linker works with virtually any assembler for the Apple II. Requires 32K of RAM and one disk drive.

Diskette - $49.95
Manual Only - $19.95

FASTGAMMON™ By Bob Christiansen.

Sound, hi res, color, and musical cartoons have helped make this the most popular backgammon playing game for the Apple II. But don't let these entertaining features fool you — FASTGAMMON plays serious backgammon. Runs on any Apple II with at least 24K of RAM.

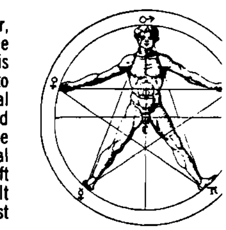
Cassette - $19.95 Diskette - $24.95

METEOROIDS IN SPACE™
By Bruce Wallace

We have taken our popular space game, formerly called Asteroids in Space, and made some important improvements. To accent these improvements we have given it a new name — METEOROIDS IN SPACE. Your space ship travels through a shower of deadly meteoroids. If your ship is hit, it will be destroyed, so you use your laser gun to blast the meteoroids. Big meteoroids shatter into smaller meteoroids when hit, and the smaller ones are usually faster and just as deadly. From time to time you will encounter an alien space ship whose mission is to destroy you, so you'd better destroy it first. All the action is displayed in fast, smooth, high resolution graphics, accompanied by sound effects. You now can control your ship usin of two options — the Apple game paddles or the keyboard. One of the game p buttons controls the laser fire. In METEOROIDS IN SPACE, the spaceship's ve gradually decreases unless more thrust is applied, adding an element of control new to this version is a hyperspace feature — translate instantly to another spot galaxy. The game is over when five of your ships have been destroyed. An addi ship is added for every 10,000 points you score. Runs on any Apple II with at leas of RAM and one disk drive.

Diskette - $19.95

ASTROAPPLE™ by Bob Male.



Your Apple computer becomes your astrologer, generating horoscopes and forecasts based on the computed positions of the heavenly bodies. This program offers a delightful and stimulating way to entertain friends. ASTROAPPLE produces natal horoscopes (birth charts) for each person based on his or her birth data. Any two people may be compared for physical, emotional, and intellectual compatibility. The program is written in Applesoft BASIC with machine language subroutines. It requires either RAM or ROM Applesoft and at least 32K of memory.

Cassette - $14.95 Diskette - $19.95

FRACAS™ by Stuart Smith.

A fantastic adventure game like no other! Up to players can participate in FRACAS at the same Journey in the land of FAROPH, searching for hi treasure while warding off all sorts of unfriendly dangerous creatures. You and your friends can cor with each other or you can join forces and gang up o monsters. Your location is presented graphically and s effects enliven the battles. Save your adventure on dis or cassette and continue it at some other time. Both in BASIC and Applesoft versions included. Requires at 32K of RAM.

Cassette - $19.95 Diskette - $24.95

BATTLESHIP COMMANDER™ by Erik Kilk and Matthew Jew.

A game of strategy. You and the computer each start out by positioning five shi different sizes on a ten by ten grid. Then the shooting starts. Place your volleys skil — a combination of logic and luck are required to beat the computer. Cartoons the ships sinking and announce the winner. Sound effects and flashing lights alsc to the enjoyment of the game. Both Applesoft and integer BASIC versions are inclu Requires at least 32K of RAM.

Cassette - $14.95 Diskette - $19.95

Also by Don Worth . . .

BENEATH APPLE MANOR — Adventure. Uses Integer BASIC.

Cassette - $14.95 Diskette - $19.95

BABBLE - Fun with words, sound, and graphics.

Cassette - $19.95 Diskette - $24.95

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

Now exclusive distributors for products from The Software Factory, Newhall, California

*Apple II and Apple II Plus are trademarks of Apple Computer, Inc.

WHERE TO GET IT: Call us at (213) 344-6599 for the name of the Quality Soft dealer nearest you. If necessary you may order directly from us. Mastercarc Visa cardholders may place orders by telephone. Or mail your check or bank number to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91 California residents add 6% sales tax. SHIPPING CHARGES: Within North Am orders must include $1.50 for first class shipping and handling. Outside N America the charge for airmail shipping and handling is $5.00. Pay in currency.

Enhanced Input Routine

Getting data into a program is one of the most important aspects of program development. This routine for the Apple does it all.

Bruce A. Robertson
1 Vanhurst Place
Ottawa, Ontario
Canada, K1V 9Z7

In professionally-written software, great care is taken to provide the program user with as much flexability as possible, as well as making the program easy to maintain. By having all input controlled by a single routine, many lines of code may be eliminated, input can be standardized, program control is more modular and in most cases the user of an interactive system is presented with a cleaner, more readable display.

The input routine shown in listing 1 is an adaptation (in Applesoft) of an input routine that will accomplish all of the above. Although it appears large at first glance, once the remarks are removed, it is actually quite small and very manageable. The large number of remarks were included to make the routine easy to understand.

This routine uses standard BASIC terms and could be keyed into any system using a variety of versions of BASIC.

From the User's Viewpoint

Anyone using a program containing this input routine has considerable power over program execution. For example, programs may be run to obtain intermediate results. The user can then back up and re-insert new data based on the results previously obtained. In a program that has a repetitive sequence where many of the prompts are repeated, only one actual input, containing the responses to all the questions, needs to be made.

To accomplish this, two characters are reserved for use by the input routine. The slash, "/", is used as a delimiter to separate multiple answers to a prompt. The question mark "?", when it is the first character, is used as a signal to back up to the previous prompt. A carriage return is interpreted as acceptance of the prompt default.

To illustrate, consider the following prompt sequence:

```
WHAT IS YOUR NAME
  (END PROGRAM)?
WHAT IS YOUR AGE (25)?
WHAT IS YOUR PHONE
  NUMBER (NONE)?
```

These prompts could be entered one at a time, or using the power of the input routine as:

```
WHAT IS YOUR NAME (END
  PROGRAM)? JOHN SMITH/
  22/555-4652
```

The program would then continue and print out the rest of the display as:

```
WHAT IS YOUR AGE (25)?22
WHAT IS YOUR PHONE
  NUMBER (NONE)?555-4652
```

If a list of names, ages and telephone numbers are being entered, a great deal of time could be saved by making only one entry. If the entries are being made one at a time, a mistake in the name, that is not discovered until the age is about to be entered, may be corrected by typing a "?" in response to the age prompt:

```
WHAT IS YOUR AGE (25)??
```

Whereupon the program would back up on the screen as well as in the program logic to the prompt:

```
WHAT IS YOUR NAME
  (END PROGRAM)? JOHN
  SMITH
```

with the cursor positioned on the "J" in "JOHN". The correct response is now typed in and the program is continued.

The user has one other command that is recognized by the input routine — the word "QUIT." If the word "QUIT" is entered as the sole response to any prompt, then program execution is immediately transferred to whatever closing routine is provided by the program, and an orderly exit is completed. To the user this could mean a quick chaining back to a controlling program or menu.

The input routine also allows the sensing of default inputs and provides an easy method for the user to enter often-used responses. As can be seen from the prompts above, a default answer is provided for each of the questions. These defaults are chosen to provide the most-used or least-harmful responses to each input request. This allows the user to progress through the program by simply pressing the carriage return for most inputs.

How It Works

Although there are many remarks in the listing to explain the operation of the routine, the following line-by-line explanation will clear any doubts and will attempt to highlight the reasoning behind the code. Line 905 — BACKUP is the variable used by the mainline of the program to indicate whether or not it is necessary to back up through the program. DISPLAY is used by the input routine to decide if it is necessary to print the present response on the terminal. If a multiple entry response is given, the second and later portion of the response must be printed when the appropriate prompt is printed. However, since they will not be keyed in from the keyboard and echoed on the screen they must be printed by the program. DISPLAY gives the signal to the routine to print the response. ALPHA, NUMERIC and DFAULT are flags used to determine if the current response is alphabetic, numeric, or acceptance of the default.

The next command in line 905 determines if the actual INPUT command should be skipped by testing to see if anything is left over from previous input. If there is something left over, it equates the input variable, ANSWER$, to everything that is left over. Provided the IF condition test is true, the GOTO statement is executed and the INPUT command is skipped.

Line 910 accepts the program input into ANSWER$ and resets the DISPLAY variable to indicate that it is not necessary to print the response on the screen. Line 915 takes care of problems caused by successive default entries by placing a null character at the start of the input string. At line 920 the length of the input is found, and then the first character of the input is picked off and tested to see if it is the back-up signal character. The character tested for is the question mark "?". This was chosen because it is on the same key as the other special character that is used by the input routine, and because it is very unlikely that it would be the first character in any input string. It is coded as a CHR$(63) rather than as "?" only to show that any character may be used, including control characters.

If the back up signal is detected, the input statement and any pending responses are zeroed to eliminate possible errors when the input routine is next entered. Since under this condition, no further processing is required, an immediate RETURN to the mainline of the program is executed. Line 925 checks to ascertain if the current response is a multiple entry input. To do this, an in-string search is done for the input delimiter, the slash — "/". The slash is an arbitrary choice and could be any character desired, except the colon and the comma, which are used by the Apple monitor. The search is carried out for the full length of the response.

The search is conducted in a loop and only the first delimiter is of interest. If the character being examined is not a delimiter, it is of no interest and the next character is taken. Successive GOSUBs to the input routine will search for successive delimiters in any multiple entry input.

At line 930, if a delimiter has been found, the input string is split into the portion ahead of the delimiter, and everything afterwards. The left part contains the current answer and the right part is the remainder of the response. It is only necessary to find one answer at a time, so a GOTO is executed to exit from the search. At line 940, if a delimiter has not been found, the program completes the loop and transfers the entire input to the routine output string. The string holding anything left over is zeroed because the last response of a multiple entry input would fall through to line 940, and OVER$ would still contain this last response on the next entry to the input routine.

Line 945 causes the current ans of a multiple entry input to be printe response to a prompt, as if an INP command had actually been execu This is necessary because line 91(skipped on subsequent entries to the put routine if more than one answe detected. It is important to note that variable DISPLAY need not be equa to anything. Applesoft, in a conditic

Listing 1

```
900     REM *** INPUT ROUTINE ***

905     BACKUP = 0:DFAULT = 0:ALPHA = 0:NUMERIC = 0
        :IF OVER$ <> '' THEN ANSWER$ = OVER$
        :DISPLAY = 1
        :GOTO 915
        :REM    IS ANYTHING LEFT OVER FROM PREVIOUS INPUT?
                SKIP INPUT IF ANYTHING LEFT OVER

910     INPUT ANSWER$ : DISPLAY = 0
        :ANSWER$ = ANSWER$ + CHR$(0)
        :REM    GET INPUT AND TURN OFF DISPLAY FLAG.

915     IF LEFT$(ANSWER$,1) = '/' THEN ANSWER$ = CHR$(0) + ANSWER
        :REM    ADD NULLS TO HANDLE PROBLEMS CREATED BY
                SLASH BEING FIRST OR LAST CHARACTER

920     LGTH = LEN(ANSWER$)
        :IF LEFT$(ANSWER$,1) = CHR$(63) THEN BACKUP = 1
        :ANSWER$ = '' : OVER$ = ''
        :RETURN
        :REM    FIND LENGTH OF INPUT
                CHECK IF BACKUP CHARACTER ENTERED
                CHR$(63) IS A QUESTION MARK
                ZERO INPUT STRINGS

925     FOR I = 1 TO LGTH
        :IF MID$(ANSWER$,I,1) <> '/' THEN GOTO 935
        :REM    HOW MANY CHARACTERS TO CHECK
                SEARCH FOR INPUT DELIMITER

930     REPLY$ = LEFT$(ANSWER$,I-1)
        :OVER$ = RIGHT$(ANSWER$,LGTH-I)
        :GOTO 945
        :REM    PICK OFF FIRST ANSWER IN STRING
                SAVE REST OF INPUT STRING
                STOP LOOKING FOR DELIMITER

935     NEXT I
        :REM    FALLS THROUGH IF NO DELIMITER FOUND

940     REPLY$ = ANSWER$
        :OVER$ = ''
        :REM    TRANSFER INPUT TO ROUTINE OUTPUT STRING
                INSURE NOTHING LEFT OVER

945     IF DISPLAY  THEN PRINT '? ';REPLY$
        :REM    IF MULTIPLE INPUTS THEN PRINT
                PRESENT INPUT ON SCREEN

950     IF REPLY$ = 'QUIT' + CHR$(0) THEN GOTO 32000
        :REM    PROVIDE QUICK EXIT FROM PROGRAM
                LINE 32000 IS START OF CLOSING SEQUENCE

955     SMALL$ = LEFT$(REPLY$,1)
        :IF SMALL$ = CHR$(0) THEN DFAULT = 1
        :REM    PICK OFF FIRST LETTER OF INPUT
                IF NULL STRING THEN INPUT IS DEFAULT

960     IF ASC(SMALL$) > 64 AND ASC(SMALL$) < 91 THEN ALPHA = 1
        :REM    CHECK IF FIRST CHARACTER ALPHABETIC
                FOR MINI - EDIT

965     IF ASC(SMALL$) > 47 AND ASC(SMALL$) < 58 THEN NUMERIC = 1
        :REM    CHECK IF FIRST CHARACTER NUMERIC
                FOR MINI - EDIT

970     RETURN
```

Listing 2

```
          *** EXAMPLE OF USAGE ***

90        HOME :REM CLEAR SCREEN

100       VTAB(10):PRINT "WHAT IS YOUR NAME <END PROGRAM> ";
          :GOSUB 900
          :REM    ESTABLISH SCREEN POSITION
                  PRINT PROMPT AND GET INPUT

110       IF BACKUP OR DFAULT  THEN GOTO 32000
          :REM    32000 IS START OF CLOSING SEQUENCE
                  THIS BACKS OUT THE TOP OF THE PROGRAM

120       IF ALPHA = 0 THEN OVER$ = ""
          :GOTO 100
          :REM    MINI EDIT - ENSURE INPUT ALPHABETIC

130       NAME$ = REPLY$
          :REM     SAVE INPUT


140       VTAB(12):PRINT "WHAT IS YOUR AGE <25> ";
          :GOSUB 900
          :REM    ESTABLISH SCREEN POSITION
                  PRINT PROMPT AND GET INPUT

150       IF BACKUP  THEN GOTO 100
          :REM    BACK UP

160       IF DFAULT THEN AGE = 25
          :GOTO 190
          :REM    DEFAULT VALUE

170       IF NOT NUMERIC THEN OVER$ = ""
          :GOTO 140
          :REM    MINI EDIT - CHECK IF INPUT NUMERIC

180       AGE = VAL(REPLY$)
          :REM    SAVE AGE

190                       .... ETC.





32000     VTAB(22):PRINT "ARE YOU FINISHED <NO>";
          :GOSUB 900
32010     IF BACKUP OR DFAULT GOTO 90
32020     IF NOT ALPHA GOTO 32000
32030     IF SMALL$ <> "Y" GOTO 32000

32100     HOME:VTAB(12):HTAB(12)
          :PRINT "THANK YOU AND GOODBYE"
```

test, need only determine if the condition is true. In the absence of an equal sign the test is true if the variable has any value other than zero.

At line 950 the word "QUIT" provides a shortcut through the program to the closing sequence, which is very useful when testing or maintaining a program. It can also be used, if the closing sequence is properly coded, to loop to the start of the program, rather than going through a long series of prompts or exiting from the program run.

At line 955, SMALL$ provides a one-character output from the routine that is most useful when "yes" or "no" responses are possible, or when a single character is sufficient to distinguish between a series of inputs. This string is tested to determine whether the current answer is a default response. At lines 960,965 a miniscule edit is performed to determine if the first character in the current answer is alphabetic or numeric. The appropriate flag is set for use by the program mainline. Any small edit can be carried out in the input routine with the edit either hard-coded as shown or passed to the routine as a variable. However, editing of a more substantial nature should be placed in a separate routine.

How to Use It

Listing 2 shows the type of coding necessary to effectively use the input routine. Each mainline input should request only one input, provide a default, test for the backup flag and the default flag, and save any input or default in the appropriate variable. The input routine provides six outputs: REPLY$, SMALL$, BACKUP, ALPHA, NUMERIC and DFAULT. Use of these outputs in an effective manner will provide positive program control and will benefit both user and programmer. Screen addressing should be used for all prompts to allow for over-printing of prompts when backing up so as not to clutter up the display with repeated prompts.

The Apple computer does not allow use of the ELSE statement, so each test of a flag must be on a separate, numbered line. On systems where the ELSE statement is allowed, all flag tests can be in case structure on the same numbered line as the prompt.

Summary

The INPUT ROUTINE is an extremely useful addition to any subroutine library and its use will certainly improve program control in new program development. It is self-contained and can be plugged into existing code with a minimum of effort. Programs using this routine will increase their through-put and improve user acceptance. A bit of practice will soon show you the power and limitations that can be expected using this routine. Good programming!

Bruce A. Robertson is an electronic specialist with over 20 years experience. He has been programming since 1977 and is currently employed by the Department of National Defense in the Directorate of Computer Applications Development as an applications programmer. He has owned an Apple II Plus computer for over a year.

MICRO

Binary File Parameter List

This utility program will list the address and length, in both hex and decimal, of all binary files on a given disk. It will also calculate the number of free sectors available on the disk. The utility works equally well with both DOS 3.2 and DOS 3.3.

Clyde R. Camp
3518 Wildflower Lane
Johnson City, Tennessee 37601

Although Apple DOS 3.2 is a relatively powerful Disk Operating System, it is geared primarily towards BASIC file management, and is somewhat short on capabilities for machine language or binary file management. Among other things, it is left to the user to remember the address and length of a binary file (BFILE) when using BSAVE and BLOAD.

This can be very aggravating when it becomes necessary to copy or relocate a BFILE, or to know where it was originally located, or what its length is. Although one can always BLOAD the file and then PEEK into page zero RAM to find the start and length parameters, this must be done manually, in immediate mode. Any program to do the PEEKing could be inadvertently overwritten by the BLOAD operation, and to blindly BLOAD one of these files could wipe out existing programs, alter data bases, or even zap DOS itself. Even though an arbitrary starting address can be specified, an unknown length is still likely to cause trouble by overwriting needed portions of RAM.

The program listed here (listing 1) avoids all of these problems by utilizing the DOS RWTS subroutine to search for and list all BFILE parameters (name, address, length) on a given disk. (In addition, it calculates the total number of remaining free sectors on the disk, which is a very useful piece of information.) It accomplishes this by searching the disk directory for binary files. Once a BFILE is located, the first four bytes of the first sector of the file are examined. These bytes contain the start and length parameters as follows:

Byte 0 Least significant byte of address
Byte 1 Most significant byte of address
Byte 2 Least significant byte of length
Byte 3 Most significant byte of length

Since at most, only one sector of the BFILE need be loaded (the first sector), only a known amount of buffer storage is needed (256 bytes to be exact) and the hazard of overwrite is prevented.

The program was written for an Apple II with 48K and Applesoft firmware, but it should run on any DOS system in which the user can utilize page one Hi-Res graphics. This is because the machine language routines involved reside in that memory area. Please note that most GOSUB and GOTO statements refer to REMs for documentation purposes. So, when entering the program, be sure to include at least these REM statements to prevent a lot of MISSING STATEMENT error messages.

Listing 1

```
10  REM ***BFILE PARAMETER LIST***
20  GOSUB 960
30  TEXT : HOME : PRINT "THIS PROGRAM WILL SEARCH A GIVEN DISC    FOR ALL
      BINARY FILES, GIVING THE FIRST  13 LETTERS OF THE FILE NAME FOLLOWE
     D BY THE FILE START ADDRESS AND FILE LENGTH  IN BOTH HEX AND (DECIMAL)"
40  PRINT : PRINT "THE NUMBER OF FREE SECTORS ON THE DISC   WILL ALSO BE
     CALCULATED"
50  INVERSE : FLASH : VTAB 12: PRINT "     INSERT DISC TO BE SEARCHED       "
60  PRINT : PRINT "     DEPRESS RETURN TO CONTINUE,              ANY OTHER
      KEY TO EXIT PROGRAM      ";: NORMAL : GET A$: PRINT A$: IF A$ < > CHR$
     (13) THEN  TEXT : HOME : END
70 VT = 17:VS = 0:BASE = 9216:NULL$ = "":TC = 2 ^ 15 - 1
80  TEXT : HOME : INVERSE
90  PRINT " FILE NAME     START          LENGTH    "
95  PRINT "1ST 13 CHAR.  HEX(DEC.)      HEX(DEC.)  ": POKE 34,3: VTAB 6: HOME
100  NORMAL
170 TN = VT:SN = VS: GOSUB 880:CT =  PEEK (BASE + 1):CS =  PEEK (BASE + 2)
180 LC =  - 1: POKE 35,21
190  GOSUB 1350: GOSUB 1530
230 TN = CT:SN = CS: GOSUB 880
280 NTC =  PEEK (BASE + 1):NSC =  PEEK (BASE + 2)
340  FOR B2 = 11 TO 224 STEP 35
350 B3 = BASE + B2
360  IF  PEEK (B3) = 0 AND  PEEK (B3 + 1) = 0 THEN  POKE 35,23: VTAB 21: CALL
      - 958: PRINT : GOTO 500
370  IF  PEEK (B3) = 255 THEN 390
380 PR =  PEEK (B3 + 2): IF PR = 4 OR PR = 128 + 4 THEN  GOSUB 540
390  IF LC < 16 THEN 440
400  VTAB 24: PRINT "CONTINUE (Y-N) ? ";: GET A$: PRINT A$;: IF A$ = "Y" THEN
     420
401  HTAB 1: VTAB 24: PRINT "                              ";
402  POKE 35,23
405  VTAB 21: CALL  - 958
407  PRINT : VTAB 21: PRINT
410  GOTO 510
420  HTAB 1: VTAB 24: PRINT "                              ";
430 LC =  - 1: VTAB 21
435  PRINT
440  NEXT B2
450 CT = NTC:CS = NSC
460  GOTO 230
480  REM  EXIT PROGRAM
500  VTAB 21: PRINT "                       NO MORE BINARY FILES"
510  PRINT "                     FREE SECTORS= ";CNT: TEXT : VTAB 22: END
```

(Continued)

***Table 1:* Input/Output Control Block (IOB)**

Hex Address	Hex Data	Function
2300	01	IOB type indicator
2301	60	Slot number × 16
2302	01	Disk drive number
2303	00	Expected volume number
2304	11	Initial track number
2305	00	Initial sector number
2306	11 23	DCT address
2308	00 24	Buffer address
230A	00 00	Not used
230C	01	Command code (1 = READ 2 = WRITE)
230D	00	Error code
230E	FE	Actual volume number
230F	60	Previous slot × 16
2310	01	Previous drive

***Table 2:* Device Characteristic Table (DCT)**

Hex Address	Hex Data	Function
2311	00	Device type code
2312	01	Phases per track
2313	EF	Time count
2314	D8	Time count

Listing 1 *(Continued)*

```
520  END
540  REM  DISPLAY FILE PARAMETERS
620  FOR I = 3 TO 15
630  PRINT  CHR$ ( PEEK (B3 + I));: NEXT I
680 TN =  PEEK (B3):SN =  PEEK (B3 + 1): GOSUB 880
730 TN =  PEEK (BASE + 12):SN =  PEEK (BASE + 13): GOSUB 880
780 A =  PEEK (BASE) +  PEEK (1 + BASE) * 256:AA = A: IF AA > TC THEN AA =
     AA - 2 ^ 16
790 L =  PEEK (BASE + 2) + 256 *  PEEK (BASE + 3):LL = L: IF LL > TC THEN
     LL = LL - 2 ^ 16
800  HTAB 15:Z =  USR (AA): PRINT "("A")";
810  HTAB 28:Z =  USR (LL): PRINT "("L")"
820 LC = LC + 1
860 TN = CT:SN = CS: GOSUB 880: RETURN
880  REM  READ TRACK/SECTOR
940  POKE TA,TN: POKE SA,SN: POKE RD,1: CALL RWDRV: RETURN
960  REM  SETUPRWTS DRIVER
1090  HIMEM: 8191
1130  DATA  169,035,160,00,32,217,3,96,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     0,0: REM  19 ZEROES
1140  DATA 1,96,1,0,17,0,17,35,0,36,0,1,1,0,254,96,1,0,1,239,216
1150  FOR I = 8448 TO 8474: READ J: POKE I,J:: NEXT
1160  FOR I = 8960 TO 8980: READ J: POKE I,J: NEXT
1290 RWADV = 8448:TA = 8964:SA = 8965:RD = 8972
1300 SL = 6:DR = 1
1310 DA = 37148
1320  POKE DA,SL * 16: POKE DA + 14,SL * 16: POKE DA + 1,DR: POKE DA + 15 ,DR
1330  RETURN
1350  REM  DETERMINE FREE SPACE
1390  DATA 76,0,032,32,12,225,165,160,160,0,162,9,24,42,16,1,200,202,208,
     249,165,161,162,9,24,42,16,1,200,202,208,249,169,0,32,242,226,96,96
1400  FOR I = 10 TO 12: READ Z: POKE I,Z: NEXT
1410  FOR I = 8192 TO 8227: READ Z: POKE I,Z: NEXT
1490 CNT = 0
1500  FOR I = 56 TO 195 STEP 4:V =  PEEK (BASE + I) * 256 +  PEEK (BASE +
     I + 1):V =  INT (V / 2)
1510 CNT = CNT +  USR (V): NEXT : RETURN
1530  REM  SETUP DEC-HEX CONV.
1560  DATA 76,0,032,32,12,225,165,160,166,161,32,65,249,96
1570  FOR I = 10 TO 12: READ Z: POKE I,Z: NEXT I
1580  FOR I = 8192 TO 8202: READ Z: POKE I,Z: NEXT I
1590  RETURN
```

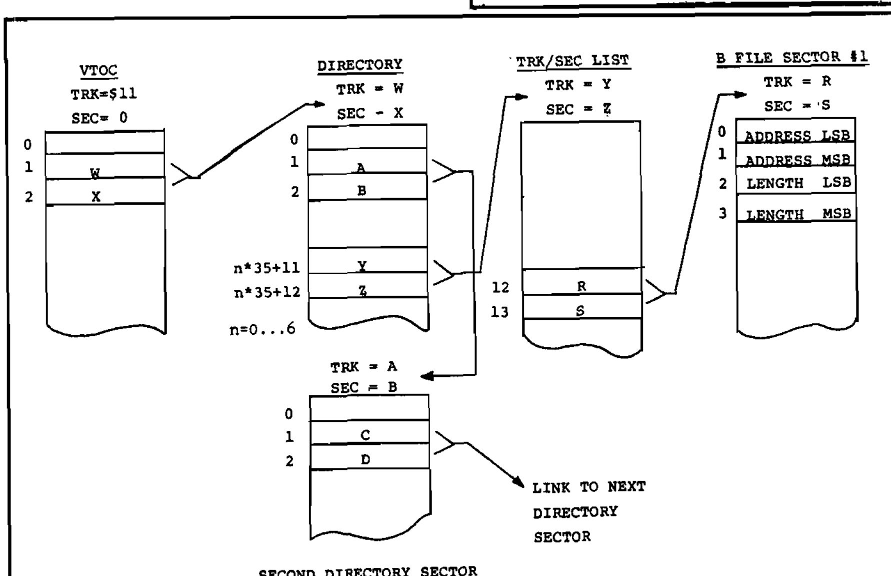


***Figure 1:* What Goes Where**

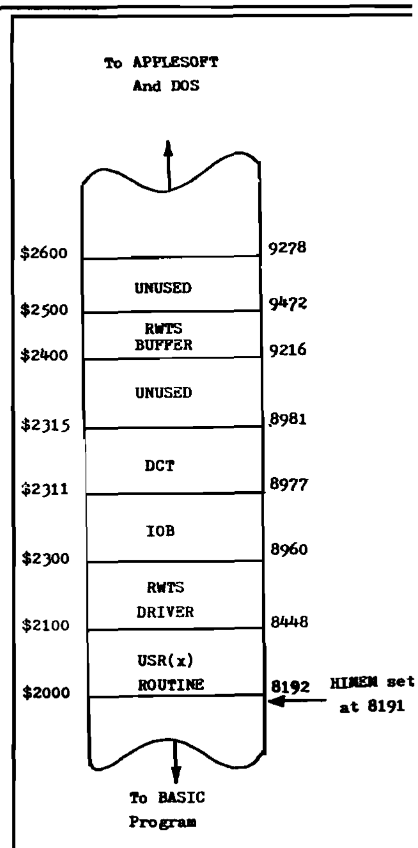


***Figure 2:* Memory Map**

The rest of this explanation assumes that the reader is somewhat familiar with Chapter 9 and Appendix C of the DOS manual. If not, he should read it before continuing with this article so that the terminology is familiar.

When the program is RUN it first sets HIMEM, then POKEs the first of three machine language programs into the protected area and asks the user to insert the disk to be searched into the drive. (The normal default drive of slot 6, drive 1, is used. To utilize another, line 1300 should be changed.) Subroutine 960 then sets up the RWTS driver, IOB and DCT described on page 94 of the DOS manual.

Listing 2

```
                    ;*
                    ;* RWTS DRIVER
                    ;*
                    RWTS    EQU $3D9
                    IOBADD  EQU 2300          ;IOB ADDRESS
                    ;
2100                        ORG $2100
2100
2100                ;
2100 A923                   LDA /IOBADD
2102 A000                   LDY #IOBADD
2104 20D903                 JSR RWTS
2107 60                     RTS               ;RETURN TO BASIC
```

The RWTS Driver (shown in listing 2) serves to load the 6502 microprocessor (registers A and Y) with the IOB address, and then JSR to the entry point of the RWTS subroutine. The Input/Output control Block (IOB) contains the critical operating parameters for the RWTS subroutine. These are initialized as shown in table 1. The Device Characteristics Table (DCT) has been placed immediately following the IOB. Its contents are determined by the actual physical characteristics of the disk drive itself, as well as the interface card and DOS. The standard values which DOS uses are also given in table 2.

Line 1090 protects all of this from Applesoft BASIC and also protects the short machine language program at memory address 8192. This program is one of two which are called by the Applesoft USR (x) function. The USR (x) routine defined at line 1350 (listing 3) is used to calculate the number of free sectors on the disk by utilizing the Track Bit Map found in the Volume table of contents (Track $11, Sector $00). Once this has been done the USR (x) function is redefined (listing 4) to perform decimal to hex conversion. See figure 2 for a memory map.

Listing 3

```
               ;*
               ;* ROUTINE TO COUNT 1'S IN
               ;*        INTEGER X
               ;*
               XTOINT EQU $E10C
               AYTOFP EQU $E2F2
               ;
               BYTA   EPZ $A0
               BYTB   EPZ $A1
               ;
2000                  ORG $2000
2000
2000           ;
2000 200CE1           JSR XTOINT     ;CONVERT X TO 16-BIT INTEGER
2003 A5A0             LDA BYTA       ;A=1ST BYTE OF INTEGER
2005 A000             LDY #$00       ;Y=BIT ACCUMULATOR
2007 A209             LDX #$09       ;X=LOOP COUNTER
2009 18               CLC            ;INITIALIZE CARRY
200A 2A        LBLA   ROL            ;LOOK AT NEXT BIT
200B 1001             BPL LBLB       ;SKIP ACCUMULATOR IF MSB IS ZERO
200D C8               INY            ;ELSE BUMP BIT ACCUMULATOR
200E CA        LBLB   DEX            ;DECREMENT LOOP COUNTER
200F D0F9             BNE LBLA       ;LOOP TILL DONE
2011 A5A1             LDA BYTB       ;A=2ND BYTE OF INTEGER
2013 A209             LDX #$09       ;NOW
2015 18               CLC            ; REPEAT
2016 2A        LBLC   ROL            ;  ABOVE
2017 1001             BPL LBLD       ;   FOR
2019 C8               INY            ;     SECOND
201A CA        LBLD   DEX            ;      BYTE
201B D0F9             BNE LBLC
201D A900             LDA #$00       ;A=0 FOR FP CONVERSION
201F 20F2E2           JSR AYTOFP     ;CONVERT A,Y TO FLOATING POINT
2022 60               RTS            ;RETURN TO BASIC
```

Referring to figure 1 for the following discussion, the BFILE search begins by picking up bytes 1 and 2 from the VTOC (statement 170). (Note that byte 1 is actually the 2nd byte; the first is byte 0.) These bytes contain the track and sector numbers, respectively, of the first directory-sector. Once known, that sector is read into the RWTS buffer by line 230.

Each directory sector contains up to seven directory entries and a link to the next directory sector. This link, in bytes 1 and 2 of each directory sector, is captured by line 280.

Each of the seven directory entries is 35 bytes long, starting at byte 11 of the buffer. Byte 0 and byte 1 of each entry (e.g. buffer bytes 11 and 12 for the first entry) contain the track and sector numbers, respectively, of the Track and Sector List (TSL) for that entry. If both bytes are zero, it indicates that the end of the directory has been reached. If byte 0 contains a 255 (hexadecimal FF), it indicates that the entry was once used, but since has been deleted. Only if both bytes are non-zero and less than 255 is the entry a valid entry.

Listing 4

```
               ;*
               ;* PRINT HEX EQUIVALENT OF DECIMAL INTEGER
               ;*
               XTOINT EQU $E10C
               AXTOHX EQU $F941
               ;
               BYTA   EPZ $A0
0800           BYTB   EPZ $A1
2000           ;
2000                  ORG $2000
2000
2000           ;
2000 200CE1           JSR XTOINT     ;CONVERT X TO 16-BIT INTEGER
2003 A5A0             LDA BYTA       ;A=MS BYTE
2005 A6A1             LDX BYTB       ;B=LS BYTE
2007 2041F9           JSR AXTOHX     ;PRINT AX IN HEX
200A 60               RTS            ;RETURN TO BASIC
```

Once the entry has been determined valid, byte 2 (of that entry) is examined to determine the file type. A "4" indicates an unprotected binary file and a "132" indicates a protected file. For either of these cases, the BFILE name is retrieved from bytes 3 through 13 and the track and sector numbers in bytes 0 and 1 are used to pull in the first sector of the TSL for the file (line 680). (Otherwise, the search continues with the next directory entry.)

The TSL is normally used to link multiple sectors of a program together. For our purpose, only bytes 12 and 13 are of interest. These two bytes contain the usual (by this time) track and sector of the first valid sector of the BFILE. Line 730 then pulls this sector into the buffer.

After picking out the start address and length of the BFILE (lines 780 and 790) and printing them in hex (and decimal), line 860 restores the original catalog sector to the buffer and the search continues.

After the seventh directory entry, assuming that a double-zero end-of-directory mark is not found, the next directory sector is loaded and the search continues with that and each succeeding directory sector.

Once the directory search is completed (determined by line 360) the program prints the number of free sectors and terminates.

The routines and techniques presented here can be utilized to implement a variety of "CATALOG" type programs which can be tailored to the user's individual needs. For instance, changing line 940 from "...POKE RD,1..." to "...POKE RD,2..." will write the buffer to the designated sector instead of reading from the sector to the buffer. However, a strong word of caution is in order: when debugging this type of program it is *extremely* easy to erase all or part of a disk. For this reason, always use a scratch disk when "RUNNING" the program (until it is thoroughly debugged) and "SAVE" the program on another disk prior to "RUNNING".

Clyde Camp has a BSEE from Virginia Polytechnic Institute (1969) and an MS in Computer Science from Southern Methodist University (1974). He has been employed by Texas Instruments since 1969 and is currently Systems Engineering Manager for the Industrial Controls division of Texas Instruments in Johnson City, Tennessee. His system consists of an Apple II with 48K, single disk, Heathkit printer, Integer and Applesoft ROM.

MICRO

Expressions Revealed, Part 1

Assemblers, compilers, and interpreters all have to be able to process expressions. This article, and the visually-oriented Apple II programs included, reveal the inner workings of expression processing — scanning, parsing, and translation.

Richard C. Vile, Jr.
3467 Yellowstone Dr.
Ann Arbor, Michigan 48105

Almost all programming languages allow the programmer to form a variety of expressions. In fact, expressions are such a "fact of programming life" that few programmers think much about them, beyond their application in programs. Nonetheless, a study of the processing of expressions by translators such as interpreters, assemblers, and compilers provides an interesting and worthwhile look "behind the scenes." In this article we shall present some simple techniques for the scanning, parsing, and translation of expressions. Programs, for the Apple II computer, will be presented which visually reveal the inner workings of some of the classical algorithms in this area.

The Makeup of Expressions

In the world of expressions, the cast of characters consists of *operators* and *operands*. Operands are themselves considered to be expressions, with the simplest being *constants* and *variables*. Of course, constants and variables represent but a small portion of the entire taxonomy of expressions. Simple expressions may be combined using operators to "make big ones out of little ones" (to paraphrase a well-known saying).

Each expression, great or small, represents a *value* of some *type*. One or more operators appropriate to each type are provided by a language. Table 1 catalogues some of the most common types and operators.

Type	Operators
Integer	+ − * / MOD REM = # < > <= >=
Real	+ − * / ↑ = # < > <= >=
String	+ (concatenation)
Boolean	AND OR NOT

***Table 1:* Types and Operators**

In the abstract, each operator must be applied to operands which are of the *same* type and are consistent with the type of operands which are expected by the operator. Thus, a relational operator such as < = applies to two numbers of the same type (both real or both integer) and not to logical values such as TRUE or FALSE. Likewise, the boolean operator AND does not apply (logically) to numerical values. Now, in the early days of high-level programming languages, the attitude toward such matters was quite lenient. Operators were allowed to "coerce" their operands into an appropriate form. After all, everything was eventually represented in terms of binary numbers inside the computer anyway. So, for example, in BASIC it is legal to write:

```
IF (X< Y) * (Y< Z) THEN ...
```

This is so since logical values are represented by the numbers 0 and 1 and may be treated as integers in BASIC. We *know* that the internal representation of FALSE is 0, and consequently that the expression (X<Y) * (Y<Z) will represent FALSE if *either* X<Y or Y<Z. Of course, instead of being so clever, we could simply have written

```
IF (X< Y) AND (Y< Z) THEN ...
```

instead. Knowledge about how information is represented inside the machine has gradually become less and less necessary in order to use high level languages effectively. Consequently, the rule of "different strokes for different folks" is *strictly enforced* in languages like Pascal. Writing the expression (X<Y)*(Y<Z) in Pascal will get you a severe scolding from the Pascal compiler. So, we speak of Pascal as a *type-checking* or *type-enforcing* language.

While one way of classifying operators is by the types of their operands, another is by the *number* of operands they require. Ninety-nine and forty-four one-hundredths percent of all operators require either one or two operands. Those requiring two operands are called *binary* operators, whereas those that require only one operand are referred to as *unary* operators.

The Meaning of Expressions

In order to be evaluated by a computer, expressions written in a high-level language must first be translated into a sequence of simpler, low-level instructions. Such instructions may be the machine language for a real processor such as the 6502, or the *pseudo-code* for an imaginary or *virtual* machine which is imitated by an interpretive program instead of a real processor. Each such instruction will typically manipulate only one or two operand quantities and involve, at most, one operator. In order to make the transition from a higher to a lower level form, we must be able to decide in which order to carry out the individual operations indicated by the original expression. This means that expressions which involve more than one operator must be made unambiguous as to the order of evaluation.

Consider the expression X + Y * Z. This expression could mean either of two quantities:

a. the result of adding X and Y followed by multiplication by Z.

b. the result of multiplying Y and Z followed by addition of X.

There is no "correct" choice between these two possibilities, only various *conventions* or methodologies dictate which choice to make. Each high-level language must select one such convention in order to make its expressions intelligible. Let us consider some of the techniques which may be used.

Left-to-Right Evaluation

This is perhaps the simplest method. The convention is that if we scan from left to right in the expression, then each operator will be evaluated as soon as it is encountered, using the result so far obtained as the "left" operand, and the variable immediately following the operator as the "right" operand. Using this rule will cause our sample expression to be interpreted as indicated by possibility *a* described earlier. In order to achieve the result indicated by possibility *b*, the expression would have to be rewritten as: Y * Z + X.

Very few, if any, languages rely *solely* on the left-to-right rule. However, nearly all languages do use it in some contexts, as we shall see.

Use of Parentheses to Group Operands

Another simple way to make expressions completely unambiguous is to use "fully parenthesized" notation. This means that enough parentheses must be supplied in order to uniquely specify the two operands of each operator in the expression. For the example under discussion, the two possible meanings given would be written as:

(X + Y) * Z and X + (Y * Z)

respectively.

Precedence	Operators
1	NOT
2	AND, OR
3	<>, =, <, >, <=, >=
4	+, -
5	*, /
6	↑

Figure 1

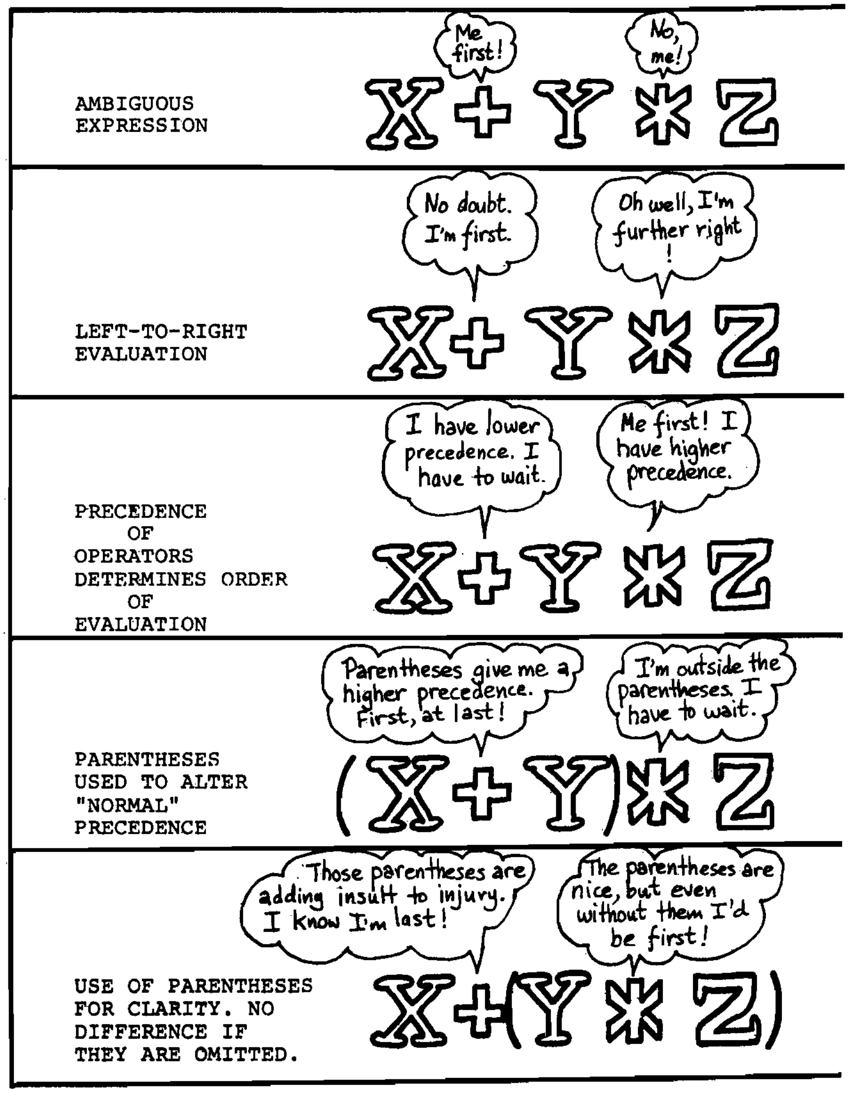


Figure 2

Precedence Rules

The method of choice in nearly all modern languages is the use of precedence rules. Each operator is assigned a *precedence level* (or simply, *precedence*). This establishes a "pecking order" among the operators. When it comes to the evaluation of an expression those operators with higher precedence levels are evaluated *first*. They take precedence (hence the terminology) over those operators at lower levels. Figure 1 illustrates a typical assignment of precedence levels, in this case for the BASIC language. Using that assignment of levels, the expression X + Y * Z would be considered equivalent to (X + Y) * Z, since * has a higher precedence than +.

Precedence rules alone do not us ly suffice for common practice, h ever. Two issues are not resolved i rely solely on precedence:

1. How do we decide the orde evaluation of operators w have been assigned the s precedence level (e.g. ' + ' and in figure 1)?
2. How do we *defeat* the order plied by the precedence leve we so desire?

The solutions are simple! For the f use left-to-right evaluation. For the ond, use parentheses. Thus, using left-to-right rule will tell us that th pression X + Y − Z should be i preted to mean (X + Y) − Z. Likev we may always write (X + Y) * Z, when we desire the addition to precede the multiplication. Parentheses may be thought of as *boosting* the precedence levels of all the operators they contain, in order to make them higher than all the operators outside.

Figure 2 summarizes the techniques and conventions under discussion, using the expression X + Y * Z as the example.

Translation of Expressions

The notation used in writing expressions is sometimes referred to as *infix* notation. This obviously derives from the fact that the operators appear in-between their operands:

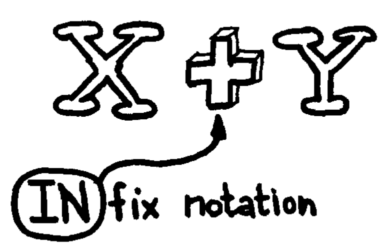


Figure 3

Infix notation is potentially ambiguous as we have seen. Translation of an expression usually replaces the human oriented infix notation with a more machine-oriented notation.

A very common choice for the intermediate representation exists which requires no parentheses at all. It is known as *postfix* notation and is characterized by the fact that each operator always immediately follows its operands. Thus, the infix expression X + Y will be written as follows:

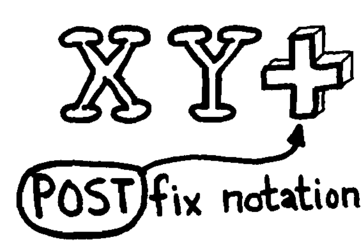


Figure 4

The order of evaluation in a postfix notation expression is *always* completely specified by a single left-to-right scan. To change the order of evaluation, the order of the operators is changed. Figure 5 shows the two possible postfix versions of the expression X + Y * Z, corresponding respectively to (X + Y) * Z and X + (Y * Z).

The fact that postfix notation is completely unambiguous makes it a strong candidate for use as the pseudo-code of a virtual machine representation for expressions. Some machines and/or systems go so far as to use postfix notation, or Reverse Polish Notation (RPN) as it is also called in the external representation of expressions as well. For example, the handheld calculators manufactured by Hewlett-Packard require its use.

Also, one computer language which has recently gained much popularity, namely FORTH, requires that expressions and statements, as well, be expressed in RPN. (A description of the FORTH language is beyond our purpose in this article, but we mention it to illustrate the importance and pervasiveness of postfix form.)

Given that it is desirable to use RPN as an internal form for representing expressions, we arrive at the first roadblock: How are parenthesized, infix notation expressions translated into RPN? The answer is embodied in one of the classical algorithms of computer science. Its description will occupy most of the remainder of this article.

The conversion algorithm makes use of a data structure known as a *stack*. The stack concept has gradually crept into the spotlight, especially since the advent of the microprocessor. A stack is a storage mechanism first of all — it may be used to store objects of computation: numbers, characters, strings, records, etc. It uses a storage discipline known as the "last-in first-out" method: *last* or most recent item to be stored in the stack is always the *first* to be available for retrieval. The *operations* which may be performed on a stack are:

PUSH(Item): This operation causes "Item" to be stored at the TOP of the stack (see below for more on the TOS — *"Top Of Stack"*).

POP(Loc): This operation causes the Item currently stored at the TOP of the stack to be removed from the stack, or "Popped off" the stack and transferred into the memory location represented by "Loc."

The concept of Top Of Stack, abbreviated TOS, may be explained as follows:

Top Of Stack: The last location in the stack into which an item was stored is defined to be the Top Of Stack. When a PUSH operation is performed, the Top Of Stack is *first* advanced one location, before storing the Item being PUSHed onto the stack. When a POP operation is performed, the Top Of Stack recedes by one location, *after* the Item being POPped off the stack is transferred.

When the stack is empty, that is, no items have ever been pushed onto the stack, then the Top Of Stack is conceptually one location *before* the first location available for the stack. At first this is a bit awkward for some people to comprehend, since it means that the "Top" of the stack is in some sense "outside" the stack. However, since TOS is advanced before the data is stored during a PUSH, this awkwardness is healed by the first PUSH operation that takes place when a stack is used. However, trying a POP on an empty stack will only lead to headache #95!

When a stack is *full*, then TOS corresponds to the last location available for stack storage. Thus any further attempt to PUSH an item will cause the stack to "overflow."

All of this may be old hat to many readers, but for the novitiates, figures 6-8 illustrate the above terminology and explanations. Also, if analogies are near and dear to your heart, you may compare a stack to many similar entities in the real world: a stack of papers, a pile of dishes, a stack of pancakes, a railroad siding track, and so on.

Listing 1 presents an Integer BASIC program which implements an interesting game that illustrates simple manipulations using a stack. The object of the "game" is to rearrange a string of digits into a different order. The original string is in the counting order 12...n, where n in our implementation may be, at most, 9. The "goal" or "target" string is a randomly selected permutation of the original. Thus, for example, if n = 5 the original string will be 12345 and the target string might be 53124, or any permutation of 12345.

The rules of the game are quite simple. The original string is scanned from left to right in order to attempt to achieve the rearrangement. Since one

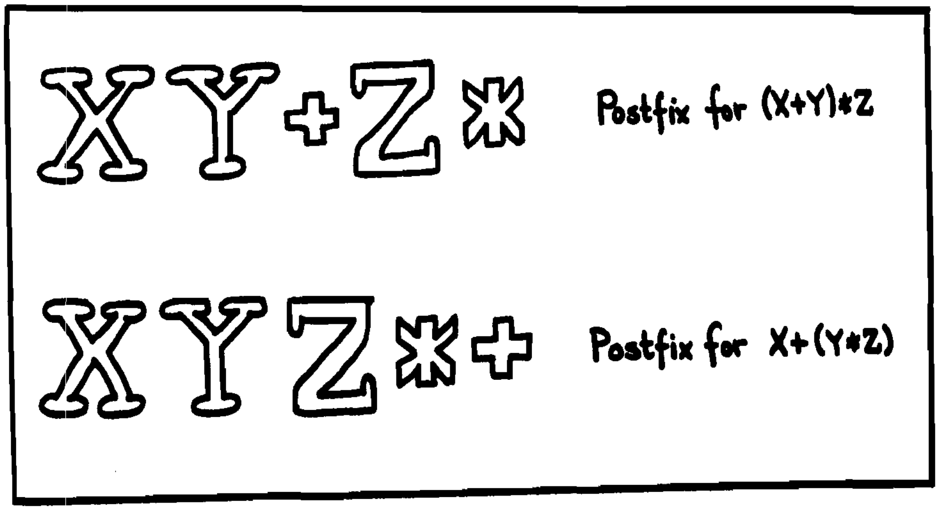


Figure 5

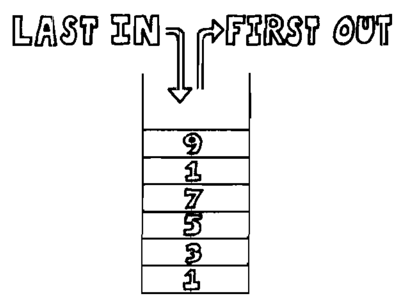


Figure 6: **A STACK of Integers**

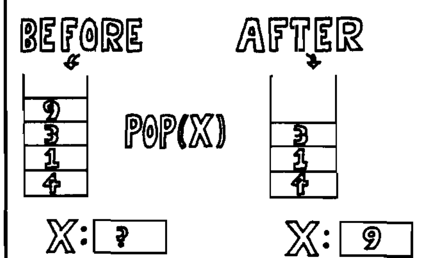


Figure 8: **A POP Operation**

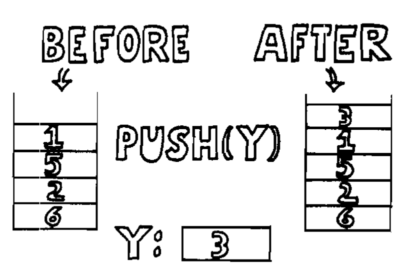


Figure 7: **A PUSH Operation**

LSSOP	3	←TOS
EQLOP	13	
ANDOP	12	
EXPOP	26	
MULTOP	5	
PLUSOP	14	

Figure 9: **A STACK of Records**

scan may not suffice to achieve the target string, repeated scans are allowed with the intermediate results copied back into the input string. The scanning process allows digits to be PUSHed onto a stack and later POPped from the same stack onto an output string. More precisely, at each stage of a given scan, one digit of the input string will be in the spotlight. This digit must eventually be PUSHed onto the stack, at which point the scan will advance to the next digit. However, if at any point there are digits in the stack, they may be POPped (some or all) onto the output string. The output string is added to at its right end, whenever a new digit is POPped onto it. Note that when the stack is empty, the only option is to PUSH the current digit and advance to the next. The input may be copied without alteration to the output by merely repeating the sequence:

PUSH POP PUSH POP ...

for as many digits as there are in the input. Finally, when the scan reaches the end of the string, the stack will be emptied onto the output.

The play of the game involves not only achieving the rearrangement of the original string, but also in doing so with the least number of scans possible. Hint: It is always possible to achieve any target string from the original string 123...n in *at most* n scans. This is because it is always possible to put *one more* digit into its correct position on a given scan.

Returning to the question of converting an infix notation expression to RPN, the translation algorithm we shall discuss will make use of a stack of "operators" to assist in its job. Actually the algorithm needs to keep track of not only *what* the operators are, but also what their precedence is in the expression being scanned. Therefore, each entry in the stack of "operators" will contain two pieces of information: an *identification* of the operator concerned, and its precedence in the expression. This idea of a stack of "compound" items is illustrated in figure 9. Later we shall present two implementations of the translation algorithm, one in BASIC and one in Pascal. The implementation in Pascal uses a particularly convenient representation of the stack as a Pascal *record* type.

Infix to Postfix: The Translation Algorithm

The input to the translation algorithm will be an expression in partially parenthesized infix form. The expression will be scanned from left to right and dissected into its component parts:

Operands
Operators
Parentheses

(Blanks embedded in the input will be considered to be insignificant.)

The output of the translation will consist of a string, containing all the operands and operators of the input, but with all parentheses removed. The string will represent the RPN for the input expression.

As the input is dissected, the "object" being scanned at any point will determine the action to be taken. These objects are also referred to as *tokens*. It is the job of the *scanner* to extract tokens. In our implementations of the translation algorithm, the scanner will be quite simple. Each token will be assumed to be only a single character long. The scanner will examine each

```
        ┌TOS ← 0
   1.  {  STACK(TOS) ← (NOOP, -2)
        │NEST ← 0
        └DONE ← FALSE
     ┌─ while NOT DONE do
     │  3. TOKEN ← SCAN;
     │   ┌case TOKEN of
     │   │OPERAND:      OUTPUT(OPERAND);
     │   │LPAREN:       NEST ← NEST + 1;
     │   │RPAREN:       NEST ← NEST - 1;
2.  {    4{ OPERATOR:   ┌begin
     │   │              │6. NOWP ← NEST*10+PRECEDENCE(OPERATOR);
     │   │              │   while NOWP<PRECEDENCE(TOS) do
     │   │       5.    {  8.   POP(OUTPUT);
     │   │              │   endwhile;
     │   │              │7. PUSH(OPERATOR,NOWP);
     │   │              └end;
     │   └______endcase;
     └─ endwhile;
```

Figure 10: Pseudo-code for Translation Algorithm

```
                         ( ( X + Y ) / ( Z - ( W * U ) ) ↑ A ) / B

Nesting Level:           1 2       1   2     3       2 1    0

Absolute Precedence:         4     5     4     5       6     5

Relative Precedence:        24    15    24    35      16     5
```

Figure 11: Absolute vs. Relative Precedence

character and assign it an internal "token number" which may be more convenient for the remainder of the program to manipulate.

Figure 10 presents the essential details of the algorithm expressed in *pseudo-code*. Various portions of the program have been bracketed and/or numbered in order to provide reference points for further discussion.

1. Initialization

The stack is initially set up with a "dummy entry" which is needed for two reasons:

a. In order to allow the test in the *while* loop labelled *5* to make sense when no operators have yet been pushed onto the stack.

b. In order to provide a way to stop the same loop when the stack is "emptied out" at the end of the scanning process.

The pair (NOOP, -2) is put onto the bottom of the stack to accomplish these goals. The nesting level of parentheses is given its initial value of 0 (in the variable NEST), and the logical variable DONE is set to FALSE: we can't be DONE, we've only just begun!

2. Main Program Loop

The fundamental control structure of the algorithm is a *while* loop (a loop controlled by a condition which is tested *before* any statements of the loop are executed on each pass through) controlled by the logical expression "NOT DONE." The variable DONE will become TRUE when both of the following conditions are met:

a. The input expression has been completely scanned.

b. The OPERATOR stack has been emptied to the output.

The details of how these tests are carried out in the implementation may be gleaned by studying the actual programs of listings 2 and 3, which will be presented in part 2, next month.

3. Token Extraction

While in general this process may be as painful as tooth extraction, in our case it is relatively simple. A routine must be provided which picks off the next character of the input and converts it into the internal form that is used by the remainder of the algorithm. In the pseudo-code incarnation this is called SCAN and it is invoked each time at the head of the main program loop. The routine SCAN is actually a function (with no actual arguments) which has its returned value assigned to the variable TOKEN.

4. Translation Actions

The actions taken by the translator at each step depend on the TOKEN found. The pseudo-code uses a case statement to select the appropriate action based on the value of TOKEN. The possible categories of TOKEN are:

OPERAND
LPAREN
RPAREN
OPERATOR

For each of these categories, the case statement specifies corresponding actions:

a. OPERANDS are immediately copied to the output.

b. Left parentheses (LPAREN) cause the variable NEST to increase by 1.

c. Right parentheses (RPAREN) cause the variable NEST to decrease by 1.

d. OPERATORS cause the section of code labelled *5* to be executed.

5. Stack Manipulation for Operators

This section represents the heart of the translation algorithm. Since decisions are made based on the values of PRECEDENCE, these values are calculated for each operator (see *6* below). In addition, operators are PUSHed and POPped from the stack based on the precedence values calculated.

6. Calculation of Operator Precedence

Each operator of the input expression has an associated precedence calculated according to the formula:

NOWP = NEST * 10 +
PRECEDENCE(OPERATOR)

This value represents the *relative* precedence of the operator within the particular expression at hand. It is based on the *absolute* precedence, PRECEDENCE(OPERATOR), of the operator and the nesting level within the expression. The absolute values of precedence in our implementations are all less than 10. The factor NEST * 10 is therefore guaranteed to boost all the values for operators inside a given pair of parentheses to be higher than all those outside. Figure 11 shows a fairly complex expression, with each operator labelled with its nesting level, absolute precedence, and relative precedence.

7. PUSHing Operators onto the Stack

8. POPping Operators from the Stack

Each operator in the input expression must eventually be PUSHed onto the stack; none go directly to the output. When an operator is encountered in the input, its relative precedence is calculated and compared with that of the operator on top of the stack. As long as the TOS operator has *higher* precedence, it will be POPped to the output — this is expressed by the *while* loop at 8. When control falls out of that loop, the current operator is then PUSHed onto the stack (i.e. the pair of values "operator, relative precedence") and the main loop is repeated.

Figure 12 gives a history of the execution of the translation algorithm at work on the input expression:

$$Z = (X + Y) * (X - Y) + (U + V)$$

For lack of space, we have shown the stack with only the operator characters. The column headed LASTP always shows the relative precedence for the operator at the top of the stack. The arrows in the EXPRESSION column mark the progress of the scan. The column headed < ? tells whether the current precedence is less than PRECEDENCE(TOS).

EXPRESSION	OUTPUT	NEST	NOWP	LASTP	<?	STACK
Z=(X+Y)*(X−Y)+(U+V)	0	0	−1	−2	F	0
Z=(X+Y)*(X−Y)+(U+V)	Z	0	−1	−2	F	0
Z=(X+Y)*(X−Y)+(U+V)	Z	0	3	−2	F	=
Z=(X+Y)*(X−Y)+(U+V)	Z	1	3	3	F	=
Z=(X+Y)*(X−Y)+(U+V)	ZX	1	3	3	F	=
Z=(X+Y)*(X−Y)+(U+V)	ZX	1	14	3	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY	1	14	14	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY	0	14	14	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY	0	5	14	T	=+
	ZXY+	0	5	3	F	=
	ZXY+	0	5	5	F	=*
Z=(X+Y)*(X−Y)+(U+V)	ZXY+	1	5	5	F	=*
Z=(X+Y)*(X−Y)+(U+V)	ZXY+X	1	5	5	F	=*
Z=(X+Y)*(X−Y)+(U+V)	ZXY+X	1	14	5	F	=*−
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY	1	14	14	F	=*−
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY	0	14	14	F	=*−
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY	0	4	14	T	=*−
	ZXY+XY−	0	4	5	T	=*
	ZXY+XY−*	0	4	3	F	=
	ZXY+XY−*	0	4	4	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*	1	4	4	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*U	1	4	4	F	=+
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*U	1	14	4	F	=++
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*UV	1	14	14	F	=++
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*UV	0	14	14	F	=++
Z=(X+Y)*(X−Y)+(U+V)	ZXY+XY−*UV	0	−1	14	T	=++
	ZXY+XY−*UV+	0	−1	4	T	=+
	ZXY+XY−*UV++	0	−1	3	T	=
	ZXY+XY−*UV++=	0	−1	−2	F	0

Final Output = = = > ZXY+XY−*UV++=

***Figure 12:* Trace of Infix to Postfix Translation**

Listing 1

```
  10 DIM STACK( 9 ),TARGET( 9 ),OUTPUT(
     9 )
  11 DIM CURRENT( 9 )
  15 INTRO=9000:SETUP=8000
  16 HOME=-936:CLREOL=-868:KBD=-
     16384:CLR=-16368
  17 GETKEY=3000:WAIT=3100:PERMUTE=
     3200
  18 FLASHINIT=3300:PUSH=3400:PULL=
     3500
  19 CHRDOLLAR=3600:SCAN=2000
  20 DISPLAY=3700:INIT=3800
  21 POINTS=3900:AGAIN=4000:RESTART=
     8050
  50 STARTLINE=2:STACKLINE=4:MENULINE
     =12
  51 OUTPUTLINE=6:TARGETLINE=9
  52 ERRLINE=17:DEBUGLINE=17
1000 REM MAIN PROGRAM
1001 REM ===========
1010 GOSUB INTRO
1012 GOSUB SETUP
1015 GOSUB INIT
1018 GOSUB RESTART
1020 GOSUB SCAN
1025 GOSUB POINTS
1030 IF NOT DONE THEN 1015
1035 GOSUB AGAIN
1040 IF NOT ADDIO THEN 1012
1099 CALL HOME: END
2000 REM SCAN CURRENT STRING ONE
2001 REM CHARACTER AT A TIME AND
2002 REM REQUEST USER MOVES.
2003 REM =====================
2005 SCANPTR=1
2010 GOSUB DISPLAY
2015 VTAB MENULINE: TAB 1: PRINT
     "   CHOOSE ONE OF THE FOLLOWING:"

2020 TAB 5: PRINT LBRA$;PU$;"] PUSH"

2022 TAB 5: PRINT LBRA$;PO$;"] POP"

2025 VTAB ERRLINE: CALL CLREOL
2030 TAB 5: GOSUB GETKEY
2035 IF KEY#PULLKEY THEN 2040
2037 GOSUB PULL: GOTO 2015
2040 IF KEY#PUSHKEY THEN GOTO 2015

2045 GOSUB PUSH
2050 SCANPTR=SCANPTR+1
2055 IF SCANPTR<=SLEN THEN 2010
2060 IF STACKPTR<=0 THEN 2099
2065 GOSUB PULL: GOTO 2060
2099 RETURN
3000 REM GETKEY ROUTINE
3001 REM =============
3005 KEY= PEEK (KBD)
3010 IF KEY<128 THEN 3005
3015 IF KEY>=161 AND KEY<=222 THEN
     3040
3020 POKE CLR,0: GOTO 3005
3040 POKE CLR,0
3049 RETURN
3100 REM STANDARD WAIT ROUTINE
3101 REM ====================
3105 POKE CLR,0
3110 POKE 50,63: VTAB 24: TAB 5
3115 PRINT "PRESS ANY KEY TO CONTINUE
     ";
3120 POKE 50,255
3125 IF PEEK (KBD)<128 THEN 3125

3130 POKE CLR,0
3135 VTAB 24: TAB 1: CALL CLREOL
3149 RETURN
3200 REM SET UP TARGET STRING
3201 REM AND INITIALIZE THE
3202 REM CURRENT POSITION ARRAY.
3203 REM ======================
3205 FOR I=1 TO SLEN:CURRENT( I )=
     I: NEXT I
3210 FOR I=1 TO 9:TARGET( I )=0: NEXT
     I
3215 FOR I=1 TO SLEN
3220 L= RND (SLEN)+1: IF TARGET(
     L )>0 THEN 3220
3225 TARGET( L )=I
3230 NEXT I
3245 COUNT=0
3249 RETURN
3300 REM  POKE IN THE FLASHIT
3301 REM  SUBROUTINE
3302 REM
3305 POKE 1,201
3306 POKE 2,160
3307 POKE 3,176
3308 POKE 4,3
3309 POKE 5,76
3310 POKE 6,240
3311 POKE 7,253
3312 POKE 8,201
3313 POKE 9,192
3314 POKE 10,176
3315 POKE 11,6
3316 POKE 12,56
3317 POKE 13,233
3318 POKE 14,64
3319 POKE 15,76
3320 POKE 16,240
3321 POKE 17,253
3322 POKE 18,233
3323 POKE 19,128
3324 POKE 20,76
3325 POKE 21,240
3326 POKE 22,253
3330 FLASH=3350:REGULAR=3375
3349 RETURN
3350 POKE 54,1: POKE 55,0: RETURN

3375 POKE 54,189: POKE 55,158: RETURN

3400 REM PUSH CURRENT DIGIT ONTO
3401 REM STACK.
3402 REM ======================
3405 STACKPTR=STACKPTR+1
3410 VTAB STACKLINE: TAB 10+STACKPTR
3415 PRINT CURRENT( SCANPTR );
3420 STACK( STACKPTR )=CURRENT( SCANPTR )

3449 RETURN
3500 REM POP STACK TO OUTPUT AND
3501 REM UPDATE DISPLAY.
```

(Continued)

```
3502 REM =====================
3503 IF STACKPTR>0 THEN 3509
3504 GOSUB FLASH: PRINT ""
3505 VTAB ERRLINE: TAB 5: PRINT
     "EMPTY STACK"
3506 GOSUB REGULAR: GOSUB WAIT
3507 RETURN
3509 TOS=STACK(STACKPTR)
3510 VTAB STACKLINE: TAB 10+STACKPTR
3511 PRINT " ";
3515 VTAB OUTPUTLINE: TAB 18+OUTPTR
3520 PRINT TOS;
3522 OUTPUT(OUTPTR)=TOS
3525 OUTPTR=OUTPTR+1
3530 STACKPTR=STACKPTR-1
3549 RETURN
3600 REM CONVERT NUM TO CHARACTER
3601 REM INTEGER BASIC CHR$ FUNCTION
3602 REM IN USER CONTRIBUTED SOFT-
3603 REM WARE.
3604 REM ========================
3610 CHS=CHR+128*(CHR<128)
3615 LC1= PEEK (224):LC2= PEEK (
     225)-(LC1>243): POKE 79+LC1-
     256*(LC2>127)+(LC2-255*(LC2>
     127))*256,CHS:CHR$="-": RETURN

3700 REM DISPLAY CURRENT SCAN
3701 REM POSITION IN INVERSE
3702 REM ==================
3705 GOSUB FLASH
3710 VTAB STARTLINE: TAB 18+SCANPTR
3715 PRINT CURRENT(SCANPTR)
3720 GOSUB REGULAR
3725 IF SCANPTR=1 THEN RETURN
3730 VTAB STARTLINE: TAB 18+SCANPTR-
     1
3732 PRINT CURRENT(SCANPTR-1)
3749 RETURN
3800 REM INIT IMPORTANT VARIABLES
3801 REM =======================
3805 STACKPTR=0
3810 OUTPTR=1
3811 DONE=0
3815 GOSUB FLASHINIT
3899 RETURN
3900 REM CHECK IF TARGET STRING
3901 REM HAS BEEN ACHIEVED. IF
3902 REM SO, THEN SET DONE=TRUE;
3903 REM OTHERWISE, BUMP COUNT
3904 REM AND SET DONE=0
3910 FOR I=1 TO SLEN
3915 IF TARGET(I)#OUTPUT(I) THEN
     3950
3920 NEXT I
3925 REM TARGET AGREES WITH OUTPUT
3926 REM SO WE ARE "DONE".
3927 REM =======================
3930 DONE=1
3935 COUNT=COUNT+1: RETURN
3950 DONE=0
3955 REM COPY OUTPUT TO CURRENT
3956 REM FOR RESCAN. BUMP COUNT.
3957 REM =====================
3960 COUNT=COUNT+1
3965 FOR I=1 TO SLEN
3966 CURRENT(I)=OUTPUT(I)
3967 NEXT I
3999 RETURN
4000 REM SCORE PLAYER AND ALLOW
4001 REM DECISION AS TO RETRY.
4002 REM ====================
4005 VTAB DEBUGLINE: TAB 1
4010 GOSUB FLASH: PRINT "CONGRATULATI
     ONS!"
4011 GOSUB REGULAR: PRINT "YOU DID IT
      IN ";COUNT;" SCANS."
4012 PRINT "GO AGAIN? (Y/N)";: GOSUB
     GETKEY
4015 IF KEY#206 AND KEY#217 THEN
     4005
4020 IF KEY=217 THEN ADDIO=0
4025 IF KEY=206 THEN ADDIO=1
4030 VTAB DEBUGLINE: TAB 1: PRINT
     : PRINT : PRINT
4049 RETURN
8000 REM SETUP ROUTINE
8001 REM ============
8005 CALL HOME
8006 CHR=219: GOSUB CHRDOLLAR:LBRA$
     =CHR$
8010 VTAB 5: PRINT "PLEASE INDICATE L
     ENGTH OF STARTING"
8011 PRINT "STRING===>";: CALL CLREOL

8015 INPUT SLEN: IF SLEN>=1 AND
     SLEN<=9 THEN 8020
8018 PRINT "TRY AGAIN"
8019 GOTO 8010
8020 VTAB 7: PRINT "PLEASE HIT KEY YO
     U WISH TO"
8021 PRINT "USE FOR A PUSH";: GOSUB
     GETKEY:PUSHKEY=KEY
8022 CHR=PUSHKEY: GOSUB CHRDOLLAR:
     PU$=CHR$
8025 VTAB 9: TAB 1: PRINT "PLEASE HIT
      KEY YOU WISH TO"
8026 PRINT "USE FOR A POP";: GOSUB
     GETKEY:PULLKEY=KEY
8027 CHR=PULLKEY: GOSUB CHRDOLLAR:
     PO$=CHR$
8030 GOSUB PERMUTE
8049 RETURN
8050 REM RESTART ROUTINE
8051 REM CALLED IF NEW SCAN IS
8052 REM NEEDED; I.E. TARGET
8053 REM NOT REACHED.
8054 CALL HOME
8055 VTAB STARTLINE: PRINT "STARTING
     POSITION:";
8057 FOR I=1 TO SLEN: PRINT CURRENT(
     I);: NEXT I
8060 VTAB STACKLINE: TAB 1: PRINT
     "STACK===>"
8065 VTAB OUTPUTLINE: TAB 1: PRINT
     "OUTPUT POSITION:"
8070 VTAB TARGETLINE: TAB 1: PRINT
     "TARGET STRING:";
8071 FOR I=1 TO SLEN: PRINT TARGET(
     I);: NEXT I
8075 VTAB 23: TAB 1:CHR=PUSHKEY:
      GOSUB CHRDOLLAR
8076 PRINT "KEY FOR PUSH= '";CHR$
     ;"'";: PRINT "  KEY FOR POP= '" ;
```

(Continued

```
8077 CHR=PULLKEY: GOSUB CHRDOLLAR:
      PRINT CHR$;
8078 PRINT "/";
8099 RETURN
9000 REM INTRODUCTION AND RULES
9001 REM OF PLAY.
9002 REM =====================
9010 CALL HOME
9015 PRINT "  WELCOME TO THE GAME OF
     STACK!"
9016 PRINT : PRINT "THE OBJECT IS TO
     REARRANGE A STRING"
9017 PRINT "OF DIGITS, SUCH AS 123456
     , INTO A "
9018 PRINT "DIFFERENT ORDER, SUCH AS
     615342."
9019 PRINT "THE ORIGINAL STRING IS SC
     ANNED FROM LEFT";
9020 PRINT "TO RIGHT.  AT EACH DIGIT
     YOU HAVE THE"
9021 PRINT "FOLLOWING OPTIONS:"
9022 PRINT : TAB 5: PRINT "PUSH ===>
     PUTS THE CURRENT DIGIT ON"
9023 TAB 15: PRINT "THE STACK, AND CA
     USES THE"
9024 TAB 15: PRINT "SCAN TO GO TO THE
      NEXT"
9025 TAB 15: PRINT "DIGIT.": PRINT

9026 TAB 5: PRINT "POP  ===> TRANSFER
     S THE TOP OF THE"
9027 TAB 15: PRINT "STACK TO THE OUTP
     UT AND"
9028 TAB 15: PRINT "ALLOWS ANOTHER AC
     TION -"
9029 TAB 15: PRINT "I.E. PUSH OR POP
     - BEFORE"
9030 TAB 15: PRINT "ADVANCING THE SCA
     N."
9035 GOSUB WAIT
9040 CALL HOME
9045 VTAB 5: TAB 1: PRINT "  THE NUMB
     ER OF DIGITS TO BE"
9050 PRINT "REARRANGED IS CHOSEN BY T
     HE PLAYER,"
9051 PRINT "AS WELL AS THE KEYS TO BE
      USED TO "
9052 PRINT "INDICATE A PUSH OR A POP.
     "
9053 PRINT : PRINT "  THE ORIGINAL ST
     RING WILL BE SCANNED"
9054 PRINT "REPEATEDLY UNTIL THE TARG
     ET STRING IS"
9055 PRINT "ACHIEVED.  THE SCORING IS
      BASED ON THE"
9056 PRINT "NUMBER OF SCANS REQUIRED
     FOR THE"
9057 PRINT "PLAYER TO REACH THE TARGE
     T POSITION."
9998 GOSUB WAIT
9999 RETURN
```

MICRO

The Newest In

Apple Fun

We've taken five of our most popular programs and combined them into one tremendous package full of fun and excitement. This disk-based package now offers you these great games:

Mimic—How good is your memory? Here's a chance to find out! Your Apple will display a sequence of figures on a 3 x 3 grid. You must respond with the exact same sequence, within the time limit.

There are five different, increasingly difficult versions of the game, including one that will keep going indefinitely. Mimic is exciting, fast paced and challenging—fun for all!

Air Flight Simulation—Your mission: Take off and land your aircraft without crashing. You're flying blind —on instruments only.

A full tank of fuel gives you a maximum range of about 50 miles. The computer will constantly display updates of your air speed, compass heading and altitude. Your most important instrument is the Angle of Ascent/Bank Indicator. It tells if the plane is climbing or descending, whether banking into a right or left turn.

After you've acquired a few hours of flying time, you can try flying a course against a map or doing aerobatic maneuvers. Get a little more flight time under your belt, the sky's the limit.

Colormaster—Test your powers of deduction as you try to guess the secret color code in this Mastermind-type game. There are two levels of difficulty, and three options of play to vary your games. Not only can you guess the computer's color code, but it will guess yours! It can also serve as referee in a game between two human opponents. Can you make and break the color code...?

Star Ship Attack—Your mission is to protect our orbiting food station satellites from destruction by an enemy star ship. You must capture, destroy or drive off the attacking ship. If you fail, our planet is doomed...

Trilogy—This contest has its origins in the simple game of tic-tac-toe. The object of the game is to place three of your colors, in a row, into the delta-like, multi-level display. The rows may be horizontal, vertical, diagonal and wrapped around, through the "third dimension". Your Apple will be trying to do the same. You can even have your Apple play against itself!

Minimum system requirements are an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive. Mimic requires Applesoft in ROM, all others run in RAM or ROM Applesoft.

Order No. 0161AD $19.95

Solar Energy For The Home

With the price of fossil fuels rising astronomically, solar space-heating systems are starting to become very attractive. But is solar heat cost-effective for you? This program can answer that question.

Just input this data for your home: location, size, interior details and amount of window space. It will then calculate your current heat loss and the amount of gain from any south facing windows. Then, enter the data for the contemplated solar heating installation. The program will compute the NET heating gain, the cost of conventional fuels vs. solar heat, and the calculated payback period—showing if the investment will save you money.

Solar Energy for the Home: It's a natural for architects, designers, contractors, homeowners...anyone who wants to tap the limitless energy of our sun.

Minimum system requirements are an Apple II or Apple II Plus with one disk drive and 28K of RAM. Includes AppleDOS 3.2.

Order No. 0235AD (disk-based version) $34.95

Math Fun

The Math Fun package uses the techniques of immediate feedback and positive reinforcement so that students can improve their math skills while playing these games:

Hanging—A little man is walking up the steps to the hangman's noose. But YOU can save him by answering the decimal math problems posed by the computer. Correct answers will move the man down the steps and cheat the hangman.

Spellbinder—You are a magician battling a computerized wizard. In order to cast death clouds, fireballs and other magic spells on him, you must correctly answer problems involving fractions.

Whole Space—Pilot your space craft to attack the enemy planet. Each time you give a correct answer to the whole number problems, you can move your ship or fire. But for every wrong answer, the enemy gets a chance to fire at you.

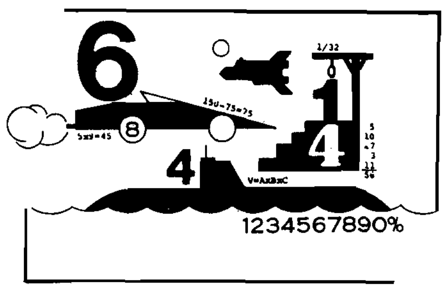
Car Jump—Make your stunt car jump the ramps. Each correct answer will increase the number of buses your car must jump over. These problems involve calculating the areas of different geometric figures.

Robot Duel—Fire your laser at the computer's robot. If you give the correct answer to problems on calculating volumes, your robot can shoot at his opponent. If you give the wrong answer, your shield power will be depleted and the computer's robot can shoot at yours.

Sub Attack—Practice using percentages as you maneuver your sub into the harbor. A correct answer lets you move your sub and fire at the enemy fleet.

All of these programs run in Applesoft BASIC, except Whole Space, which requires Integer BASIC.

Order No. 0160AD $19.95



Skybombers

Two nations, seperated by The Big Green Moun tain, are in mortal combat! Because of the terrain their's is an aerial war—a war of SKYBOMBERS!

In this two-player game, you and your opponen command opposing fleets of fighter-bombers arme with bombs and missiles. Your orders? Fly over th mountain and bomb the enemy blockhouse into dust

Flying a bombing mission over that innocent look ing mountain is no milk run. The opposition's aircra can fire missiles at you or you may even be destroye by the bombs as they drop. Desperate pilots may eve ram your plane or plunge into your blockhouse, sui cidally.

Flight personnel are sometimes forced to parachut from badly damaged aircraft. As they float helpless to earth, they become targets for enemy missiles.

The greater the damage you deal to your enemy, th higher your score, which is constantly updated at th bottom of the display screen.

The sounds of battle, from exploding bombs to th pathetic screams from wounded parachutists, remin each micro-commander of his bounden duty. Pres On, SKYBOMBERS—Press On!

Minimum system requirements: An Apple II or Ap ple II Plus, with 32K RAM, one disk drive and gam paddles.

Order No. 0271AD (disk-based version) $19.95

Paddle Fun

This new Apple disk package requires a steady eye and a quick hand at the game paddles! It includes:

Invaders—You must destroy an invading fleet of 55 flying saucers while dodging the carpet of bombs they drop. Your bomb shelters will help you—for a while. Our version of a well known arcade game! Requires Applesoft in ROM.

Howitzer—This is a one or two person game in which you must fire upon another howitzer position. This program is written in HIGH-RESOLUTION graphics using different terrain and wind conditions each round to make this a demanding game. The difficulty level can be altered to suit the ability of the players. Requires Applesoft in ROM.

Space Wars—This program has three parts: (1) Two flying saucers meet in laser combat—for two players, (2) two saucers compete to see which can shoot out the most stars—for two players, and (3) one saucer shoots the stars in order to get a higher rank—for one player only. Requires Applesoft.

Golf—Whether you win or lose, you're bound to have fun on our 18 hole Apple golf course. Choose your club and your direction and hope to avoid the sandtraps. Losing too many strokes in the water hazards? You can always increase your handicap. Get off the tee and onto the green with Apple Golf. Requires Applesoft.

The minimum system requirement for this package is an Apple II or Apple II Plus computer with 32K of memory and one minidisk drive.

Order No. 0163AD $19.95

Instant Software™

*A trademark of Apple Computer Inc.

PETERBOROUGH, N.H. 0345
603-924-7296

Electronic Typing Program for the Apple

A minimal word processor in BASIC for the Apple II that edits one-line-at-a-time.

Thomas D. Brock
1227 Dartmouth Rd.
Madison, Wisconsin 53705

Although the Apple II was not really designed with word processing in mind, it is adaptable to a number of available word processing software packages. Some of these packages are not as sophisticated as office-oriented word processors, but several work very well.

However, all word processing packages for the Apple are fairly involved programs, and require not only a disk system but a large amount of memory. They do sophisticated file handling, formatting, line justification, and various editing functions. These features are fine for office-oriented or article-writing tasks, but if you're only interested in writing a letter, you don't need disk back-up copies or fancy formatting. You'd probably like to just sit down at your Apple, type the letter, then have it printed and ready to tear off and mail.

It was with this idea in mind that I wrote the Apple electronic typing program. This program lets you enter text a line-at-a-time, edit the line on the screen, and then print it when a carriage return is pressed. As the line is printed, the screen is cleared and another line can be typed in at the same time that the previous line is being printed. Thus, you don't have to wait for a print function. When the typing is finished, the letter is already printed and ready to be sent. Simple screen-oriented editing is permitted, but once you press the carriage return, the line starts going to the printer and can no longer be changed.

Although this problem originally motivated me to write this program, once I got into the programming details I discovered I was learning a lot about how some of the more sophisticated word processing packages operated. I decided to implement both forward and backward spacing for editing, word wrap (this is a feature that avoids breaking a word in the middle when typing reaches the end of the standard 40-character Apple screen; the whole word is moved down to the following line, making reading and proofing of text much easier), upper and lower case, tabbing, and single and double spacing. Although each of these features adds to the overhead of the program and slows it down, I thought they were useful and left them in. Most of the features can be easily deleted if they don't suit your needs.

This program was written in Integer BASIC because Applesoft was simply too slow to handle it. The procedure is to do all of the character display on the screen, by direct POKEs into screen memory. PRINT statements are used only to send text out to the printer. The character called by the keyboard is determined by PEEKing the keyboard memory location (-16384), which is the way in which the Applesoft GET function is handled in Integer BASIC. At the same time that the keyboard character is POKEd to the screen, it is POKEd to one of two alternating print buffers in memory. If a line is to be printed (as signalled by a carriage return), a flag is set, and the line is printed character-by-character until an end-of-line indicator is reached. The keyboard can interrupt the print routine at any time to direct a character to the next line forming on the screen, but another carriage return will not be recognized until the previous line is completely printed. A fast typist might be able to get ahead of the printer, but if you are composing a letter at the keyboard, as the program intends, then you are usually typing slowly enough so that keyboard interrupts do not interfere with the print function. (Under no conditions will a fast typist wipe out part or all of an unprinted line. If keyboard interrupts come too frequently during a print cycle, all that will result is that you will have to type more slowly and/or wait at the end of the second line until the first line is printed.)

The reason two print buffers are used alternately is because the print function looks for an end-of-line flag, which is always inserted in the location next to that one just specified by the keyboard. If only a single print buffer were used and you type too rapidly, the second line could overprint part of the first line and a new end-of-line flag inserted, thus prematurely terminating printing.

Margins are set in a simple and direct way. When the program is first run, with the print head at the full left side of the printer, the operator is asked to move the paper into the position desired for the left margin. Then, using the Apple keyboard, the user spaces across the page, watching the print head move across the printer until the desired right margin is reached, at which point a carriage return is sent, and the margins are set. The screen now goes blank and a cursor is positioned at the left end of one of the middle rows of the Apple screen. To signify the right margin on the screen, a vertical bar is inserted, usually down and to the right on the following line (unless very narrow margins of less than 40 characters are being used).

If word wrap moves a word to the second line, the vertical bar moves over, so that the vertical bar always indicates the true right margin, as it will appear on the printer. When the typist reaches a point seven spaces from the right margin, a bell will ring. It is possible to overtype the right margin that has been set, although this would not be desirable for any more than a few extra characters.

All of the characters typed at the keyboard will be displayed in normal video and will be printed in lower case on the printer. To obtain a single upper case character, it is preceded with an ESCAPE; it will then be displayed in inverse video, and subsequently printed upper case. To obtain a series of upper case characters, precede them with a "control-A." All subsequent characters will then be displayed in inverse video and printed as upper case until a "control-S" is typed.

While the system is printing, you'll notice that a line of mostly garbage unfolds at the top of the screen, except for the upper case characters, which will appear normally. The garbage arises because the Apple interprets ASCII characters in a different manner than the printer. As outlined in table 7, page 15, of the *Apple Reference Manual*, the character that will appear on the Apple screen can be either an upper case letter, a number, or a special character (such as a period, comma, or colon).

If the ASCII code used is less than 64, then the character will appear on the screen in inverse video. If the ASCII code used is between 64 and 127, then the character will appear on the screen as a flashing character. ASCII codes between 128 and 159 are control characters, but appear on the screen as normal video (if they are POKEd to the screen, but not if placed on the screen with a PRINT statement). ASCII codes from 160 to 223 will appear as normal video, whereas ASCII codes of 224 to 255 will appear on the screen as numbers or special characters.

As if it isn't bad enough having three separate screen codes for the same character, depending upon whether it is inverse, flashing, or normal, we must also remember that the ASCII code generated by the keyboard, (which we read at memory location −16384) is different from the ASCII code that the printer recognizes. From the keyboard, the high bit is set, so that the ASCII codes run from 128 to 255, whereas the printer recognizes the ASCII code without the high bit, so it requires codes from 1 to 127. Fortunately, all we need to do to convert the keyboard code to the printer code is to subtract 128.

Another problem arises at this point. If we are to know where we are on the screen, we need a cursor. Since we are doing everything with screen POKEs, a cursor is not automatically

```
   1 REM  APPLE ELECTRONIC TYPING PROGRAM
   2 REM   BY THOMAS D. BROCK
   3 REM
  10 DIM CHR$(126): FOR I=129 TO 255: POKE 1927+(I-1),I: NEXT I: POKE 2182
     ,30
  11 GOSUB 8000
  12 CALL -936: VTAB 13
  15 INPUT "SINGLE OR DOUBLE SPACE (1/2)",DS
  20 PR#PN
  30 CALL -936
  40 S=1320:S1=S:J=0:P=768:T1=768:AC=0
  45 J1=39:F1=0:K1=1
  50 B=0:FL=0
  80 POKE 34,24: POKE S,96: POKE TERM,219
  90 F=0
 100 UC=AC
 110 X= PEEK (-16384)
 120 IF X=129 THEN AC=32
 130 IF X=147 THEN AC=0
 140 IF X=129 OR X=147 THEN GOTO 100
 150 IF X=137 THEN GOTO 5000
 160 IF X=138 THEN GOTO 5500
 170 IF X=155 THEN UC=32
 180 IF X=155 THEN GOTO 110
 190 IF X=136 THEN GOTO 3000
 200 IF X=149 THEN GOTO 4000
 205 IF X=154 THEN GOTO 7000
 210 IF X>127 THEN GOTO 1000
 220 IF F=0 THEN GOTO 100
 230 A= PEEK (P1)
 240 IF A#255 THEN GOTO 300
 250 IF DS=2 THEN PRINT CHR$(10,10);
 255 GOTO 90
 300 A$=CHR$(A,A)
 310 PRINT A$;
 320 P1=P1+1
 330 GOTO 100
1000 POKE -16368,0
1010 X1=X-128
1020 IF X1>=64 THEN X1=X1+32-UC
1030 IF X>=192 THEN X=X-(UC*6)
1040 POKE S1,X
1050 POKE P,X1
1060 POKE P+1,255
1070 IF X=141 THEN GOTO 2000
1080 P=P+1
1090 J=J+1
1095 B=B+1
1100 IF J=39 THEN GOSUB 6000
1110 IF B=MARGIN-7 THEN PRINT CHR$(7,7);
1120 S1=S+J
1130 X= PEEK (S1)
1140 IF X>=192 THEN X=X-128
1150 IF X<192 AND X>=160 THEN X=X-64
1160 POKE S1,X
1170 GOTO 100
2000 IF F=1 THEN POKE S1,96
2002 IF F=1 THEN GOTO 100
2003 P1=T1
2004 UC=0
2005 B=0
2010 POKE 34,0
2020 CALL -936
2030 POKE 34,24
2040 S=1320:J=0:F=1:S1=S:FL=0
2045 J1=39
2048 T=F1:F1=K1:K1=T
2050 P=768+F1*100
2052 T1=P
2060 POKE S,96: POKE TERM,219
2070 GOTO 100
3000 POKE -16368,0
3005 X= PEEK (S1)
3010 IF X<=127 AND X>=96 THEN X=X+64
3020 IF X>=64 AND X<=95 THEN X=X+128-(3*FL)
3030 POKE S1,X
3040 J=J-1
3045 P=P-1
3047 B=B-1
3048 FL=0
3050 IF J=127 THEN J=J1
3060 IF J<0 THEN J=0
3070 S1=S+J
3080 X= PEEK (S1)
3090 IF X>=192 THEN X=X-128
3100 IF X<192 AND X>=160 THEN X=X-64
3105 IF X<=63 THEN FL=64
3110 POKE S1,X+FL
3120 GOTO 100
4000 POKE -16368,0
4005 X= PEEK (S1)
4007 T=X
```

(Continued)

```
4010 IF X<=127 AND X>=96 THEN X=X+64
4020 IF X>=64 AND X<=95 THEN X=X+128-(3*FL)
4025 POKE S1,X
4030 IF FL=0 AND T<=95 THEN LC=32
4032 IF T<=95 THEN X1=T+LC
4034 IF T<=127 AND T>=96 THEN X1=T-64
4035 POKE P,X1
4037 LC=0
4040 J=J+1
4045 P=P+1
4047 B=B+1
4048 FL=0
4050 IF J=J1+1 THEN J=128
4060 IF J>TERM THEN J=TERM
4070 S1=S+J
4080 X= PEEK (S1)
4090 IF X>=192 THEN X=X-128
4100 IF X<192 AND X>=160 THEN X=X-64
4105 IF X<=63 THEN FL=64
4110 POKE S1,X+FL
4120 GOTO 100
5000 POKE -16368,0
5005 FOR I=1 TO 5
5010 POKE S1,160
5020 POKE P,32
5030 J=J+1
5040 IF J=40 THEN J=128
5050 P=P+1
5055 B=B+1
5060 S1=S+J
5070 NEXT I
5075 POKE S1,96
5080 POKE P+1,255
5090 GOTO 100
5500 POKE -16368,0
5505 FOR I=1 TO 30
5510 POKE S1,160
5520 POKE P,32
5530 J=J+1
5540 IF J=40 THEN J=128
5550 P=P+1
5555 B=B+1
5560 S1=S+J
5570 NEXT I
5575 POKE S1,96
5580 POKE P+1,255
5590 GOTO 100
6000 TEMP=TERM
6002 J1=J
6005 X= PEEK (S1)
6010 IF X=160 OR X=96 THEN GOTO 6100
6020 R=R+1
6030 S1=S1-1
6040 GOTO 6000
6100 J=128
6110 I=0
6112 IF I=R THEN GOTO 6162
6113 I=I+1
6115 S1=S1+1
6120 X= PEEK (S1)
6130 POKE S1,160
6140 POKE S+J,X
6150 J=J+1
6160 GOTO 6112
6162 POKE TEMP,160
6165 TEMP=TEMP+R
6170 POKE TEMP,219
6175 J1=J1-R-1
6180 R=0
6190 S1=S+J
6200 RETURN
7000 PR#0
7010 POKE 34,0
7020 CALL -936
7030 VTAB 10
7040 PRINT "YOU WILL HAVE TO RECONNECT          DOS BY TYPING 'PR #0'"

7050 END
8000 CALL -936: VTAB 10
8001 INPUT "WHAT SLOT FOR PRINTER",PN
8003 MARGIN=60
8005 INPUT "DO YOU WANT TO SET          MARGINS(Y/N)",Y$
8010 IF Y$#"Y" THEN TERM=1468
8015 IF Y$#"Y" THEN RETURN
8017 PR#PN: PRINT CHR$(13,13);: PR#0
8018 VTAB 10
8020 PRINT "ADJUST PRINT HEAD AND PAPER          FOR LEFT MARGIN"
8030 PRINT "THEN SPACE ACROSS TO RIGHT MARGIN"
8040 PRINT "YOU MAY ALSO BACKSPACE"
8041 PRINT "WHEN YOU HAVE PROPER RIGHT          MARGIN, PRESS RETURN"
```

(Continued)

generated and we must provide one. The procedure here is to read the character next to the one we have just inserted on the screen and convert it to flashing. This is done by PEEKing at the location just after the one we have POKEd, adjusting its value appropriately to make it flash, and POKEing it back where we found it. Once we are able to adjust our ASCII codes properly, most of the rest of the programming is relatively straightforward, although some complications arise from the word wrap, backspace, and forward space arrows. (The details of the program will be given later.)

When it is all finished, the program seems surprisingly complicated for what it does. Is it worth it? I have found the program quite useful for typing routine letters that I did not need to save to disk, or did not anticipate editing. Since the format to be printed is seen on the printer before it is used, it is simple to adjust margins for narrow printing jobs, such as envelopes, labels, and file cards. Perhaps the most useful thing about the program is that it forces you to understand how the Apple keyboard and screen function. It also illustrates the principle of how you can have the computer do two different tasks (typing and printing) at the same time.

The next step in making this program more useful is to convert it to machine language so that it will run faster and thus not slow down a fast typist. This is left as an exercise for the reader!

Program

Variables Used

S = screen start position; memory location 1320 (mid-screen).

S1 = screen cursor position; initialized to S.

J = counter for screen column position.

J1 = end-of-screen column position = 39.

P = print buffer initial position = hex 300 or decimal 768 (alternate print buffer position is hex 364 or decimal 868).

T1 = temporary print buffer location (for alternating print buffer routine).

UC = upper case flag; initialized to zero and set to 32 when "Escape" pressed.

AC = all caps flag; initialized to zero and set to 32 when all caps called by "control-A"; reset to zero when "all caps" terminated by "control-S".

```
8045 PR#PN
8047 MARGIN=0
8050 X= PEEK (-16384)
8055 IF X=141 THEN GOTO 8400
8060 IF X=160 THEN GOTO 8200
8070 IF X=136 THEN GOTO 8300
8080 GOTO 8050
8200 POKE -16368,0:A$=CHR$(32,32)
8210 MARGIN=MARGIN-1
8220 PRINT A$;
8230 GOTO 8050
8300 POKE -16368,0:A$=CHR$(8,8)
8310 MARGIN=MARGIN-1
8320 PRINT A$;
8330 GOTO 8050
8400 POKE -16368,0
8405 IF MARGIN<40 THEN GOTO 8440
8407 TERM=1448+(MARGIN-40)
8410 PRINT CHR$(13,13);
8420 PR#0
8430 RETURN
8440 TERM=1320+MARGIN
8450 PRINT CHR$(13,13);: PR#0: RETURN
```

F1 = flag for use in alternating print buffer routine; set alternately to 0 or 1 at each pass through the print routine.

K1 = flag working opposite F1; set to 0 when F1 set to 1 and vice-versa.

B = bell counter for margin.

FL = flag to indicate character picked from screen by forward or backspace is upper case (inverse video); set to either 0 or 64.

LC = lower case flag for forward space routine, for making character lower case for the printer.

F = print flag; if set to 1 then a line is being printed; reset to zero when printing of line is finished (end-of-line flag is reached).

T = temporary variable for switch routines.

DS = double/single space flag; set to 1 for single-space and 2 for double-space.

P1 = print buffer current position; location in print buffer where next character will be POKEd.

R = counter for word-wrap.

TERM = terminus of printer line as marked on screen; set to printer line length of 60 characters by default; set to selected right margin by subroutine 8000.

MARGIN = length of line counter; set by subroutine 8000.

I = general index counter for tab and word-wrap functions.

Keyboard and Screen Codes Used

96 = flashing space on screen; cursor for next character to be placed on screen.

129 = control – A; indicates to start all caps; sets AC to 32 until a control – S is typed.

136 = control – H; backspace arrow.

137 = control – I; tab 5 spaces.

138 = control – J; tab 30 spaces.

141 = control – M; carriage return.

147 = control – S; end all caps; set AC to 0.

149 = control – U; forward space arrow.

154 = control – Z; quit program.

155 = Escape; next character is upper case; sets UC to 32 for the next character only.

219 = ASCII screen code for vertical bar.

255 = Hex FF; end-of-line flag for print buffer.

Routines and Subroutines

Line 10: CHR$ function in Integer BASIC.

Lines 11-80: initialization of variables.

Lines 100-300: read keyboard and print line routines; if a line is being printed, the keyboard may interrupt.

Line 110: read keyboard character.

Lines 120-200: check for keyboard control character.

Line 210: check to see if keyboard has been pressed.

Line 220: check to see if print flag (F) has been set, if not loop and read keyboard again.

Lines 230-330: print routine; Line 240 checks for end-of-line flag (Hex FF or decimal 255).

Line 1000: clear keyboard strobe.

Lines 1000-1170: screen and print buffer business; adjust character for upper or lower case, POKE to screen and print buffer, advance counters, check for margin and ring bell, loop to read keyboard for next character.

Lines 2000-2070: printer business; sets print flag (F) to 1, changes print buffer, clears screen, resets cursor, resets end-of-line signal.

Lines 3000-3120: Backspace functior (back arrow on keyboard); reads screer position at cursor and changes fron flashing to normal or inverse, backs up reads screen position backed up to checks to see if character is invers video (= cap) and sets FL to indicate changes character picked up from nor mal or inverse to flashing, returns t keyboard.

Lines 4000-4120: Forward space func tion (forward arrow on keyboard); read screen character, saves it for print bu fer in T, changes from flashing to no mal or inverse, converts to prope ASCII and POKEs into print buffe moves forward (will not forward spac past end-of-line set by Margin), set next character to flashing and sets ir verse video flag (FL).

Lines 5000-5590: Tab 5 function; FOR-NEXT loop; puts normal space (ASCII 160) on screen and norm spaces (ASCII 32) in print buffer for th next 5 spaces.

Lines 5500-5590: Tab 30 spaces.

Lines 6000-6190: Word-wrap functio If end-of-line reached (J = 39) on scree then GOSUB 6000. Checks for wheth character at cursor position is a spa (ASCII 160 or 96). If not, backs up unt it finds a space, counting the number positions backed up with R. When finds a space it sets the screen positi for output to the next line (with S + J then moves forward on the previo line (with S1), picks up each charact and transfers it to the next line. Clea the end-of-line signal (vertical b from its initial location and moves right the number of spaces printed the 2nd line. Resets S1 to the next fr screen location and returns.

Lines 7000-7050: program terminati routine; clears screen, reminds us that DOS must be reinitialized fro the keyboard, and quits.

Lines 8000-8450: Sets printer slot a margin.

Special Functions

X = PEEK (– 16384) reads the keyboa as the code of the key pressed is stor in memory location – 16384.

POKE – 16368,0 clears the keybo strobe. This must be done each ti after the keyboard is read.

IF x > 127 : If a key is pressed, t value at the keyboard memory locati will be greater than 127 (high bit is se

AICR

A Typewriter Bell for Your Microcomputer

This hardware and software combination sounds an alarm when you near the end of a BASIC input line. The hardware can also be used to improve game programs.

Charles L. Stanford
2903 Georgetown Road
Cinnaminson, New Jersey 08077

A wordprocessor, or even a simple screen editor, can be a great aid in writing articles and formatting text or graphics printouts. But the lack of any audible indication of line end can cause many delays while letters or words are moved down to the next line, or hyphenated. Even programming in BASIC can be substantially improved by a "bell." For example, I like to cram as much as possible into each DATA statement line. So it's a real pain when I run over the 72 character limit of the buffer, and have to redo the whole line.

Luckily, Microsoft made it easy to program a line position detector, by putting vectors and flags in the first three pages of RAM on most of their programs. Memory maps of PET, Apple, Atari, OSI, and several others indicate the presence of a "line buffer pointer." Its location varies, but it is usually pretty low in page zero. On the OSI, location $000E holds the pointer to the next open character space in the line buffer, which happens to start at $0013. Thus, a tool is available to check your current location while entering data, or printing to the screen. But how do we access this information and put it to use?

BASIC uses a routine located in the monitor ROM at $FFBA to input a character, whether from program memory, the keyboard, or the ACIA. While most such routines and subroutines are either not accessible, or must be reached by the USR function, this particular one (along with a few others) is reached by BASIC via an indirect jump through RAM at $0218. So, it's no real trick to "intercept" the routine and use it for our bell. The BASIC routine shown in listing 1 does just that.

Listing 1 shows a program which will POKE a machine language program into free RAM at the top of page zero. Please note that while this RAM is not used by BASIC, it is used by the monitor, so a break and warm start will require that the vectors in line 40 be reset, and a break to the monitor will require that the entire program be re-entered. Otherwise, once the program has been run, NEW can be typed and the computer is available for normal use.

Listing 2 shows the actual machine language program. By changing the vectors as we do in line 40 of listing 1, the BASIC routine jumps to $00D8 instead of to $FFBA. That, of course, has to be done at some point, but we can use the time for our own purposes. First, the value of the data at location $000E is loaded into the accumulator, and compared with the desired location for the bell to ring. This can be changed as you desire; it is set as shown to ring at the 64th of the 72 characters. Next (and this is optional) a solid square is POKEd to the screen at the exact location of the 73rd character, to give a good visual indication of the end of the line. I have found this to be particularly useful for BASIC programming, so that the line can use every character possible.

Finally, we ring the bell. This is done by setting two of the keyboard rows located in memory location $DF00 to low. (Actually, while only two rows need to go low, I just set all eight to zero. This triggers a small oscillator which will be described shortly.) The lines stay low for only a few microseconds, until the keyboard scan routine takes over and sets all but one at a time back to high. Thus, you get a visual and an audible warning when nearing the end of the line. It is also possible to trigger the bell by monitoring the cursor location at $0200, but then the C1P owner will get a sound three times for each line, due to the 24 character screen width.

The C2 user can make the change easily. Other variations, such as PEEKing the screen to see if the scanned location has a blank or a character, suggest themselves. As my screen editor is for a modified C1P with 64 characters, and is written in machine language, I use a variation of this method. With the cursor travelling from the upper left corner of the screen, it is necessary to AND the low byte of its location with #$3F to get only the location in the line, rather than the location in the page.

Circuit Description

The bell itself is a model of simplicity. Only two chips are required, and both are readily available at Radio Shack or similar stores. What we're doing is using the keyboard as an output port. The problem is that the keyboard scan routine in the monitor also uses it as both an output and an input port, and continually switches the rows, and then checks the columns for a key closure. The trick here is to use a combination of rows, which the scan routine does not do. Some programs must, as I get an

Listing 1

```
10  REM --BELL & MARK FOR 24 CHR OSI C1P
20  REM --C.L. STANFORD
30  REM
40  FOR X = 216 TO 235: READ D: POKE X,D: NEXT
50  POKE 536,216: POKE 537,0
60  DATA 169,64,197,14,208,10,169,161,141,124
70  DATA 211,169,0,141,0,223,32,186,255,96
```

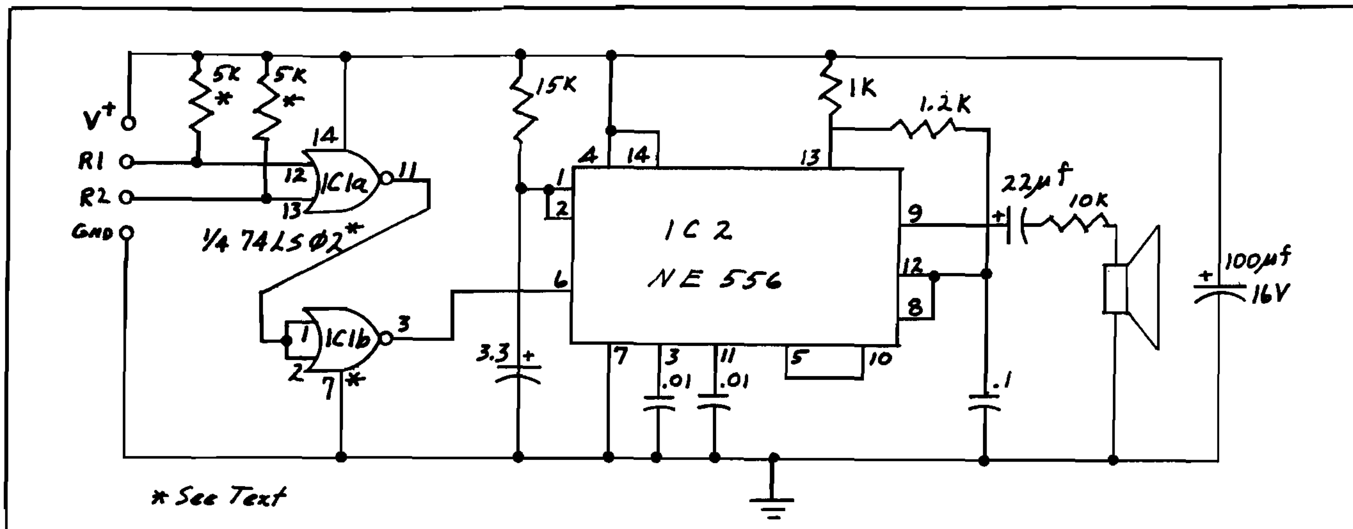


Figure 1: Schematic

occasional odd ring. But this is very seldom, and never occurs in such a way as to interfere with its main function.

The detector IC is a quad dual input NOR gate, and two of the four gates are used. The first will go high only when *both* inputs are low. Otherwise, its output remains low. The second is wired as an invertor to condition the signal for the oscillator. That is an NE556 (the dual 555 timer). Of course, two 555's can be used just as well, but I wanted to reduce package count to save space. The front half of the 556 is wired as a monostable multi-vibrator, and the R/C combination used gives a tone duration of about 1/5 of a second. The second half of the 556 is on only while the output of the first is high. It is wired as an astable multi-vibrator with a frequency of about 1KHz. Its output is wired directly to a small speaker through an electrolytic capacitor and a low-value resistor. The result is a sharp high-pitched "beep" whenever the keyboard rows go low.

Building the Bell Circuit

Generally, wire wrap is best for a project of this size, although the Radio Shack dual IC prototype board can be used if a large enough case is selected. Also, the speaker size will dictate other dimensions to a certain degree. In other words, select components which will fit into your box! You can use either a 74LS02 for IC1 as shown, or a CMOS CD4001AE. If the CMOS chip is chosen, change the 5K pullup resistors to 100K, and be sure to connect unused inputs 5, 6, 8, and 9 to ground. Otherwise, both will work fine, and the CMOS design will use a fraction of the power of the LS chip. None of the components is critical, and substitutions can be made within reason. Increasing the value of either the resistor or capacitor associated with pins 1 and 2 of IC2 will result in a longer tone. Increasing those connected to pins 8, 12, and 13 will result in a lower pitch.

Drill your case for a four-conductor cable, and cut one to a suitable length. The connector can be any of several, depending on the configuration of your computer. Superboard owners can just use a Molex pin plug. C1P's need a bit more sophistication. I had previously brought all the rows and columns to the front of my C1P on a DB25 (RS-232) connector, so it was easy. A very good plug and socket available everywhere is the European DIN series. Mount the socket carefully on either the front or rear panels of your computer, and connect to the main board at jack J4. Pins 1, 2, and 10 have rows 1, 7, and 6 respectively; pick any two. You will have to connect an additional wire to +5 volts at any convenient location on the board. There is a good ground location near the jack.

Other Applications

Shortly after building this add-on circuit, I found a pretty nice Breakout game written in BASIC for the C1P in a magazine. Adding the bell was simple! The program tested for the paddle, walls, etc., with IF...THEN statements. I just keyed "POKE 57088,0" within each dependent statement line, and now the "bell" rings every time the puck hits any obstruction. The bell does not retrigger, as Control/C is not disabled, and the keyboard scan is thus in continuous operation. If Control/C is disabled, a "POKE 57088,255" will be required to turn off the bell.

There is absolutely no reason this circuit cannot be connected to a port on just about any computer. It will, of course, be a lot harder to control if the BASIC interpreter does not have Microsoft's vector format, but this little bit of hardware eliminates the need to program the port to make the tone in real time; just POKE it on, POKE it off, and resume the program.

MICRO

Charles L. Stanford is a Civil Engineer, has a PE license, and manages the Facilities Department of Philadelphia's transit system. He got into microcomputing as a hobby from the hardware side, designing toys and games with chips, and bought a C1P about two years ago. He has been "redesigning" both the hardware and software ever since.

Listing 2

```
                      ;*
                      ;* BELL RINGER
                      ;*
                      LINLEN EPZ $0E
                      GETCHR EQU $FEBA
                      ;
                             ORG $D8
                             OBJ $800
                      ;
00D8 A940                    LDA #$40      ;LINE LENGTH
00DA C50E                    CMP LINLEN    ;CHECK IT
00DC D00A                    BNE END
00DE A9A1                    LDA #$A1      ;PUT A SQUARE ON
00E0 8D7CD3                  STA $D37C     ;SCREEN AT LINE END
00E3 A900                    LDA #$00      ;RING THE BELL
00E5 8D00DF                  STA $DF00
00E8 20BAFF           END    JSR GETCHR    ;GET A CHARACTER
00EB 60                      RTS
```

Monobyte Checksum Dumper for C1P

This two page machine language dump/load utility provides fast tape I/O and checksum protection.

Peter D.H. Broers
Overijsselstr.9
5144 EH WAALWIJK
The Netherlands

This routine saves programs or data to tape and uses $1E00-1FFF. When relocated, locations 1E4F (1F) and 1E54 (00) have to be replaced by the high/low bytes of the LOADER-start location ($1F00 here).

The routine is entered at $1E00 (.1E00G in monitor) and prompts

CHECKSUM DUMPER

FRST/LAST/AUTO ?
(first location, *last + 1* and autostart)

waiting for 12 valid hex digits to be typed in, (no corrections, sorry); next it prompts

START RECORDER

waiting for a carriage return from the keyboard.

It then dumps a loader (1F00-1FFF) and next the program or data in blocks of 256 bytes. The last block may be shorter. The format is:

CR,	ten zeroes, line feed (the carriage return is neglected)
;	identifier of a block of data
0240	four bytes (hex address, in ASCII)

Listing 1

```
                    ;********************************
                    ;*                              *
                    ;* SINGLE BYTE CHECKSUM DUMPER *
                    ;*                              *
                    ;*       BY PETER BROERS        *
                    ;*                              *
                    ;********************************
                    ;*
                    ;* DUMPER PART
                    ;*
                    BYTIN  EQU $FFEB           ;GET BYTE FROM TAPE OR KEYBOARD
                    BYTOUT EQU $FFEE           ;DISPLAY (AND SAVE) A BYTE
                    SAVBYT EQU $FCB1           ;SAVE BYTE WITHOUT DISPLAY
                    ;
                    ADRES  EPZ $E0             ;AUTOSTART LOCATION
                    END    EPZ ADRES+2         ;LAST LOCATION TO BE DUMPED
                    PNTR   EPZ ADRES+4         ;FIRST LOCATION, CURRENT POINTER
                    CHCK   EPZ ADRES+6         ;CHECKSUM (TWO BYTES)
                    CNTR   EPZ ADRES+8         ;COUNTER (ONE BYTE)
                    ;
                    LOADER EQU $1F00
                    ADCHCK EQU LOADER+$69      ;ADD BYTE TO CHECKSUM SUBR
                    ADRIN  EQU LOADER+$73      ;GET ADDR IN HEX SUBROUTINE
                    PRMPTS EQU LOADER+$96      ;PRINT MESSAGES SUBROUTINE
                    ;
1E00                       ORG $1E00
1E00                       OBJ $800
1E00                ;
1E00 A900           RESET  LDA #$00
1E02 850D                  STA $0D             ;NO NULLS
1E04 A202                  LDX #$02            ;PRINT "DUMP B/M" (BASIC OR MACHINE)
1E06 20961F                JSR PRMPTS
1E09 2000FD                JSR $FD00           ;GET KEY
1E0C C942                  CMP 'B              ;IF KEY IS "B" THEN BASIC
1E0E D03A                  BNE MACHIN          ;ELSE MACHINE LANGUAGE PROGRAM OR DUMP
1E10                ;
1E10 A204           BASIC  LDX #$04            ;PRINT "READY ?"
1E12 20961F                JSR PRMPTS
1E15 2000FD                JSR $FD00           ;GET KEY
1E18 C959                  CMP 'Y              ;IF KEY IS "Y" THEN PROCEED
1E1A D0F4                  BNE BASIC           ;ELSE REDO PROMPT "READY?"
1E1C 20F7FF                JSR $FFF7           ;SAVE
1E1F A207                  LDX #$07            ;PRINT ".0079/"; (BASIC POINTERS START)
1E21 20961F                JSR PRMPTS
1E24 A200                  LDX #$00
1E26                ;
1E26                ;$79,7A START-OF-BASIC
1E26                ;$7B,7C END-OF-BASIC
1E26 B579           LOOPA  LDA $79,X           ;SAVE POINTERS IN MONITOR LOADABLE FORM
1E28 20DF1E                JSR MONOUT
1E2B E8                    INX
1E2C C904                  CMP #$04
1E2E D0F6                  BNE LOOPA
1E30 A579                  LDA $79             ;SET START POINTER TO DUMP THE CONTENTS
1E32 A47A                  LDY $7A             ;OF THE BASIC START POINTER
1E34 85E4                  STA PNTR
1E36 84E5                  STY PNTR+1
1E38 A57B                  LDA $7B             ;SET END PNTR OF DUMP TO CONTENTS OF
1E3A A47C                  LDY $7C             ;THE BASIC END-OF-PROG POINTER
1E3C 85E2                  STA END
1E3E 84E3                  STY END+1
1E40 A974                  LDA #$74            ;SET AUTOSTART ADDRESS TO $A274
1E42 A0A2                  LDY #$A2            ;(BASIC WARM START)
1E44 85E0                  STA ADRES
1E46 84E1                  STY ADRES+1
1E48 D019                  BNE DMPLOD          ;JUMP TO "DUMP LOADER"
```

(Continued)

0 counter (for a full block, or less, for a shorter block (binary byte)

DATA (up to 256 bytes, no ASCII, no masked off bits: full binary)

L a binary byte giving the checksum low

H a binary byte giving the checksum high

The checksum is the binary sum of all the data bytes in the block; the "household bytes" such as the CR, zeroes and LF, identifier, address and counter and the checksum itself, are not included in the count.

After the last block, comes the autostart: "$1300." When loaded, the loader starts itself, and after the checksum load is completed, the machine goes to the autostart location, which may be the entry point of the routine or any other location.

At 300 bauds, the loader takes about 30 seconds to come in, and 10 seconds for any page. (My 4.5K assembler loads in about 3½ minutes.) The MONITOR "L" format (hex + carriage return) takes about 9 minutes, and the hex-checksum format (OSI standard?) about the same time. There should be no problems at 600 baud or more, as long as the cassette supports the higher baud rate.

The program might be shortened to fit within one page if one does not use the checksum control. I tried a "monobyte dumper" without a checksum, and no blocks. The whole program dumped one byte after the other, and it worked all right. However, the time one wins by this fastest possible dump is very little, as this checksum dump takes only 20 household bytes per page.

Peter Broers is a grammar school teacher of French, and a member of the Dutch province of Brabant Superboard Users Group BRABOSI. He is trying to introduce a small computer in the school for computer class and administrational services. His main interest lies in system programs.

Listing 1 *(Continued)*

```
1E4A              ;
1E4A A203         MACHIN LDX #$03        ;PRINT"FIRST/LAST/AUTO?"
1E4C 20961F              JSR PRMPTS
1E4F A005                LDY #$05        ;GET 6 HEX (2 DIGITS EACH) ADDRESSES
1E51 20731F              JSR ADRIN       ;AND STORE THEM IN ADRES/END/POINTER
1E54 A204                LDX #$04
1E56 20961F              JSR PRMPTS      ;PRINT "READY?"
1E59 2000FD              JSR $FD00       ;GET KEY
1E5C C959                CMP 'Y          ;IF KEY IS "Y" THEN PROCEED
1E5E D0EA                BNE MACHIN      ;ELSE REDO PROMPT "FIRST/LAST/AUTO?"
1E60 20F7FF              JSR $FFF7       ;SAVE
1E63              ;
1E63 A205         DMPLOD LDX #$05        ;DUMP THE LOADER IN "MONITOR LOADABLE"
1E65 20961F              JSR PRMPTS      ;FORMAT, PRINTING LOADER START ADDRESS
1E68 A200                LDX #$00        ;(".1F00/" AS SUPPLIED HERE)
1E6A              ;
1E6A BD001F       LOOPB  LDA LOADER,X    ;AND 256 BYTES AS 2 HEX DIGITS,
1E6D 20DF1E              JSR MONOUT      ;PLUS CARRIAGE RETURN
1E70 E8                  INX
1E71 D0F7                BNE LOOPB
1E73 A206                LDX #$06        ;PRINT THE LOADER SELF-START ADDRESS
1E75 20961F              JSR PRMPTS      ;(".1F00G", AS SUPPLIED HERE)
1E78              ;
1E78 A900         CHDUMP LDA #$00        ;RESET THE COUNTER TO ZERO
1E7A 85E8                STA CNTR
1E7C 38                  SEC             ;CALC NUMBER OF BYTES STILL TO
1E7D A5E2                LDA END         ;BE DONE, USING CHECKSUM LOW REGISTER
1E7F E5E4                SBC PNTR        ;TO STORE THE LOW RESULT TEMPORARILY.
1E81 85E6                STA CHCK
1E83 A5E3                LDA END+1       ;CALCULATE THE NUMBER OF PAGES
1E85 E5E5                SBC PNTR+1
1E87 3041                BMI OFF         ;IF OVER $7F, THEN READY (NEGATIVE!)
1E89 D006                BNE BLOCK       ;IF NOT ZERO, THEN MORE WHOLE PAGES
1E8B A5E6                LDA CHCK        ;IF ZERO, THEN RESET COUNTER TO LOW
1E8D 85E8                STA CNTR        ;RESULT (POSSIBLY LESS THAN 256)
1E8F F039                BEQ OFF         ;IF LOW RESULT ZERO, THEN READY & OFF
1E91              ;
1E91 206CA8       BLOCK  JSR $A86C       ;PRINT CR, 10 ZEROES AND LF
1E94 A93B                LDA ';          ;PRINT BLOCK IDENTIFIER
1E96 20EEFF              JSR BYTOUT
1E99 A5E5                LDA PNTR+1      ;SAVE BLOCK ADDR IN HEX FORMAT
1E9B 20E71E              JSR HEXOUT
1E9E A5E4                LDA PNTR
1EA0 20E71E              JSR HEXOUT
1EA3 A5E8                LDA CNTR        ;SAVE THE COUNTER IN BINARY
1EA5 20B1FC              JSR SAVBYT
1EA8 A000                LDY #$00        ;RESET THE CHECKSUM TO ZERO
1EAA 84E6                STY CHCK
1EAC 84E7                STY CHCK+1
1EAE              ;
1EAE B1E4         LOOPC  LDA (PNTR),Y    ;SAVE THE BLOCK BYTE BY BYTE
1EB0 20B1FC              JSR SAVBYT
1EB3 20691F              JSR ADCHCK      ;ADDING IT TO THE CHECKSUM
1EB6 C8                  INY
1EB7 C4E8                CPY CNTR        ;IF BLOCK DONE,
1EB9 D0F3                BNE LOOPC
1EBB A5E6                LDA CHCK        ;THEN SAVE THE CHECKSUM IN BINARY,
1EBD 20B1FC              JSR SAVBYT      ;LOW FIRST, HIGH NEXT
1EC0 A5E7                LDA CHCK+1
1EC2 20B1FC              JSR SAVBYT
1EC5 E6E5                INC PNTR+1      ;NEXT PAGE
1EC7 4C781E              JMP CHDUMP      ;REDO THE WHOLE THING
1ECA              ;
1ECA 206CA8       OFF    JSR $A86C       ;PRINT CR, 10 ZEROES, AND LF
1ECD A924                LDA '$          ;PRINT THE AUTOSTART IDENTIFIER "$"
1ECF 20EEFF              JSR BYTOUT
1ED2 A5E1                LDA ADRES+1     ;PRINT THE AUTOSTART ADDRES IN HEX
1ED4 20E71E              JSR HEXOUT
1ED7 A5E0                LDA ADRES
1ED9 20E71E              JSR HEXOUT
1EDC 4C00FE              JMP $FE00       ;AND GO TO MONITOR OR ANY LOCATION
1EDF              ;
1EDF 20E71E       MONOUT JSR HEXOUT      ;SUBROUTINE TO DUMP A BYTE AS
```

Listing 2

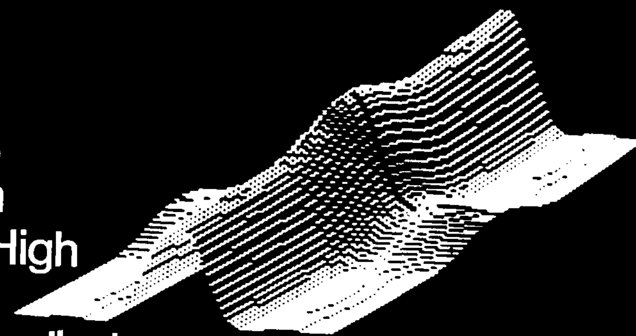
```
0800              ;*
0800              ;* SINGLE-BYTE CHECKSUM DUMPER
0800              ;*
0800              ;* LOADER PART
0800              ;*
0800              BYTIN  EQU $FFEB           ;GET BYTE FROM TAPE OR KEYBD
0800              BYTOUT EQU $FFEE           ;DISPLAY (AND SAVE) BYTE
0800              ADRES  EPZ $E0             ;CURRENT LOCATION
0800              END    EPZ ADRES+2         ;(NOT USED IN LOADER)
0800              PNTR   EPZ ADRES+4         ;(NOT USED IN LOADER)
0800              CHCK   EPZ ADRES+6         ;CHECKSUM
0800              CNTR   EPZ ADRES+8         ;COUNTER--NO. BYTES IN A BLOCK
0800              ;
1F00                     ORG $1F00
1F00                     OBJ $800
1F00              ;
1F00 20F4FF       LOADER JSR $FFF4           ;LOAD
1F03 A20A         LDBLCK LDX #$0A
1F05 20EBFF       ZEROIN JSR BYTIN           ;WAIT FOR 10 ZEROES TO COME IN
1F08 D0F9                BNE LDBLCK
1F0A CA                  DEX
1F0B D0F8                BNE ZEROIN
1F0D              ;
1F0D 20EBFF       LINEFD JSR BYTIN           ;WAIT FOR LINE FEED TO COME IN
1F10 C90A                CMP #$0A
1F12 D0F9                BNE LINEFD
1F14 20E0A8              JSR $A8E0           ;AND DISPLAY A SPACE
1F17              ;
1F17 20EBFF       IDENT  JSR BYTIN           ;WAIT FOR AN IDENTIFIER BYTE
1F1A C924                CMP '$              ;IF IT IS "$" THEN AUTOSTART
1F1C F03D                BEQ AUTOST
1F1E C93B                CMP ';              ;IF IT IS ";" THEN LOAD A BLOCK
1F20 D0F5                BNE IDENT           ;ELSE WAIT
1F22              ;
1F22 A001         ADDR   LDY #$01            ;WAIT FOR 2 HEX BYTES (4 DIGITS)
1F24 20731F              JSR ADRIN           ;(HIGH FIRST, LOW NEXT), STORE IN "ADRES"
1F27              ;
1F27 20EBFF       CNTRIN JSR BYTIN           ;GET COUNTER FROM TAPE
1F2A 85E8                STA CNTR
1F2C A000                LDY #$00            ;RESET THE CHECKSUM TO ZERO
1F2E 84E6                STY CHCK
1F30 84E7                STY CHCK+1
1F32              ;
1F32 20EBFF       MAINLP JSR BYTIN           ;MAIN LOOP: HAVE A BYTE FROM TAPE
1F35 91E0                STA (ADRES),Y       ;AND STORE TO CURRENT LOCATION
1F37 20691F              JSR ADCHCK          ;ADDING IT TO THE CHECKSUM
1F3A C8                  INY
1F3B C4E8                CPY CNTR            ;IF BLOCK DONE
1F3D D0F3                BNE MAINLP
1F3F 20EBFF       CHECK  JSR BYTIN           ;GET THE CHECKSUM FROM TAPE
1F42 C5E6                CMP CHCK            ;LOW FIRST, COMPARE IT WITH THE CALC
1F44 D007                BNE ERROR           ;CHECKSUM DURING LOAD, IF <>,
1F46 20EBFF              JSR BYTIN           ;THEN ERROR MESSAGE
1F49 C5E7                CMP CHCK+1
1F4B F0B6                BEQ LDBLCK          ;IF =, THEN NEXT BLOCK
1F4D              ;
1F4D A201         ERROR  LDX #$01            ;PRINT ERROR MESSAGE "ERROR<<HIT G"
1F4F 20961F              JSR PRMPTS
1F52              ;
1F52 2000FD       WAITG  JSR $FD00           ;WAIT FOR "G" (TIME TO REWIND)
1F55 C947                CMP 'G
1F57 D0F9                BNE WAITG
1F59 F0A8                BEQ LDBLCK          ;AND LOAD NEXT BLOCK
1F5B              ;
1F5B 20EEFF       AUTOST JSR BYTOUT          ;AUTOSTART: DISPLAY "$"
1F5E A001                LDY #$01            ;GET AUTOSTART ADDR FROM TAPE
1F60 20731F              JSR ADRIN           ;(TWO BYTES AS 4 HEX DIGITS)
1F63 EE0302              INC $203            ;CLEAR THE LOAD FLAG
1F66 6CE000              JMP (ADRES)
1F69              ;
1F69 18           ADCHCK CLC                 ;ADD THE BYTE TO THE CHECKSUM
```

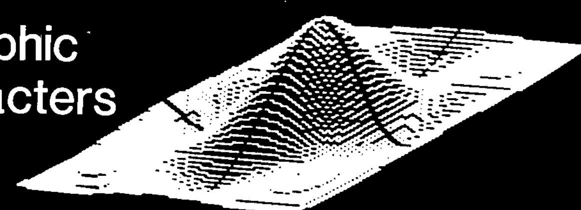
(Continued)

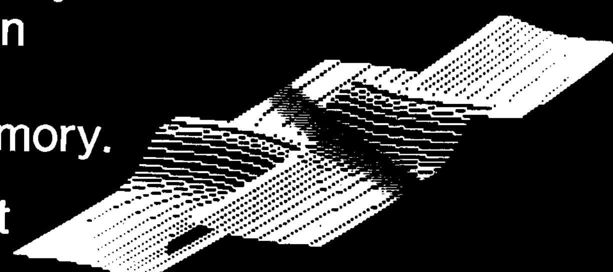
Listing 2 *(Continued)*

```
1F80 19E000          ORA ADRES,Y
1F83 99E000          STA ADRES,Y
1F86 88              DEY
1F87 10EA            BPL ADRIN          ;REDO FOR Y+1 BYTES
1F89 60              RTS
1F8A          ;
1F8A 20EBFF   DIGIN  JSR BYTIN          ;GET ONE HEX DIGIT
1F8D 20EEFF          JSR BYTOUT         ;DISPLAY IT
1F90 2093FE          JSR $FE93          ;TEST IT FOR VALID HEX AND MAKE BINARY
1F93 30F5            BMI DIGIN          ;0-15. IF NOT VALID, REDO.
1F95 60              RTS
1F96          ;
1F96 A0FF     PRMPTS LDY #$FF           ;MESSAGE PRINTER "PROMPTS"
1F98 C8       PLOOPA INY                ;FIND MESSAGE NR. X
1F99 B9AE1F          LDA MESSAG,Y
1F9C D0FA            BNE PLOOPA
1F9E CA              DEX
1F9F D0F7            BNE PLOOPA
1FA1 C8       PLOOPB INY                ;AND PRINT (& SAVE?)
1FA2 B9AE1F          LDA MESSAG,Y
1FA5 F006            BEQ RETURN
1FA7 20EEFF          JSR BYTOUT
1FAA 4CA11F          JMP PLOOPB
1FAD 60       RETURN RTS
1FAE          ;
1FAE 00       MESSAG BYT 00             ;MESSAGE 0
1FAF          ;
1FAF 455252   MESSA  ASC 'ERROR << HIT G' ;ERROR MESSAGE
1FB2 4F5220
1FB5 3C3C20
1FB8 484954
1FBB 2047
1FBD 00              BYT 00             ;DURING THE LOADING
1FBE          ;
1FBE 0A0D     MESSB  HEX 0A0D           ;MESSAGE 2--MESSAGE WHEN
1FC0 44554D          ASC 'DUMP B/M'     ;STARTING THE DUMPER
1FC3 502042
1FC6 2F4D
1FC8 00              BYT 00
1FC9          ;
1FC9 0A0D     MESSC  HEX 0A0D           ;MESSAGE 3--ASKING FOR
1FCB 465253          ASC 'FRST/LAST/AUTO?' ;THE ADDRESSSES
1FCE 542F4C
1FD1 415354
1FD4 2F4155
1FD7 544F3F
1FDA 0A0D            HEX 0A0D
1FDC 00              BYT 00
1FDD          ;
1FDD 0A0D     MESSD  HEX 0A0D           ;MESSAGE 4--ASKING FOR A "Y"
1FDF 524541          ASC 'READY ?'      ;WHEN READY TO DUMP
1FE2 445920
1FE5 3F
1FE6 00              BYT 00
1F6A 65E6            ADC CHCK
1F6C 85E6            STA CHCK
1F6E 9002            BCC *+4
1F70 E6E7            INC CHCK+1
1F72 60              RTS
1F73          ;
1F73 208A1F   ADRIN  JSR DIGIN          ;GET 2 HEX DIGITS
1F76 0A              ASL                ;AND CALCULATE BYTE, STORING IT
1F77 0A              ASL                ;IN LOCATION "ADRES+Y"
1F78 0A              ASL
1F79 0A              ASL
1F7A 99E000          STA ADRES,Y
1F7D 208A1F          JSR DIGIN
```

MICRO







Line Editor for OSI 540 Board

The program presented here allows elementary line editing functions for OSI computers using BASIC-in-ROM. The reader can expand the program as he feels is necessary to include more advanced features, such as insert and delete.

E.D. Morris Jr.
3200 Washington
Midland, Michigan 48640

Users of OSI computers are painfully aware that if a mistake is discovered in the 63rd character of a BASIC line, the entire line must be retyped. I have watched in awe as PET owners zip the cursor across the screen and correct the offending character in a few keystrokes. OSI machines lack this very useful feature as standard equipment. However don't despair, this article describes a software patch to allow line editing on OSI machines using the 540 video board and BASIC-in-ROM. The program provides the basic editing functions, but the user can add additional features as he wishes. The technique can also be applied to the C1P, subject to limitations discussed later.

A line editor must perform three functions. First it must find the line to be edited, then make the changes, and finally put the line back into the BASIC program. Finding the line is easy, just LIST it. The data is then on the screen. The line editor can read a character from the screen, copying it exactly, whenever a designated key is hit. If any other character is typed, that character is inserted into the new line instead of the screen character. Now comes the hard part: How do you get the line back into BASIC?

The new line must be inserted at the proper location, moving the rest of the program and refixing all the pointers. This is exactly the job done by the BASIC input routines. The line editor can be much simpler if BASIC can be fooled into believing that you re-typed the entire line.

Let us first examine the workings of the BASIC input routines. After cold starting BASIC, try typing in the following line

10ABCDE

If you press RETURN, this line will be entered into the BASIC text. However, instead of RETURN, press the BREAK key and jump to the machine monitor mode. Examine the data stored at locations $0013 to $0019. You should find

Location	Data	ASCII
$0013	31	1
$0014	30	0
$0015	41	A
$0016	42	B
$0017	43	C
$0018	44	D
$0019	45	E

The data at these locations is the hex representation of the ASCII characters you just typed. Locations $0013 through $005A are the input buffer. Thus to simulate keyboard input, the line editor must store the corrected line in this buffer. The next trick is to get BASIC to accept this data. First the "X" and "Y" registers must be set to point at the input buffer and then a jump made to the proper location in BASIC.

Try the following experiment. Cold start BASIC, then jump to the machine monitor. Using the monitor, fill locations $0013 to $0019 with the hex data from the above example adding a $00 at location $001A. Again using the machine monitor, write the following program at $0250.

```
$0250 A2 12     LDX #$12
$0252 A0 00     LDY #$00
$0254 4C 80 A2  JMP $A280
```

Then execute the program starting at $0250. The pointers are set to the input buffer, then a jump is made into ROM. There will be no indication that anything happened, but you are now back in BASIC. Type LIST and

10ABCDE

will appear. This technique has convinced BASIC to accept a line of data stored in the input buffer as if it had been typed in. Try using this method to input other lines of data, remembering to make the final character a null or $00.

The final link to writing a line editor is now at hand. Following is a listing of an editor assembled at address $0240. The program assumes that the line to be edited has been previously listed and now appears on the screen starting at $D641. The line editor is called through the USR function. After clearing several screen locations, the program displays an "up arrow" ($5E) as a cursor immediately below the line to be edited. The subroutine at $FFEB gets a character from the keyboard. If this character is a "space bar" ($20), one character is copied from the old line into the input buffer and displayed on the screen below the cursor. The cursor will move backwards on a "backspace" or $5F input. A RETURN or $0D indicates that you are finished editing that line. Since the space bar is used for direct copying, something else must be used for a "space". I have chosen the "#" sign or $23. Any other character typed is assumed to be corrected input, and is stored in the buffer and on the screen.

The RETURN key causes the program to display "OK" and places a null at the end of the input line. The pointers are set as described above, and a jump made back into BASIC.

If the program is moved to reside in a different memory location, the jump absolute instructions at lines $0282 and $0288 must be changed.

For those of you who are not into machine code, I have included a BASIC program to set up this patch and then erase itself. Once the line editor is entered, either by BASIC or via machine code, load the program you wish to edit. Then add the following line to your BASIC program:

```
1 POKE 11,64: POKE 12,2:
  Z=USR(1)
```

LIST the line you wish to edit, then type RUN. This will call the line editor and display the cursor directly under the listed line. The various valid commands were listed above. To run your program, either delete line one or enter RUN 10 (assuming your first line is 10). Before saving the corrected program, delete line one.

Now for the limitations of this simple editor. The line to be corrected must appear at a fixed position on the video screen. This is determined by the screen read instruction LDA $D641,X. The editor will not work if the line is not exactly at this position. For example, if a line is longer than 64 characters, the screen will scroll, moving the text up one line. A similar problem occurs when attempting to edit the last line of a program: the listed line appears too low on the video screen. In this case simply hit a RETURN to scroll up one line, and then type RUN to enter the editor.

Lines longer than 64 characters can be edited by changing the screen read instruction from LDA $D641,X to LDA $D601,X. This is accomplished by using different keys for the "copy" function, depending on the length of the line being edited. Lines shorter than 64 characters are copied by pressing the space bar. Longer lines are copied with the exclamation (!) key.

This editor can be modified to run on a C1P or Superboard by changing the appropriate screen locations. A BASIC listing of a C1P version is also given below. The editor is limited to a single video line, which, in the case of the C1P, is only 25 characters. In order to edit multiple lines, the editor must be able to skip over the unused bytes on the edges of the C1P video screen.

Listing 1

```
                ;*
                ;* LINE EDIT FOR OSI 540 BOARDS
                ;*
0240                    ORG $240
0240
0240            ;
0240 A920               LDA #$20
0242 A280               LDX #$80
0244 9DC0D6     CLR     STA $D6C0,X        ;CLEAR SCREEN BOTTOM
0247 CA                 DEX
0248 10FA               BPL CLR
024A A200               LDX #$00
024C A920       CUR     LDA #$20           ;REMOVE CURSOR
024E 9D80D6             STA $D680,X
0251 9D82D6             STA $D682,X
0254 A95E               LDA #$5E           ;CURSOR
0256 9D81D6             STA $D681,X        ;PLACE CURSOR
0259 20EBFF             JSR $FFEB          ;GET KEY STROKE
025C C920               CMP #$20           ;SPACE BAR FOR SHORT LINE
025E F019               BEQ COPY
0260 C921               CMP #$21           ;EXCLAMATION FOR LONG LINE
0262 F010               BEQ LONG
0264 C90D               CMP #$0D           ;RETURN
0266 F023               BEQ DONE
0268 C95F               CMP #$5F           ;BACKSPACE
026A F019               BEQ BACK
026C C923               CMP #$23           ;# FOR SPACE
026E D00C               BNE WSCR           ;MUST BE CORRECTION
0270 A920               LDA #$20           ;SPACE
0272 D008               BNE WSCR           ;ALWAYS
0274 BD01D6     LONG    LDA $D601,X        ;READ SCREEN (LONG)
0277 D003               BNE WSCR           ;ALWAYS
0279 BD41D6     COPY    LDA $D641,X        ;READ SCREEN (SHORT)
027C 9DC1D6     WSCR    STA $D6C1,X        ;WRITE SCREEN
027F 9513               STA $13,X          ;INPUT BUFFER
0281 E8         L1      INX
0282 4C4C02             JMP CUR
0285 CA         BACK    DEX                ;BACK-SPACE
0286 30F9               BMI L1             ;LIMIT BACK SPACE
0288 4C4C02             JMP CUR
028B A900       DONE    LDA #$00
028D 9513               STA $13,X          ;NULL INTO BUFFER
028F A992               LDA #$92
0291 A0A1               LDY #$A1
0293 20C3A8             JSR $A8C3          ;DISPLAY "OK" MESSAGE
0296 A212               LDX #$12
0298 A000               LDY #$00
029A 4C80A2             JMP $A280          ;BACK TO BASIC
```

Listing 2

```
10  PRINT "LINE EDITOR FOR OSI C1P OR SUPERBOARD"
80  FOR I = 576 TO 668: READ J: POKE I,J: NEXT
90  NEW
100  DATA 169,32,162,128,157,192,214,202,16,250
110  DATA 162,0,169,32,157,128,214,157,130,214
120  DATA 169,94,157,129,214,32,235,255,201,32
130  DATA 240,25,201,33,240,16,201,13,240,35
140  DATA 201,95,240,25,201,35,208,12,169,32
150  DATA 208,8,189,1,214,208,3,189,65,214
160  DATA 157,193,214,149,19,232,76,76,2,202
170  DATA 48,249,76,76,2,169,0,149,19,169
180  DATA 146,160,161,32,195,168,162,18,160,0
190  DATA 76,128,162
```

OSI AARDVARK OSI
NOW MEANS BUSINESS!

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

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

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

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

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

Best of all, it is in BASIC (OS65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS – $39.95

THE EDSON PACK ALL MACHINE CODE GAMES FOR THE 8K C1P

INTERCEPTOR —You man a fast interceptor protecting your cities from Hordes of Yukky Invaders. A pair of automatic cannon help out, but the action speeds up with each incoming wave. It's action, action everywhere. Lots of excitement! $14.95

MONSTER MAZE – An Arcade style action game where you run a maze devouring monsters as you go. If one sees you first, you become lunch meat. Easy enough for the kids to learn, and challenging enough to keep daddy happy. $12.95

COLLIDE – Fast-paced lane-switching excitement as you pick up points avoiding the jam car. If you succeed, we'll add more cars. The assembler code provides fast graphics and smooth action. $9.95

SPECIAL DEAL—THE ENTIRE EDSON PACK— ALL THREE GAMES FOR $29.95

THE AARDVARK JOURNAL

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

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

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

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

24K and OS65D required, dual disks recommended. Specify system.
Accounts Receivable. $99.95

*** * * SPECIAL DEAL – NO LESS! * * ***

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

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

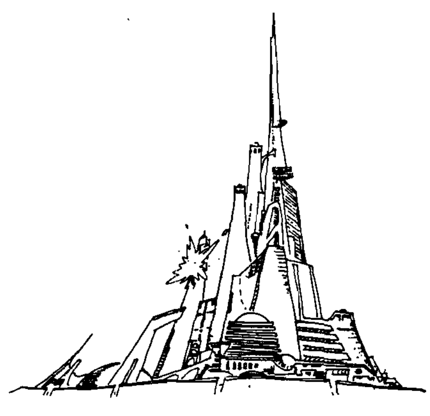
SUPERDISK II

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

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

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

AND FUN, TOO!



BOOKKEEPING THE EASY WAY – WITH BUSINESS I

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

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

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

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

GAMES FOR ALL SYSTEMS

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

MINOS - 8K - – Features amazing 3D graphics You see a maze from the top, the screen blanks and when it clears, you are in the maze at ground level finding your way through on foot. Realisti enough to cause claustrophobia. – $12.95

NEW – NEW – NEW

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

TIME TREK - 8K - Real Time and Real graphic Trek. See your torpedoes hit and watch you instruments work in real time. No more ur realistic scrolling displays! – $9.95

SUPPORT ROMS FOR BASIC IN ROM MA CHINES – C1S/C2S. This ROM adds line ed functions, software selectable scroll window bell support, choice of OSI or standard keyboar routines, two callable screen clears, and softwar support for 32-64 characters per line vide Has one character command to switch mod 2 C1P from 24 to 48 character line. When i stalled in C2 or C4 (C2S) requires installatic of additional chip. C1P requires only a jump change. – $39.95
C1E/C2E similar to above but with extend machine code monitor. – $59.95

Please specify system on all orders

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

OSI

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

OS

Life In a Wrap-around Universe

A novel variation on the oldest computer game of all.

Paul Krieger
3268 S. Cathay Cr.
Aurora, Colorado 80013

Ever wonder what would happen if your gliders could soar for a 1000 generations? Where does a puffer train go? Here is a wraparound version of John Conway's cellular automata "LIFE."

Life is normally limited to a fairly small grid of squares where patterns run out of space after only a few generations. In this version it is a string of 1024 cells so a pattern going off either side of the screen will re-appear at the other.

By testing the first 3 bits of the 4th, 8th, and 12th bytes, a matrix is created and the standard rules of LIFE are applied. The 1st 3 bits of byte 4 are numbered 1,2,3. The 1st 3 bits of byte 8 are numbered 4,5,6 and the 1st 3 bits of byte 12 are 7,8, and 9. Cell 5 is the subject cell.

First, the program counts the number of bits (except for #5) that are "1." Then bit 5 is tested to determine if it is on or off. If bit 5 is on and there were exactly 2 or 3 cells on, it is left on. If there were not, cell 5 is set to zero. If 5 was not on and exactly 3 of the other cells were on, it is set on.

Once the cells have been counted and set the 128 bytes are shifted 1 bit left, and the process continues again until all 128 bytes have been tested. As they are set, the bits being set are transformed into bytes on the screen so that at this point, they must be copied back to the bit list before the entire process begins once again.

Main Program

```
4  REM
5  REM  VIRTUAL LIFE
6  REM  BY PAUL KRIEGER
7  REM
10  GOSUB 1400
15 Q = 111
20  PRINT "INSTRUCTIONS"
24  PRINT
25  PRINT "THIS PROGRAM CREATES"
26  PRINT "A SIMULATION OF"
27  PRINT "ONE CELLED LIFE."
28  PRINT "ENTER A PATTERN"
29  PRINT "OF CELLS TO START."
30  PRINT "CURSOR CONTROLS": PRINT
31  PRINT "O=UP, P=RIGHT"
35  PRINT "K=LEFT, L=DOWN"
40  PRINT "J=ERASE,I=CENTER"
50  PRINT "+=DEPOSIT CELL"
55  PRINT "E=GENERATE CELLS": PRINT "T=END PROGRAM"
60  PRINT
70  PRINT "TYPE 'R' TO CONTINUE"
95  INPUT A$: GOSUB 1400
130  INPUT "(R)ANDOM OR (P)LAN";A$
131  IF  LEFT$ (A$,1) = "R" THEN 200
132  GOSUB 1400
134 S = 53775
135  POKE S,43: GOSUB 1500
136  POKE 11,00: REM  LOW DESTINATION
137  POKE 12,25: REM  HIGH, =$1900
138 Q =  USR (Q): GOTO 2100
139 .REM  2100 IS PAUSE BETWEEN SCREENS
140  REM  138--GOTO GENERATE CELLS
200  PRINT : PRINT "HOW MANY CELLS"
205  PRINT "SHOULD I GENERATE";
210  INPUT E
240  GOSUB 1400
250  FOR C = 1 TO E
260 D =  INT (1024 *  RND (1) + 1)
270 D = D + 53379
280  POKE D,Q
290  NEXT C
300  GOTO 136
1399  REM  CLEAR SCREEN SUBROUTINE
1400  POKE 11,237
1410  POKE 12,25: REM  SETUP $19ED
1420 Q =  USR (Q)
1430  RETURN
1499  REM  TEST CURSOR KEYS
1500  POKE 530,1
1510 K = 57088
1520  POKE K,223
1530  IF  PEEK (K) = 191 THEN 1830: REM  L,DOWN
1540  IF  PEEK (K) = 223 THEN 1870: REM  O,UP
1550  POKE K,247
1570  IF  PEEK (K) = 251 THEN 1920: REM  J,ERASE
1580  IF  PEEK (K) = 253 THEN 1940: REM  K,LEFT
1590  POKE K,253
```

```
1600  IF  PEEK (K) = 253 THEN 1980: REM  P,RIGHT
1610  IF  PEEK (K) = 251 THEN 2020: REM  +,DEPOSIT
1620  POKE K,239
1640  IF  PEEK (K) = 253 THEN 1800: REM  I CENTER
1645  IF  PEEK (K) = 191 THEN 1660: REM  E GENERATE
1650  GOTO 1520
1660  POKE 530,0
1670  RETURN
1799  REM  PERFORM SCREEN COMMANDS
1800  IF  PEEK (S) <  > Q THEN  POKE S,32
1810 S = 53775
1820  IF  PEEK (S) <  > Q THEN  POKE S,43
1825  GOTO 1510
1830  IF  PEEK (S) <  > Q THEN  POKE S,32
1840 S = S + 32: IF S > 54171 THEN S = S - 800
1850  IF  PEEK (S) <  > Q THEN  POKE S,43
1860  GOTO 1510
1870  IF  PEEK (S) <  > Q THEN  POKE S,32
1880 S = S - 32: IF S < 53379 THEN S = S + 800
1890  IF  PEEK (S) <  > Q THEN  POKE S,43
1895  GOTO 1510
1920  POKE S,32
1930  GOTO 1510
1940  IF  PEEK (S) <  > Q THEN  POKE S,32
1950 S = S - 1: IF S < 53379 THEN S = 54171
1960  IF  PEEK (S) <  > Q THEN  POKE S,43
1970  GOTO 1510
1980  IF  PEEK (S) <  > Q THEN  POKE S,32
1990 S = S + 1: IF S > 54171 THEN S = 53379
2000  IF  PEEK (S) <  > Q THEN  POKE S,43
2010  GOTO 1510
2020  POKE S,Q: GOTO 1510
2035  IF Q > 255 THEN Q = 0
2040  GOTO 1510
2099  REM  COUNT CYCLES, PAUSE BETWEEN SCREENS
2100  PRINT "CYCLE";CY
2110 CY = CY + 1
2115  REM  INSERT "GOTO 2170" HERE
2116  REM  IF YOU DON'T WANT TO STOP
2120  POKE 530,1
2125 K = 57088
2130  POKE K,239
2140  IF  PEEK (K) = 191 THEN 2170: REM "E"
2150  IF  PEEK (K) = 239 THEN  END : REM "T"
2160  GOTO 2125
2170  POKE 11,46
2180  POKE 12,25: REM  TO $192E
2190 Q =  USR (Q)
2200  GOTO 2100
2990  REM  STORES OR READS MACHINE LANGUAGE
2991  REM  SUBROUTINE:REM NOTE ***
2992  REM  WHEN SAVING TO OSI TAPE YOU MUST
2993  REM  TYPE "RUN 3000" AFTER BASIC
2994  REM  "OK".  ON LOAD, MACHINE WILL
2995  REM  PERFORM THIS FUNCTION FROM TAPE
3000  IF  PEEK (515) = 255 THEN 3070
3010  FOR X = 6400 TO 6656: REM  DECIMAL OF MAC CD
3020 K =  PEEK (X)
3030  PRINT K
3040  NEXT X
3050  END : REM  END OF CODE TO COPY MACH TOTAPE
3060  REM  ROUTINE TO READ MACHINE CODE FROM TAPE
3070  FOR X = 6400 TO 6656
3080  INPUT K
3085  POKE X,K
3090  NEXT X
3100  POKE 515,0
3110  END
```

This is a hybrid program for the Ohio Scientific C1P with 8K of memory, written in both Microsoft BASIC and machine language. Since no page zero processing is done it should be fairly easy to convert it to any 6502 computer.

Key in the following machine language code using your monitor. Then you can save both the BASIC and the machine code with the SAVE/LIST, as though it were a BASIC program. While the tape is still running, and after the BASIC portion has finished, type "RUN3000 return."

Copy screen to matrix subroutine.

```
1900-   A9 D0      LDA   #$D0
1902-   8D 0C 19   STA   $190C
1905-   D8         CLD
1906-   A0 04      LDY   #$04
1908-   A2 00      LDX   #$00
190A-   BD 00 D3   LDA   $D300,X
190D-   C9 20      CMP   #$20
190F-   F0 08      BEQ   $1919
1911-   AD 04 18   LDA   $1804
1914-   09 80      ORA   #$80
1916-   4C IE 19   JMP   $191E
1919-   AD 04 18   LDA   $1804
191C-   29 7F      AND   #$7F
191E-   8D 04 18   STA   $1804
1921-   20 C2 19   JSR   $19C2
1924-   E8         INX
1925-   D0 E3      BNE   $190A
1927-   EE 0C 19   INC   $190C
192A-   88         DEY
192B-   D0 DD      BNE   $190A
```

Test and set cells.
Move result to screen.

```
192D-   60         RTS
192E-   A9 D0      LDA   #$D0
1930-   8D 9A 19   STA   $199A
1933-   A0 04      LDY   #$04
1935-   4C 3E 19   JMP   $193E
1938-   EA         NOP
1939-   EA         NOP
193A-   EA         NOP
193B-   EA         NOP
193C-   EA         NOP
193D-   EA         NOP
193E-   A2 21      LDX   #$21
1940-   A9 00      LDA   #$00
1942-   8D 00 18   STA   $1800
1945-   A9 20      LDA   #$20
1947-   2C 04 18   BIT   $1804
194A-   08         PHP
194B-   10 03      BPL   $1950
194D-   EE 00 18   INC   $1800
1950-   28         PLP

1951-   08         PHP
1952-   50 03      BVC   $1957
1954-   EE 00 18   INC   $1800
1957-   28         PLP
1958-   F0 03      BEQ   $195D
195A-   EE 00 18   INC   $1800
195D-   A9 20      LDA   #$20
195F-   2C 08 18   BIT   $1808
1962-   08         PHP
1963-   10 03      BPL   $1968
1965-   EE 00 18   INC   $1800
1968-   28         PLP
1969-   F0 03      BEQ   $196E
196B-   EE 00 18   INC   $1800
196E-   A9 20      LDA   #$20
1970-   2C 0C 18   BIT   $180C
1973-   08         PHP
1974-   10 03      BPL   $1979
1976-   EE 00 18   INC   $1800
1979-   28         PLP
```

```
197A-   08          PHP
197B-   50 03       BVC   $1980
197D-   EE 00 18    INC   $1800
1980-   28          PLP
1981-   F0 03       BEQ   $1986
1983-   EE 00 18    INC   $1800
1986-   2C 08 18    BIT   $1808
1989-   50 18       BVC   $19A3
198B-   AD 00 18    LDA   $1800
198E-   C9 02       CMP   #$02
1990-   30 0C       BMI   $199E
1992-   C9 04       CMP   #$04
1994-   B0 08       BCS   $199E
1996-   A9 6F       LDA   #$6F
1998-   9D 00 D4    STA   $D400,X
199B-   4C AD 19    JMP   $19AD
199E-   A9 20       LDA   #$20
19A0-   4C 98 19    JMP   $1998
19A3-   AD 00 18    LDA   $1800
19A6-   C9 03       CMP   #$03
19A8-   F0 EC       BEQ   $1996
19AA-   4C 9E 19    JMP   $199E
19AD-   20 C2 19    JSR   $19C2
19B0-   E8          INX
19B1-   F0 03       BEQ   $19B6
19B3-   4C 40 19    JMP   $1940
19B6-   EE 9A 19    INC   $199A
19B9-   88          DEY
19BA-   F0 03       BEQ   $19BF
19BC-   4C 40 19    JMP   $1940
19BF-   4C 00 19    JMP   $1900
```

Rotate 128 bytes left 1 bit.

```
19C2-   8A          TXA
19C3-   48          PHA
19C4-   98          TYA
19C5-   48          PHA
19C6-   A2 7F       LDX   #$7F
19C8-   2C 04 18    BIT   $1804
19CB-   08          PHP
19CC-   3E 04 18    ROL   $1804,X
19CF-   CA          DEX
19D0-   D0 FA       BNE   $19CC
19D2-   3E 04 18    ROL   $1804,X
19D5-   28          PLP
19D6-   10 08       BPL   $19E0
19D8-   AD 83 18    LDA   $1883
19DB-   09 01       ORA   #$01
19DD-   4C E5 19    JMP   $19E5
19E0-   AD 83 18    LDA   $1883
19E3-   29 FE       AND   #$FE
19E5-   8D 83 18    STA   $1883
19E8-   68          PLA
19E9-   A8          TAY
19EA-   68          PLA
19EB-   AA          TAX
19EC-   60          RTS
```

End of code. Machine language clear screen routine.

```
19ED-   A0 FF       LDY   #$FF
19EF-   A9 20       LDA   #$20
19F1-   99 00 D0    STA   $D000,Y
19F4-   99 00 D1    STA   $D100,Y
19F7-   99 00 D2    STA   $D200,Y
19FA-   99 00 D3    STA   $D300,Y
19FD-   88          DEY
19FE-   D0 F1       BNE   $19F1
1A00-   60          RTS
```

There are 6 BASIC language subroutines and 4 machine code subroutines. The BASIC routines are:

1. Housekeeping, display instructions
2. Call machine screen clear
3. Test keys for setup cells
4. Perform cell setup screen commands
5. Read and Write machine code from tape into memory
6. Count cycles and pause between generations.

The machine routines are:

1. Copy screen to bit list
2. Test and set cells, move result to screen
3. Rotate 128 bytes left one bit
4. Clear screen.

Of special interest is the machine code read and store routine located in BASIC lines 3000-3110. The 6 statements in 3000-3050 store machine code tape onto the end of a BASIC program when you type RUN3000. The 6 statements from 3070-3110 will read the machine code back into memory after the BASIC program is loaded. You can save any machine language code, using these 12 statements, by changing the low and high memory addresses in lines 3010 and 3070.

MICRO

New Publications

(Continued from page 30)

Quiz-The FOR Statement. *Program Control With Decision Making*—The IF-THEN Decision Maker; AND, OR, and NOT; IF-THEN-ELSE; Metric Conversion Program; Quiz-IF-THEN and IF-THEN-ELSE. *Further Control*—The WHILE Statement; REPEAT-UNTIL; Revising the Metric Program; GOTO Where; CASE: An Easier Way To Make Multiple Choices; CASE and BOOLEANs; The Metric Conversion Program Once Again; Quiz. *Procedures (The Second Time Around) and Functions*—Procedures Once Again; Quiz-Parameters; Functions-the Cousin of Procedures; FORWARD-Naming a Procedure or Function Before Its Time; Quiz-Functions. *STRINGs and LONG INTEGERs*—Maximum STRING Length; STRING Intrinsics; Inputting Numbers With STRINGs; Quiz-STRINGs; Using LONG INTEGERs for Increased Accuracy; Exercises; Quiz-LONG INTEGERs. *More Data Types*—Arrays-Linking Scalars Together; Quiz-Arrays; Customized Types-"Enumerated User-Defined Types; Quiz-Enumerated User-Defined Types; Subrange Data Types; Quiz-Subrange Types; Sets; Quiz-Sets; Putting It All Together-The Tic-Tac-Toe Program. *Appendices A: Pascal's Advantages—A Summary. B. Pascal's Bummers. C. Other Parts of a Pascal System*—Assembler; Library Linker; Dynamic Debugger. *D. ASCII Character Codes. E. Assembly Language Interfacing*—Why Use Assembly Language With Pascal?; How Pascal Handles Assembly Language; External Procedures and Functions; The Five Steps; A Practical Assembly Language Example: PEEKPOKE; The Pascal Library; Quiz. *F. The 6502 Microprocessor. G. Inaccuracies of the Amortization Loan Formula. H. Answers to Quizzes. Index.*

The Pascal Handbook by Jacques Tiberghien. Sybex, Inc. (2344 Sixth Street, Berkeley, California 94710), 1981, x, 476 pages, diagrams, 7 × 9 inches, paperbound.
ISBN: 0-89588-053-9 $14.95

A comprehensive, alphabetical dictionary of every Pascal symbol, reserved word, identifier, and operator for most existing versions of Pascal, including Jensen & Wirth (standard and CDC versions), H-P1000, OMSI(DEC), Pascal/Z, ISO, and UCSD Pascal. Each of the 180 entries contains the definition, syntax diagram, semantic description, implementation details, and program examples.

LOGICAL SOFTWARE, INC.

ANNOUNCES:

DATA EASE

A NEW, EASY TO USE DATA BASE MANAGER FOR OHIO SCIENTIFIC 65 U SYSTEMS

— Standard Data Base Features
PLUS
— Hierarchal Structure
— Modify or Restructure the Data Base At Any Time
— Up to 8 Files in Memory At Once
— Can be Used from Basic Programs
— Complete Documentation and Tutorial

Introductory Price $99.95
$2.00 Postage & Handling

Finally ....logical software for OSI

Logical Software, Inc.
P.O. Box 354
Farmington, MI 48024
(313) 474-8774

Dealer's Inquiries Invited

SOFTWARE FOR OHIO SCIENTIFIC

VIDEO EDITOR
Video Editor is a powerful full screen editor for disk-based OSI systems with the polled keyboard (except C1P). Allows full cursor-control with insertion, deletion and duplication of source for BASIC or OSI's Assembler/Editor. Unlike versions written in BASIC, this machine-code editor is co-resident with BASIC (or the Assembler), autoloading into the highest three pages of RAM upon boot. Video Editor also provides single-keystroke control of sound, screen format, color and background color. Eight-inch or mini disk: $14.95. Specify amount of RAM.

SOFT FRONT PANEL
Soft Front Panel is a software single-stepper, slow-stepper and debugger-emulator that permits easy development of 6502 machine code. SFP is a fantastic monitor, simultaneously displaying all registers, flags, the stack and more. Address traps, opcode traps, traps on memory content and on port and stack activity are all supported. This is for disk systems with polled keyboard and color (b&w monitor ok). Uses sound and color capabilities of OSI C2/C4/C8 systems (not for C1P). Eight-inch or mini disk $24.95. Specify amount of RAM. Manual only, $4.95 (May be later credited toward software purchase). Six page brochure available free upon request.

TERMINAL CONTROL PROGRAM
OSI-TCP is a sophisticated Terminal Control Program for editing OS-65D3 files, and for uploading and downloading these files to other computers through the CPU board's serial port on OSI C2, C4 and C8 disk-based systems with polled keyboards. Thirteen editor commands allow full editing of files, including commands for sending any text out the terminal port and saving whatever text comes back. INDUTL utility included for converting between BASIC source and TCP file text. Eight-inch or mini disk $39.95. Manual only, $2.95.

OSI-FORTH 2.0 / FIG-FORTH 1.1
OSI-FORTH 2.0 is a full implementation of the FORTH Interest Group FORTH, for disk-based OSI systems (C1,C2,C3,C4,C8). Running under OS65D3, it includes a resident text editor and 6502 assembler. Over one hundred pages of documentation and a handy reference card are provided. Requires 24K (20K C1P). Eight-inch or mini disk $79.95. Manual only, $9.95. "OSI-FORTH Letters" software support newsletter $4.00/year.

All prices postpaid. Florida residents add 4% tax. Dealer inquiries are invited. Allow 30 days for delivery.

WRITE FOR FREE CATALOG OF SOFTWARE AND HARDWARE FOR OHIO SCIENTIFIC !!

Technical Products Company
P.O. Box 12983 Univ. Station
Gainesville, Florida 32604

SOFTWARE FOR OSI

VIDEO GAMES 3 ..NEW!.......................... **$14.95**
Three games. Meteor Mission is an asteroids game. Space Wars is a battle between two starships. Meteor Wars is a combination of the two above games. All three are in machine language with fast, real time action, and super graphics.

ADVENTURE: IMMORTALITY **$11.95**
You are an intrepid explorer searching for the fabled "Dust of Immortality". This is the largest adventure yet for 8K OSI! With hidden room load so you can't cheat.

SUPER BUG! **$6.95**
Here's a super-fast, BASIC/Machine language hybird race game. Ten levels of difficulty and a infinately changing track will keep you challenged.

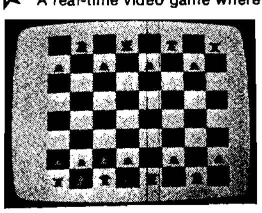
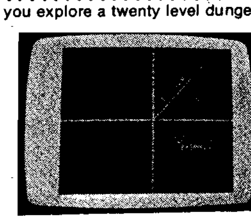
DUNGEON CHASE **$9.95**
A real-time video game where you explore a twenty level dungeon.

STARGATE MERCHANT ..NEW!.................. **$9.95**
You are a trader in the distant future, traveling through 'stargates' to get to various star systems. Part video game, part board game, always challenging.

ADVENTURE: MAROONED IN SPACE **$11.95**
An adventure that runs in 8K! Save your ship and yourself from destruction.

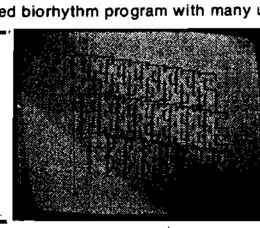
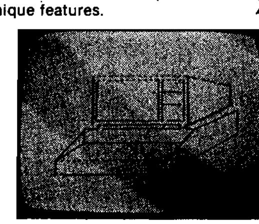
DISASSEMBLER **$11.95**
Use this to look at the ROMs in your machine to see what makes BASIC tick. Reconstruct the assembler source code of machine language programs to understand how they work. Our disassembler outputs unique suffixes which identify the addressing mode being used, no other program has this!

SUPER! BIORHYTHMS **$14.95**
A sophisticated biorhythm program with many unique features.

SUPER EXPANSION BOARD

GAMING DISPLAYS
HI-RES PLOTTING
CUSTOM CHARACTERS
3-D VIEWS

SEB-1 Assembled, Tested, W/5k RAM—$249.00 Kit $139.00 Bare Board $59.00, SEB 2/3/4 Prices on request

The **SEB-1** is a high resolution color graphics/memory expansion board for the Ohio Scientific Challenger 1P/Superboard II personal computers. The board is designed to connect to the computer with no modification and provides the user with the following capabilities:
• Eleven software selectable display modes-from a 32X16 alphanumeric to a 256X192 point addressable display. • 16K user expansion RAM.
• User configurable RF or composite video output.
• Up to eight colors. • 6K display RAM.
• An eight bit parallel port w/handshake. • Two sixteen bit timers/counters.

The following expansion boards for the C2/C4/C8 series computers are available
SEB-2 Color graphics/floppy controller. Same graphics capabilities as the SEB-1 and a software/hardware compatible floppy controller. **SEB-3** 80X24 video display generator/floppy controller. **SEB-4** 48K RAM expansion board.

Coming soon, a ROM chip to support the SEB! Powerful screen editing, plus Apple-like graphics commands allow you to run modified Apple software. Write for prices.

Write for **FREE** catalog
(For International requests, please supply 2 oz. postage)

ORION SOFTWARE ASSO.
147 Main St. Ossining, NY 10562

Coming soon

ADVENTURE: VOLCANO OF KANTHOR **$29.95**
Simply the largest adventure anywhere for OSI! Requires 32K and one disk drive. Over 150 rooms! Magic, monsters, and the evil Klopsiwel await you. Sure to keep you amused for a LONG time.

VIDEO GAMES 4 .... **$14.95**
Invaders from Orion (R). Three fantastic, fast, real time invaders games, in machine language, of course. A simple invaders game, an invaders for two, and the most fantastic invasion game yet awaits you.

Step and Trace for C1P

This article presents a single step trace for BASIC programs.

M. Piot
36 r R.Poulin
14200 Herouville, France

Type RUN, press RETURN: nothing occurs! Is it the BIGBUG?

No! Press S and the first instruction is executed, press S again and the next instruction is executed, press T and the number of the line embedding the last executed instruction is displayed. Press U and the third instruction is executed *and* the number of the line is displayed. Press CTRL C and you can ask the computer for the value of a variable. Are you dreaming? No, you just use the 40 byte program in listing 1.

Since I believe a true computerist must never run a program before he has tried to understand how it works, here are some explanations for those of you not experienced enough with the routines in ROM (interpreter and monitor).

Though the monitor and the interpreter are in ROM, they sometimes jump briefly in RAM (at 0001, 0003, 0071, 00A1, 00A2, 00BC, 00C2, 0207, 020A for the interpreter and 0000, 00FE, 0218, 021A, 021C, 021E, 0220 for the monitor). The five last addresses (named VECTORS) are particularly interesting. Let me show you how they work with an example — the one concerning 021A.

Every time BASIC wants to output a character to the screen, it executes the following instruction:

```
20 EE FF
  (You can see one at
  A8F4-A8F5-A8F6)
```

This means jump to the routine beginning at FFEE (not EEFF), execute it and then come back.

Let's look at FFEE (in the monitor); there you find

```
6C 1A 02
```

which means jump to the routine whose address is stored in 021A-021B.

At 021A (in RAM) you find 69FF stored there by the monitor every time the BREAK key is pressed. What is FF69 (not 69FF)? It is the beginning of the video output routine which ends with a 60 at FF8A. This 60 means go back to the instruction following 20 EE FF.

You may wonder why Richard W. WEILAND (the next time you "cold start" your machine, answer A to the question "MEMORY SIZE?"!) didn't write 20 69 FF at A8F4. It is to allow you to eventually change the normal process by changing the address in 021A-021B. For example, change 69FF to 6CFF and you'll suppress the video output. 0207 is used every time BASIC asks for a character (from the keyboard or the cassette) through 20 EB FF. 021E is used whenever BASIC asks for SAVE through 20 F4 FF, and 0220 is used when BASIC asks for LOAD through 20 F7 FF.

Every time an instruction has been executed, BASIC jumps to the address stored at 021C-021D through 20 F1 FF. This address is normally FF9B, the beginning of the CTRL C routine. This is the heart of the program.

I have changed FF9B for 0222 where I have stored a program which is executed after every instruction of the BASIC program. Four commands are recognized:

S executes the next instruction
T displays the number of the line
U executes one instruction and displays the number of the line
CTRL C works as usual and allows you to ask the computer for the value of a variable (or more) by typing

```
PRINT X or PRINT X;Y
(for example)
```

in the immediate mode.

After a CTRL C, you may re-enter my program by pressing S, typing CONT, and pressing RETURN. This jumps to two routines in ROM: one beginning at FD00 which gets a character from the keyboard and stores it in the accumulator (A) of the 6502 microprocessor; one beginning at B95A which displays the number of the line.

How to Store the Program in RAM

To store your program in RAM you may "BREAK M" your system, type 0222/. Then enter the 40 bytes (one byte CR one byte CR etc....) and then "BREAK W" the system to run your program. You may also store those 40 bytes by "POKEing" them with the following program you run, using RUN 63992:

```
63991 END
63992 FOR I=546 TO 585
63993 READ W
63994 POKE I,W
63995 NEXT
63996 DATA 32,0,253,162,105,142,
      26,2,201,3,240,25,201,83
63997 DATA 240,21,201,85,240,14
63998 DATA 201,84,208,232,32,90,
      185,162,108,142,26,2,240
63999 DATA 3,32,90,185,76,155,255
```

How to Get Into the S T U Mode

As the first line of your program (or of the portion you want to study), you must use

```
POKE 667,96: POKE 541,2:
  POKE 540,34
```

POKE 541,2 and POKE 540,34 (numbers in decimal) store 0222 instead of FF9B in 021C-021D. I will let you find the *why* of POKE 667,96! (Hint: the 96 is an RTS.

Problems with INPUT?

When a program that is run in the T mode reaches an INPUT statement, the displaying of line numbers stops but "no ?" appears on the screen. Press RETURN U, answer the INPUT request as usual and go on tracing.

This program is not only a debugging aid, it is also very helpful to understand the way the interpreter runs programs.

```
                    GETCHR   EQU $FD00
                    DISPLN   EQU $B95A
                    CNTRLC   EQU $FF9B
                    BRKVEC   EQU $021A
0222 2000FD         START    JSR  GETCHR    CHARACTER IN A
0225 A269                    LDX  #$69
0227 8E1A02                  STX  BRKVEC    TO SUPPRESS VIDEO
                                            OUTPUT
022A C903                    CMP #$03       IS THIS A CTRL C?
022C F019                    BEQ  RTN
022E C953                    CMP 'S
0230 F015                    BEQ  RTN
0232 C955                    CMP 'U
0234 F00E                    BEQ  LNDISP
0236 C954                    CMP 'T
0238 D0E8                    BNE  START
023A 205AB9                  JSR  DISPLN    DISPLAYS LINE NO.
023D A26C                    LDX  #$6C
023F 8E1A02                  STX  BRKVEC    TO RESTORE VIDEO
                                            OUTPUT
0242 F003                    BEQ  RTN       (ALWAYS!)
0244 205AB9         LNDISP   JSR  DISPLN    DISPLAY LINE NO.
0247 4C9BFF         RTN      JMP  CNTRLC    NORMAL CTRL C
                                            ROUTINE
                             END
```

MICRO

An Introduction to Bit Pads

By Loren W. Wright

The following articles describe two microcomputer implementations of a bit pad. In the first, Peter Coyle describes how to use the 8-bit parallel interface version (cheaper than the IEEE-488 interface version) with a PET. The hardware aspect of the article is applicable to any microcomputer with a parallel port, and the software is convertible, with few changes, to almost any 6502 machine. The second article, by Ralph Erickson, describes a program to process data through an RS-232 interface (AIM 65) and save the data to tape or DAIM disk.

A bit pad can be a valuable addition to your microcomputer system, but many people are unaware of what a bit pad is, and what it can do. The following article (and photo) was compiled from information supplied by Summagraphics Corporation, the manufacturer of Bit Pad One and other bit pad and digitizing products.

Essentially, a bit pad is a rectangular tablet that senses the position of an electronic stylus or a crosshair "cursor" above its surface. This information is converted to digital information and sent to the computer. The stylus, with interchangeable non-marking and marking tips, is included with Bit Pad One, but one-, four-, and thirteen-button crosshair cursors are also available.

Operating Modes

Bit Pad One modes and sampling rate may be controlled externally under program control, or internally by switches on the logic board. The power-up mode and sampling rate are determined by the positions of the internal switch. Both the mode and sampling rate may be changed under program control from the host computer by sending the Bit Pad One either one ASCII character or eight-bit byte, depending on the resident interface. The following modes are available:

Point Mode—Depression of the stylus on the tablet, or pressing a button on the cursor causes one *x*-, *y*-coordinate pair (sample) to be output in the appropriate format.

Stream Mode—*x*-, *y*-coordinate pairs (samples) are generated continuously at the selected sampling rate when the stylus or cursor is in the proximity of the active area of the tablet. Pressing the stylus to the tablet, or depressing a button on the cursor marks the flag character (F) bit in the output string. This mode is typically used for CRT cursor control (cursor steering).

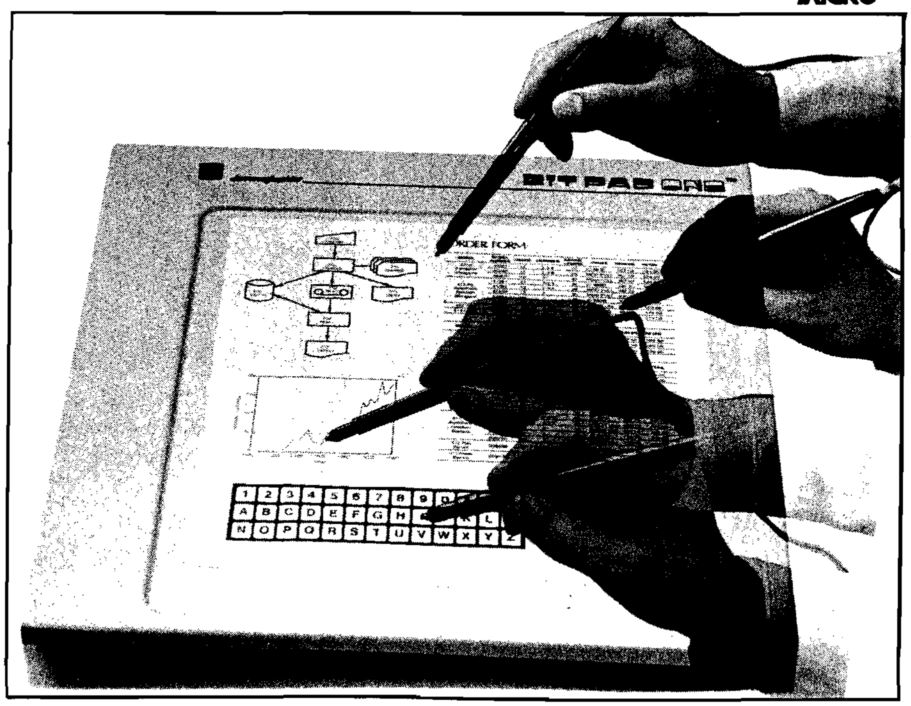
Switch Stream Mode—Depression of the stylus, or pressing a button on the cursor causes *x*-, *y*-coordinate pairs (samples) to be output continuously at the selected sampling rate until the stylus or button is lifted.

Bit Pad One comes in two sizes — 11" × 11" and 15" × 15", and with three interfaces — RS-232, 8-bit parallel, and IEEE-488. Prices (at press time) range from $730 for the 8-bit parallel version in the 11" × 11" size to $1395 for the IEEE-488 version in the 15" × 15" size. Also, I understand that Bit Pad One is now available with a 16-bit parallel interface, although first-hand details are not available at present. A power supply is also required — $95 for the U.S. model.

Applications

Applications of a bit pad are only limited by the user's imagination. Data entry can be done by checking the appropriate box on a pre-printed form laid on the tablet. To select items from the computer screen, the CRT cursor can be directed with the movement of the bit pad stylus. Patterns can be drawn on the screen using the bit pad as an electronic brush and canvas. In drafting, often-repeated symbols like doors and windows or NAND gates and transistors can be selected, and then positioned properly, using the stylus. In education, the process of typing in an answer can be eliminated, thus allowing the student to focus on the subject. Of course, game applications are probably the first things to come to mind.

MICRO



PET Interface to Bit Pad

A PET machine language sampling routine to read x-, y-coordinate data through the 8-bit parallel interface of the Summagraphics Bit Pad. Additional information has been supplied for hardware and software implementation on a SYM or AIM. A PET BASIC program is provided to drive the routine and write data to tape. Another reads data from tape.

Peter Coyle
Dept. of Anatomy
University of Michigan
Ann Arbor, Michigan 48109

Editor's Note: The Summagraphics Bit Pad described here is a discontinued model. Bit Pad One is the current comparable model. The main difference is that Bit Pad had a separate console, whereas Bit Pad One has all the electronics contained in the tablet unit. The hardware interface and program requirements are the same for the two models.

Mr. Coyle's original machine language sampling routine for the PET has been modified slightly by the MICRO staff to make implementation on other systems easier. Hardware connection information is summarized in table 1, and programming information is provided in table 2.

Data or instruction entry into a microcomputer via the keyboard is relatively slow. Quicker entry can be accomplished by placing a stylus over a coded string of information on a chart. A sensor detects the spatial position of the stylus, digitizes, and then transfers the x- and y-coordinate values to a computer for decoding. Coordinate values can code variables such as points in space, computer instructions, names, titles, parts, recipes, grades, costs records, and many others. A nearly-endless list may be generated.

Pet User Port J2 Contact		Function	Bit Pad D-Connector Pin #
B	(CA1)	Byte Available	20
C	(PA0)	D0	8
D	(PA1)	D1	10
E	(PA2)	D2	12
F	(PA3)	D3	14
H	(PA4)	D4	16
J	(PA5)	D5	18
L	(PA7)	Byte Received	19
M	(CB2)	Next Byte	21
N			23 GND

***Figure 1:* Hardware Interface**

***Table 1:* Parallel Port Connections**

Signal Name	J2 PET/CBM Parallel User Port	AA SYM VIA #2	AIM 65 J1
CA1	B	E	20
PA0	C	D	14
PA1	D	3	4
PA2	E	C	3
PA3	F	12	2
PA4	H	N	5
PA5	J	11	6
PA6	K	M	7
PA7	L	10	8
CB2	M	5	19
GND	N	1	1

Information compiled by MICRO staff.

***Table 2:* Parallel Port Addressing**

Address Description	Program Symbol	PET	SYM	AIM
Output register A, with handshaking	ORAHS	$E841 (59457)	$A801 (43009)	$A001 (40961)
Data direction register, Port A	DDRA	$E843 (59459)	$A803 (43011)	$A003 (40963)
Peripheral control register	PCR	$E84C (59468)	$A80C (43020)	$A00C (40972)
Interrupt flag register	IFR	$E84D (59469)	$A80D (43021)	$A00D (40973)
Output register A, without handshaking	ORANHS	$E84F (59471)	$A80F (43023)	$A00F (40975)

Information compiled by MICRO staff.

A two-dimensional coordinate system offers flexibility for many problems and the mapping of two variables, each on a different spatial axis. The Summagraphics Bit Pad, a digitizer for entering two-coordinate information into a computer, was interfaced to the 16K Commodore PET parallel user port. This article gives the hardware interface and presents software developed for successful interdevice communications.

Hardware

The Bit Pad consists of several system elements. There is an 11-inch square pad with magnetostrictive wires on a substrate beneath the surface. A strain wave is propagated along all wires simultaneously. On the pad surface, a moveable stylus or cursor senses the passing strain wave. Delay between initiation and sense time is used to code *x*- and *y*- coordinate positions of the stylus. The active area of the pad has about 8 million resolvable points with a spatial resolution of about 0.1 millimeter. A console cabinet houses the controller card, serial TTL line and 8-bit parallel port with handshake line connectors. Power supply is self-contained and an additional purchase. Data collection modes and digitizing rates can be specified via console cabinet switches or implemented through host processor control. The developed software does not utilize host processor control of collection modes nor digitizing rates.

Figure 1 indicates the wired connections and handshake signal names. No additional hardware logic elements were required for the interface. The Bit Pad has three handshake lines but there are only two on the PET prarallel user port. The problem is easily solved for only bits 0-5 of the byte convey coordinate data. Bit 7 of the parallel user port could therefore be used as the third handshake line (BYTE RECEIVED). The sampling routine keeps track of the byte number. One Cinch 251-12-30-160 board edge connector for the PET, three feet of 12 conductor ribbon cable, and the included Bit Pad data bus connector were utilized in making the hardware link.

Data and Handshake Lines

For each digitized point, five 8-bit bytes (words) of data are put on Bit Pad even-numbered lines 8-22 inclusively. Bits of the first transmitted word indicate the status of flag buttons on the optional cursor. These bits can be used to control program or computer activities, but the developed software discards the first byte. The second word bits 0-5 are less significant for the *x*-coordinate, while byte three bits 0-5 are

Listing 1

```
500 REM***PROG DIGITIZE
510 REM***BY PETER COYLE
515 REM***WRITTEN FOR 16K OR LARGER
520 REM***LOAD BIT PAD SAMPLING ROUTINE
530 REM***LOAD BLANK TAPE TO STORE X     AND Y VALUES
540 REM***DATA STORE 12800 DEC,3200HEX
545 REM
550 POKE 52,255: POKE 53,23: CLR: PROTECT MEMORY FROM BASIC
555 REM OLD ROMS--POKE 134,255: POKE 135,23
560 POKE 893,50: POKE 897,00: REM INITIALIZE DATA STORE BASE
570 TE=0: REM SET TAPE WRITE FLAG TO ZERO
580 PRINT"□": REM CLEAR SCREEN
590 PRINT"INPUT # SAMPLES": INPUT N: N=N*4: REM 4 BYTES/POINT
600 A%=INT(N/256): REM COMPUTE HI ORDER BYTE OF N
610 B%=INT(N-(256*A%)): REM COMPUTE LO BYTE OF N
620 POKE 828,B%: REM STORE LO N IN SAMPLING ROUTINE LOC $033C
630 POKE 829,A%: REM STORE HI N IN SAMPLING ROUTINE LOC $033D
640 PRINT"START SAMPLING DATA"
650 SYS(830): REM TRANSFER CONTROL TO SAMPLING ROUTINE
660 A=PEEK(893): REM FETCH BASE VALUE
670 B=PEEK(826): REM FETCH COUNTER
680 N=((A-50)*256+B): REM COMPUTE # PTS
690 GOSUB860: REM FETCH DATA POINTS
700 PRINT"IF DATA TO BE STORED ON TAPE, TYPE:"
710 PRINT"GOTO 730": REM PRINT ON SCREEN
720 STOP: REM WAIT FOR INSTRUCTION
730 GOSUB 750
740 END
750 REM***SUBROUTINE DUMP TO TAPE
760 TE=1: REM SET FLAG EQUAL TO ONE
770 PRINT"□": REM CLEAR SCREEN
780 PRINT"ENTER EXPERIMENT NUMBER": INPUT E$
790 PRINT"ENTER R / L HEMISPHERE": INPUT H$
800 PRINT"ENTER NUMBER OF X / Y POINTS": INPUT N$
810 OPEN1,1,1,E$+H$: REM OPEN AND NAME FILE
820 PRINT#1,STR$(N);",";E$;",";H$
830 GOSUB 860: REM FETCH X AND Y AND WRITE TO TAPE AND SCREEN
840 CLOSE 1
850 RETURN
860 REM***SUBROUTINE TO RETURN X AND Y
870 PRINT" I","  X","  Y": REM PRINT SCREEN COLUMN HEADERS
880 PRINT
890 FOR I=0 TO N-4 STEP 4
900 A=PEEK(12800+I):B=PEEK(12800+I+1): REM GET X LO AND HI BYTES
910 X=(B*64)+A: REM SHIFT X HI BITS & COMBINE WITH LO ONES
920 A=PEEK(12800+I+2): B=PEEK(12800+I+3): REM GET Y LO AND HI BYTES
930 Y=(B*64)+A: REM SHIFT Y HI BITS & COMBINE WITH LO ONES
940 IF TE=0 THEN 960: REM BYPASS WRITING TO TAPE IF FLAG 0
950 PRINT#1,X;",",Y: REM WRITE TO TAPE
960 PRINTI/4+1,X,Y: REM PRINT ON SCREEN
970 PRINT"------------------------------": REM UNDERLINE
980 NEXT I
990 RETURN
```

Listing 2

```
500 REM*****PROG DATA READER
510 REM*****BY PETER COYLE
520 REM*****READ IN X AND Y FROM TAPE
530 REM
560 DIM X(200),Y(200):REM DIM ARRAYS
570 PRINT"□":REM CLEAR SCREEN
580 PRINT"ENTER EXPERIMENT NUMBER": INPUT E$: REM ENTER FILENAME PART
590 PRINT"ENTER R / L HEMISPHERE": INPUT H$: REM ENTER FILENAME PART
600 PRINT"LOADING IN DATA"
610 OPEN1,1,0,E$+H$: REM OPEN FILE
620 INPUT#1,N,E$,H$: REM READ FROM TAPE INTO FILE
630 N=N/4: REM N= NUMBER OF SAMPLE POINTS
640 PRINT" I","   X","   Y":REM PRINT COLUMN HEADERS
650 FOR I=1 TO N
660 INPUT#1,X(I),Y(I): REM READ IN DATA
670 PRINT I,X(I),Y(I): REM PRINT DATA ON SCREEN
680 NEXT I
690 CLOSE 1: REM CLOSE FILE
700 END
```

more significant. The *y*-coordinate value is coded in bits 0-5 of words four and five, with the more significant bits in word five.

Software

Listing 1 is the program which defines (BASIC line 550) the top of RAM available to BASIC but above which the sampling routine stores coordinate values. As given, there is space for about 600 points for the 16K machine. On return (660) from the sampling routine, the Hi and Lo order data point bytes are combined (910 and 930) into a floating point number and displayed. Then the program requests input (700) if the data is to be written onto tape. Listing 2 reads stored data from tape.

For the sampling routine, Summagraphics provided a flow diagram of handshake signals that are required for any processor. An initial subroutine written in BASIC sampled points at about 1 sample/second. This was much too slow for our sampling needs. A 6502 Assembly Level Language version was written that avoids use of zero page locations which can cause problems with the new PET. The routine samples at about 64/second, which is the maximum rate of the Bit Pad. The Bit Pad One is even faster.

Listing 3 is code for the routine stored in the second cassette buffer. Data values are stored, starting at hexadecimal 3200 (decimal 12800) which can easily be changed by POKEing 897 and 898 with a new base number. Because one byte cannot code a number larger than 255, the 3200 base value is incremented when the byte counter (Y register) recycles. Consequently, when the BASIC program is run, the 3200 base is initialized each time. Software is included in the listing to drive the Huh Electronics beeper and needs no modification if the beeper is not used. We find that audio feedback during point sampling is helpful. Sampling need not continue until the entered number, N, of samples are obtained. Pressing a keyboard key stops the sampling process and causes return to the BASIC program. The number of samples obtained is computed (680) after PEEKing the values in locations 826 and 890 to determine how many times the counter recycled (660), and adding the current cycle count (670). Once obtained by the above scheme, *x*- and *y*-coordinate data can be used for distance measurements, counting, position coding, or other purposes.

MICRO

Listing 3

```
0800                ;*************************
0800                ;*                       *
0800                ;* INTERFACE ROUTINE FOR *
0800                ;* SUMMAGRAPHICS BIT PAD *
0800                ;*                       *
0800                ;*     BY PETER COYLE    *
0800                ;*                       *
0800                ;*************************
0800                ;*
0800                CNT1   EQU $033A
0800                POIN   EQU $033B
0800                LO     EQU $033C
0800                HI     EQU $033D
0800                ;
0800                CSTMTR EQU $E813          ;(59411) PET ONLY
0800                ;
0800                ;PET ADDRESSES--SEE TABLE FOR AIM &SYM EQUIVALENTS
0800                ;
0800                ORAHS  EQU $E841          ;(59457)
0800                DDRA   EQU $E843          ;(59459)
0800                PCR    EQU $E84C          ;(59468)
0800                IFR    EQU $E84D          ;(59469)
0800                ORANHS EQU $E84F          ;(59471) OUTPUT REGISTER A--NO HANDSHAKING
0800                NUMCHR EQU $009E          ;NUMBER OF CHARACTERS IN KEYBOARD BUFFER
0800                ;FOR OLD PET, NUMCHR EQU $020D
0800                ;
033E                       ORG $033E
033E                       OBJ $800
033E                ;
033E A901           INITAL LDA #$01           ;SET
0340 8D3A03                STA CNT1           ;COUNTER TO1
0343 A980                  LDA #$80           ;MAKE PA7 OUTPUT
0345 8D43E8                STA DDRA           ; & PA0-6 INPUT
0348 A000                  LDY #$00           ;INITIALIZE POINTER
034A A205           NEXTS  LDX #$05           ;BYTE COUNTER
034C 206503         HAND   JSR HAND1          ;HANDSHAKE BIT PAD
034F CA                    DEX                ;1 BYTE RECEIVED
0350 D0FA                  BNE HAND           ;GET NEXT BYTE OF SAMPLE
0352 209803                JSR ENDT           ;TEST FOR LAST BYTE
0355 CD3A03         COMP   CMP CNT1           ;TEST IF LAST SAMPLE
0358 F002                  BEQ END            ;LAST SAMPLE POINT
035A D0EE                  BNE NEXTS          ;NEXT SAMPLE POINT
035C 60             END    RTS                ;RETURN TO BASIC
035D AD4DE8         WAIT   LDA IFR            ;WAIT FOR INTERRUPT
0360 2902                  AND #$02           ;ON CA1 LINE B
0362 F0F9                  BEQ WAIT           ;BRANCH TO WAIT
0364 60                    RTS                ;RETURN
0365 A902           HAND1  LDA #$02           ;CONDITION
0367 8D4DE8                STA IFR            ;INTERRUPT FLAG REGISTER
036A A9ED                  LDA #$ED           ;SET CB2 (NEXT
036C 8D4CE8                STA PCR            ;BYTE) HI
036F 205D03                JSR WAIT           ;WAIT FOR CA1 (B.A.) HI
0372 AD41E8                LDA ORAHS          ;INPUT A, CLEAR FLAG
0375 293F                  AND #$3F           ;SHIFT BITS
0377 E005                  CPX #$05           ;1ST BYTE TEST
0379 F004                  BEQ SKIP           ;DON'T STORE 1ST BYTE
037B 990032         STORE  STA $3200,Y        ;STORE BYTE HERE
037E C8             CONT   INY                ;INCR INDEX POINTER
037F A9CC           SKIP   LDA #$CC           ;RESET NEXT BYTE LO
0381 8D4CE8                STA PCR            ;LINE CB2
0384 AD4FE8                LDA ORANHS
0387 0980                  ORA #$80           ;SET PA7
0389 8D4FE8                STA ORANHS         ;HI (B.R.)
038C 205D03                JSR WAIT           ;WAIT FOR CA1 (B.A.) LO
038F AD4FE8                LDA ORANHS
0392 297F                  AND #$7F           ;RESET PA7
0394 8D4FE8                STA ORANHS         ;(B.R.) LO
0397 60                    RTS                ;RETURN TO HAND+3
0398 AD3D03         ENDT   LDA HI
039B F00C                  BEQ TEST1          ;NO MORE HI BYTE
039D C000                  CPY #$00           ;NEW INDEX CYCLE?
039F D00F                  BNE TEST2          ;OLD INDEX CYCLE
03A1 CE3D03                DEC HI             ;DECREMENT HI BYTE
03A4 EE7D03                INC STORE+2        ;INC. BASE BY 256
03A7 D007                  BNE TEST2
03A9 CC3C03         TEST1  CPY LO             ;IS LAST BYTE IN?
03AC D002                  BNE TEST2          ;NOT THE LAST BYTE
03AE F00C                  BEQ FINI           ;LAST BYTE IN
03B0                ;
03B0                ;FOLLOWING CODE IS PET-SPECIFIC. SUBSTITUTE A
03B0                ;GETCHR ROUTINE AND TEST ON A PARTICULAR CHARACTER
03B0                ;FOR MACHINES OTHER THAN PET
03B0                ;
03B0 AD9E00         TEST2  LDA NUMCHR         ;TEST KEYBOARD IN (=$020D, OLD PET)
03B3 D007                  BNE FINI           ;KEYBOARD REQUEST STOP
03B5 A935                  LDA #$35           ;TURN ON BEEPER
03B7 8D13E8                STA CSTMTR         ;AFTER 4TH BYTE STORE
03BA D004                  BNE TALL           ;NO KEYBOARD INPUT
03BC                ;END OF PET-SPECIFIC CODE
03BC 8C3A03         FINI   STY CNT1           ;SAMPLING COMPLETE
03BF 98                    TYA                ;TRANSFER Y TO A
03C0 60             TALL   RTS                ;RETURN TO COMP
```

Bit Pad Routines for AIM 65

An assembly language program to interface AIM 65 BASIC to a digitizer (Bit Pad One) is described. The x-, y-coordinates of points on a photograph or chart can be stored in a BASIC array, simply by placing a stylus, or the crosshair of a cursor on the point, and closing a switch. Routines are also included to save and load BASIC arrays on cassette tape or disk (DAIM). These routines are called by the BASIC USR(W) command, with a single POKEd entry point, and W to indicate the desired routine.

Ralph O. Erickson
Department of Biology
University of Pennsylvania
Philadelphia, Pennsylvania 19104

The Rockwell AIM 65 is well designed for many applications in the laboratory. An important class of applications is undoubtedly the acquisition of data, either from instruments, such as a spectrophotometer (Saltero, R., 1980), or from a digitizer, as described in this article. With the programs listed here, you can log the x- and y-coordinates of a point on a photograph, drawing, or chart, mounted on the platen of the digitizer. This is done by placing the crosshair of a cursor on the point and pressing a button, or by depressing a stylus. The x-, y-values can be stored in BASIC arrays. In addition, you can save arrays as data files on cassette tape, or floppy disk, and load the saved data files into BASIC arrays.

The first routine in the source listing (2) is written for use with the Summagraphics Bit Pad One. It can be called by a BASIC program via the USR(0) function. My Bit Pad is equipped with a RS-232 interface, and its output (pin 2) connects to the serial input pin of the application connector (J1-Y) of the AIM. (Other Bit Pad models are available with 8-bit parallel, or IEEE-488 output interface.) The AIM TTY-KB switch must be left in KB position. Of the several options described in the Bit Pad User's Manual, I selected the point mode of transmission, (rather than stream, switched stream, or program control mode), set the baud rate at 1200, and selected binary data format, (rather than ASCII data format). In this mode of operation, the Bit Pad transmits one x-, -coordinate pair to the AIM as a sequence of 5 bytes, each time the stylus is depressed, or a button is pressed on the cursor. The first byte of the sequence is identified by bit 6 being set; in the next 4 bytes, bit 6 is clear. In addition, bit 2 of the first byte is set when the stylus is depressed or the button is pressed.

When the first byte, $44, is detected, the next 4 bytes are stored. They contain the binary-coded x-, y-coordinates of a point to 12-bit accuracy, and 0.005-inch resolution. Their format is changed to BASIC integer format, and they are stored indirectly in 4 bytes which can be accessed by BASIC. To make this possible, integer variables, X1%, Y1%, are defined at the beginning of the BASIC program, so that they are defined at the beginning of the BASIC variable area — the address of which is at $0075. BASIC can then re-assign them to other variables or array(s), where they are accessible for printing, saving as data

Listing 1

```
1  REM —BIT PAD INPUT & BASIC DATA FILES
2  REM —REM POSITION TAPE; TOGGLE<1>OFF; & SET "RECORD" TO SAVE DATA
3  REM —OR "PLAY" TO LOAD DATA
4  REM —TO SAVE EXISTING DATA, USE DIRECT "GOTO 60"; "RUN" DELETES ARRAY
    S!!
5   X1% = 0:Y1% = 0: REM  INITIALIZE INTEGER VARIABLES
6   POKE 4,0: POKE 5,63: REM -S/R AT $3F00
10  INPUT "NO. OF POINTS";N: DIM X%(1,N - 1)
20  INPUT "DIGITIZE(Y,N)";A$: IF A$ = "N" THEN 60
30  PRINT " 0": REM -INPUT DATA FROM BIT PAD
40  FOR J = 0 TO N - 1:BP =  USR (0):X%(0,J) = X1%:X%(1,J) = Y1%
50  PRINT J;X1%;Y1%: NEXT
60  INPUT "TAPE READY (Y,N)";A$: IF A$ = "N" THEN MN =  USR (5)
70  INPUT "SAVE(S)OR LOAD(L)";A$: IF A$ = "L" THEN 110
80  WO =  USR (1): REM -OPEN WRITE FILE
90  FOR J = 0 TO N - 1: PRINT X%(0,J);",";X%(1,J): NEXT
100 WC =  USR (2):MN =  USR (5): REM -CLOSE WRITE FILE
110 RO =  USR (3): REM -OPEN READ FILE
120  FOR J = 0 TO N - 1: INPUT X%(0,J),X%(1,J): NEXT
130  RC =  USR (4): REM -CLOSE READ
140  INPUT "VERIFY LOAD(Y,N)";A$: IF A$ = "N" THEN  END
150  FOR J = 0 TO N - 1: PRINT J;X%(0,J);X%(1,J): NEXT
```

Instructions for listing 2.

```
BPSAV—ROUTINES CALLED BY AIM BASIC
USR( ) TO:
-GET X, Y-COORDINATE PAIRS FROM BIT
PAD DIGITIZER
-SAVE BASIC ARRAYS AS DATA FILES ON
CASSETTE TAPE, OR DISK
-LOAD BASIC DATA FILES FROM CASSETTE
TAPE OR DISK INTO
BASIC ARRAYS
BIT PAD IS SET FOR:
-POINT MODE OUTPUT
-BAUD RATE = 1200(RS-232)
-BINARY DATA FORMAT

ARGUMENT OF USR( )IS USED TO FIND
SUBROUTINES

-USR(0),GTDATA—GETS 5 BYTES FROM BIT
PAD
FORMAT OF 1ST BYTE:
  0100 0100 (FLAGS)
WHEN THIS IS DETECTED, NEXT 4 BYTES
ARE STORED AT DATA,X:
  00XX XXXX (0-5)
  00XX XXXX (6-11)
  00YY YYYY (0-5)
  00YY YYYY (6-11)
THEIR FORMAT IS CHANGED:
  XXXX XXXX (0-7)
  0000 XXXX (8-11)
  YYYY YYYY (0-7)
  0000 YYYY (8-11)
```

file(s), or for computation. In the BASIC demo program (listing 1), I have used an integer array to receive the data, because this requires only 2 bytes for each element, which is enough for the 12-bit accuracy of the Bit Pad data.

The routines for saving and loading data, in listing 2, have some features in common with programs which have been published (Bresson, 1980; Flynn, 1979, 1980; Kvaal, 1980). I have tried to put as much of the coding as possible into assembly language, so as to simplify BASIC programming. A BASIC program to save and/or load data, such as listing 1, must POKE the starting address of the assembled program ($3F00 in this case). Then the USR(W) function is used to call the routines for saving and loading, with the argument of USR(W) serving as a pointer into a jump table, where the address of the desired routine is found.

The monitor subroutines, WHEREO and WHEREI are called to open files for saving and loading. These give the standard AIM prompts for device and file name, allowing a choice to be made between tape cassette or floppy disk as the recording medium. Saving on tape is in response to OUT=T, loading in response to IN=T. I have the Compas Microsystems DAIM disk operating system which uses the user hook, U, so that, with it, the dialog is OUT=U or IN=U. Some modification of the program might be needed with another disk system, or perhaps for paper tape.

To save an array which has been defined by a BASIC program, and which contains data, BASIC opens a write file with USR(1), executes a FOR loop containing the appropriate PRINT statement(s), then closes the file with USR(2). Loading a data file into an array is done in the same way, with USR(3) to open a read file, a FOR loop with INPUT statement(s), then USR(4) to close the file. Note that comma(s) must be inserted between variable names in the PRINT statement(s)! In using a cassette recorder, the tape must be positioned and the control keys operated manually; with the disk system, operation is, of course, much more automatic.

As Kvaal (1980) pointed out, attention must be given to the management of file size, to be sure that data files will fit into the arrays which have been defined to receive them. These routines can be used very flexibly. Data, or values computed from the data, can be saved by one program, and perhaps loaded by another program for further computation, plotting, etc. They are not limited to saving and loading integer values, as in the demo program.

```
AND THEY ARE MOVED TO BASIC
LOCATIONS, X1%, Y1%
ON RETURN TO BASIC, THESE MAY BE
STORED IN ARRAY(S)

BEFORE SAVING OR LOADING: POSITION
TAPE; TOGGLE RECORDER
(1)OFF; AND PLACE IT IN RECORD OR PLAY
MODE
OR INSERT DISK

-USR(1),OPENWR--SAVES PRINTER STATUS,
PROMPTS FOR DEVICE AND
FILE NAME; STARTS RECORDER OR DISK
BASIC PROGRAM SHOULD
THEN(PRINT)THE DESIRED ARRAY, AND
CALL:

-USR(2),CLOSWR--CLOSES THE FILE,
TURNS OFF THE RECORDER
OR DISK; AND RESTORES PRINTER STATUS

-USR(3),OPENRD--OPENS FILE, LIKE
OPENWR
BASIC SHOULD THEN(INPUT)DATA FILE
TO DESIRED ARRAY, THEN

-USR(4),CLOSRD--CLOSES FILE, LIKE
CLOSWR

-USR(5),MONTR--EXIT BASIC
```

```
                    ;***************************
                    ;*                         *
                    ;* AIM-65 BIT PAD ROUTINE  *
                    ;*                         *
                    ;*  BY RALPH O. ERICKSON    *
                    ;*                         *
                    ;***************************
                    ;*
                    ;MONITOR ADDRESSES
                    ;
                    COMIN  EQU $E1A1
                    DU11   EQU $E50A
                    WHEREI EQU $E848
                    WHEREO EQU $E871
                    LL     EQU $E8FE
                    RCHEK  EQU $E907
                    CRLF   EQU $E9F0
                    GETTTY EQU $EBDB
                    ;
                    ;I/O ADDRESSES
                    ;
                    GAP    EQU $A409
                    PRIFLG EQU $A411
                    INFLG  EQU $A412
                    OUTFLG EQU $A413
                    BAUD   EQU $A417
                    DRB    EQU $A800
                    ;
                    ;BASIC ADDRESSES
                    ;
                    VARPTR EPZ $75
                    BASACC EPZ $A9
                    IFIX   EQU $BEFE
                    ;
                    ;DAIM ADDRESS
                    ;
                    HEADUP EQU $9E10
                    ;
                    ;INTERNAL ADDRESS
                    ;
                    DATA   EPZ $E8              ;4 BYTES FOR DATA
                    ;
3F00                       ORG $3F00
3F00                       OBJ $800
3F00                ;
3F00                ;DECODE ARGUMENT OF USR()
3F00                ;
3F00 20FEBE         BPSAV  JSR IFIX
3F03 A5AC                  LDA BASACC+3
3F05 D012                  BNE RETURN
3F07 A5AD                  LDA BASACC+4
3F09 C906                  CMP #$06
3F0B B00C                  BCS RETURN
3F0D 0A                    ASL
3F0E 85AD                  STA BASACC+4
3F10 AA                    TAX
3F11 BD1B3F                LDA JTABL+1,X
3F14 48                    PHA
3F15 BD1A3F                LDA JTABL,X
3F18 48                    PHA
3F19 60             RETURN RTS
3F1A                ;
```

(Continued)

Listing 2 *(Continued)*

```
3F1A         ;JUMP TABLE
3F1A         ;
3F1A 263F    JTABL  ADR GTDATA-1
3F1C 793F           ADR OPENWR-1
3F1E 8D3F           ADR CLOSWR-1
3F20 AC3F           ADR OPENRD-1
3F22 B83F           ADR CLOSRD-1
3F24 C83F           ADR MONTR-1
3F26         ;
3F26 00      PSTAT  BYT $00            ;PRINTER STATUS
3F27         ;
3F27         ;GET DATA FROM BIT PAD INTO BASIC X1%,Y1%,USR(0)
3F27         ;SET BAUD RATE=1200
3F27         ;
3F27 A902    GTDATA LDA #$02
3F29 8D17A4         STA BAUD
3F2C A9FD           LDA #$FD
3F2E 8D18A4         STA BAUD+1
3F31         ;WHEN STYLUS IS DEPRESSED, GET 1ST BYTE
3F31 20DBEB         JSR GETTTY
3F34 C944           CMP #%01000100
3F36 2007E9         JSR RCHEK
3F39 D0EC           BNE GTDATA
3F3B A200           LDX #$00
3F3D         ;GET 4 DATA BYTES
3F3D 20DBEB  GET    JSR GETTTY
3F40 95E8           STA DATA,X
3F42 E8             INX
3F43 E004           CPX #$04
3F45 D0F6           BNE GET
3F47 A200           LDX #$00
3F49         ;REMOVE 2 HIGH BITS OF LOBYTE
3F49 16E8    SHIFT  ASL DATA,X
3F4B 16E8           ASL DATA,X
3F4D         ;ROTATE BOTH BYTES RIGHT WITH CARRY
3F4D 76E9           ROR DATA+1,X
3F4F 76E8           ROR DATA,X
3F51 76E9           ROR DATA+1,X
3F53 76E8           ROR DATA,X
3F55         ;CLEAR 4 HIGH BITS
3F55 B5E9           LDA DATA+1,X
3F57 290F           AND #%00001111
3F59 95E9           STA DATA+1,X
3F5B E8             INX
3F5C E8             INX
3F5D E004           CPX #$04
3F5F D0E8           BNE SHIFT
3F61         ;MOVE DATA TO BASIC LOCATIONS X1%,Y1%
3F61 A002           LDY #$02
3F63 A200           LDX #$00
3F65 B5E9    STXY   LDA DATA+1,X
3F67 9175           STA (VARPTR),Y
3F69 C8             INY
3F6A B5E8           LDA DATA,X
3F6C 9175           STA (VARPTR),Y
3F6E 98             TYA
3F6F 18             CLC
3F70         ;OFFSET FOR Y1%
3F70 6906           ADC #$06
3F72 A8             TAY
3F73 E8             INX
3F74 E8             INX
3F75 E004           CPX #$04
3F77 D0EC           BNE STXY
3F79 60             RTS
3F7A         ;
3F7A         ;OPEN WRITE FILE-USR(1)
3F7A         ;
3F7A A920    OPENWR LDA #$20
3F7C 8D09A4         STA GAP
3F7F         ;SAVE PRINTER STATUS
3F7F AD11A4         LDA PRIFLG
3F82 8D263F         STA PSTAT
3F85         ;TAPE OR DISK?
3F85 2071E8         JSR WHEREO
3F88         ;PRINTER OFF
3F88 A900    PROFF  LDA #$00
3F8A 8D11A4         STA PRIFLG
3F8D 60             RTS
3F8E         ;
```

(Continued)

For some purposes it would be preferable to operate a Bit Pad in stream mode rather than in point mode. This would let you trace an outline quickly while the Bit Pad transmits data continuously to the AIM. It might be preferable to use the 8-bit parallel interface for this. I have used a Bit Pad with the parallel interface (see Coyle, this issue) on a trial basis, and have a preliminary program to decode and store coordinate pairs in this mode. It would probably be best to use this as a subroutine called in a machine language program, because of speed limitations inherent in BASIC. You might want additional routines to find such things as maxima, minima, arc lengths, or areas, returning to a BASIC calling program only with such computed values, rather than with the raw data.

I want to thank my associates, Jim Laurino and Lee Peachey for advice.

References

1. Bresson, Steve. 1980. "AIM 65 BASIC Save/Load Scheme." *6502 User Notes*, No. 17, p. 20.

2. Flynn, Christopher. 1979. "Some Important BASIC Mods." *6502 User Notes*, No. 15, pp. 9-12.

3. Flynn, Christopher. 1980. "AIM 65 File Operations." MICRO, No. 26, pp. 61-66.

4. Kvaal, Knut. 1980. "AIM BASIC Files." *The Target*. January/February 1980, pp. 2-3.

5. Saltero, Richard. 1980. "BCD Input to a 6502 Microprocessor." MICRO, No. 27, pp. 68-70.

Ralph O. Erickson is a Professor of Botany at the University of Pennsylvania, and the author of a number of articles in scientific journals. Since 1964, he has had experience with several computers in connection with his research (IBM 7040, 360, 370; CDC 3600; PDP 10; H-P 9830; Tektronix 5041). Currently, he is enthusiastic about the potential and convenience of microcomputers, such as the AIM 65, for applications in scientific research. He also uses his AIM for recreation, such as playing music.

MICRO

Listing 2 *(Continued)*

```
3F8E                  ;CLOSE WRITE FILE-USR(2)
3F8E                  ;
3F8E 20F0E9           CLOSWR JSR CRLF
3F91 20F0E9                  JSR CRLF
3F94 200AE5                  JSR DU11
3F97 AD13A4                  LDA OUTFLG
3F9A C955                    CMP 'U
3F9C                  ;CLOSE DISK FILE
3F9C F008                    BEQ PRSTAT
3F9E                  ;TURN OFF RECORDERS
3F9E A9CF             RECOFF LDA #$CF
3FA0 2D00A8                  AND DRB
3FA3 8D00A8                  STA DRB
3FA6                  ;RESTORE PRINTER STATUS
3FA6 AD263F           PRSTAT LDA PSTAT
3FA9 8D11A4                  STA PRIFLG
3FAC 60                      RTS
3FAD                  ;
3FAD                  ;OPEN READ FILE-USR(3)
3FAD                  ;
3FAD AD11A4           OPENRD LDA PRIFLG
3FB0                  ;SAVE PRINTER STATUS
3FB0 8D263F                  STA PSTAT
3FB3                  ;TAPE OR DISK7
3FB3 2048E8                  JSR WHEREI
3FB6                  ;PRINTER OFF
3FB6 4C883F                  JMP PROFF
3FB9                  ;
3FB9                  ;CLOSE READ FILE-USR(4)
3FB9                  ;
3FB9 AD12A4           CLOSRD LDA INFLG
3FBC C955                    CMP 'U
3FBE D003                    BNE REC1
3FC0 4C109E                  JMP HEADUP
3FC3 20FEE8           REC1   JSR LL
3FC6 4C9E3F                  JMP RECOFF
3FC9                  ;
3FC9                  ;RETURN TO MONITOR-USR(5)
3FC9                  ;
3FC9 4CA1E1           MONTR  JMP COMIN
                            END
```

MICRO PET Vet

By Loren W. Wright

I had planned to do this column as a comparison of assemblers for 8K PETs. However, I have determined that there is now only one widely available. Personal Software withdrew its "Assembler in BASIC" last fall, so the remaining one is the newly-released HESEDIT/ HESBAL from Human Engineered Software.

The editor (HESEDIT), which can be useful for editing files other than assembly language source, is page-oriented. Operation revolves around the command line at the top of the screen, where commands are entered that manipulate the file with respect to the 22-line display window. Other commands, like Insert, Delete, and Replicate, are entered in the numbered (or command) portion of each line. It is very easy to make changes anywhere in the editor file. Also, a file larger than the memory available can be manipulated. Other commands save and load files on tape or disk.

The assembler (HESBAL), written in BASIC, is understandably slow. It does the job, though, and you can assemble to any available place in memory you wish (not just the second cassette buffer). Also, it is easy to make corrections at the time of assembly. All you need do is type a line (which includes the corrected source line) in the immediate mode, and you're back in business!

Probably the best part of the package is the documentation. As part of the "human engineered" concept, a full BASIC listing and program description are included. The manual suggests a number of possible changes to suit individual needs. These include accommodating a printer and assembly in the immediate mode, without a previously prepared editor source file. As a service to its customers, a copy of the public domain Micromon, an enhanced PET monitor by Bill Seiler, is included.

The slow speed of the assembler is a function of BASIC *vs.* machine language. A machine language assembler would have taken longer to develop, and hence would cost a lot more. It also would be difficult to change. The limited power (there are only four pseudo-ops) of the assembler is also a function of BASIC. There's only so much that can be put into a program for an 8K PET and still leave room enough for the source, object, and symbol table.

The assembler does not print the object as it assembles — only the program counter and source line. I'm not sure whether this deficiency can be corrected with a simple patch. My review copy of the assembler mistakenly rejected the "absolute, indexed by Y" mode. This can be corrected with the addition of a single BASIC line, and I assume the current version includes this change.

Human Engineered Software's HESEDIT/HESBAL is a very usable editor/assembler for 8K PETs. As the *only* such package currently widely available, it has filled a void in the market. Owners of larger PETs might consider this over faster, more powerful, but considerably more expensive packages. The well-documented BASIC program is easy to change to fit a number of special needs.

HESEDIT is available in three versions — one for each ROM set — for $12.95 on tape or diskette. HESBAL, with HESEDIT, is $23.95.

Symbolic Assembler for HESEDIT/ HESBAL

Before I stray too far from this subject, I should mention that Emil Volcheck has made changes in Werner Kolbe's Symbolic Disassembler (MICRO 32:23) to make it compatible with HESEDIT/HESBAL. Other changes include a greater "user-friendliness" and an additional disk filing routine. He is willing to supply a cassette copy, with listing, for $5.00 postpaid.

Emil J. Volcheck, Jr.
1046 General Allen Lane
West Chester, Pennsylvania 19380

BASIC Upgrade Update

In my overview of BASIC upgrades (MICRO 36:62), I neglected to point out that Palo Alto ICs offers an inexpensive way to upgrade to its 4.0 Toolkit. Send them your current Toolkit ROM, with a check for $22.45 postpaid, and you will receive a 4.0 version for a lot less than the $39.95 new purchase price.

Palo Alto ICs
2585 E. Bayshore Road
Palo Alto, California 94303

Name Change

Commodore Interface is the new name for the *Commodore Newsletter* of the PET Users' Club. The first issue, under the editorship of Joe Devlin, includes a number of product announcements, (with a feature of the VIC-20), news items, a couple games, programming tips, and software and book reviews. Future issues will be larger, with the addition of advertising. Contributions are encouraged. The annual $15 subscription ($25, Canada and Mexico) covers six issues. For more information, contact:

The Editor
Commodore Interface
681 Moore Road
King of Prussia, Pennsylvania
19406

Micro-Mainframe — New from Commodore

Commodore has joined the 6809 bandwagon with the introduction of its Micro-Mainframe computer (also known as "Super PET"). A demonstration unit was exhibited at the Commodore booth at the National Computer Conference in Chicago, May 4-7. Actually, it is an 8032 with a 6809-based 64K expansion board, and yes, you will be able to upgrade an existing 8032. The Micro-Mainframe will support interpreted versions of BASIC, Pascal, FORTRAN, APL, and soon, COBOL, all developed at the University of Waterloo, Waterloo, Ontario.

The Micro-Mainframe can operate as a stand-alone microcomputer, supporting all CBM/PET software and hardware (except C2N cassette), or as a development system for larger and faster mainframe computers. The 6809 board includes a standard RS-232C interface, and files are output in true ASCII, a form compatible with the mainframe computers.

The $1995 price will include the 8032 computer, 6809 board, and software, notably the "Waterloo 6809 Assembler and Linker." Deliveries are scheduled for late 1981.

OHIO SCIENTIFIC'S
SMALL SYSTEMS JOURNAL

This month's journal presents the conclusion of "User-Defined Routines in UCSD Pascal" by D.R. Turnidge.

F. PROGRAM SPECIALDEMO

This section contains a sample Pascal program which illustrates the use of the procedures in UNIT SPECIALFEATURES. The procedures from the newly installed UNIT SPECIALFEATURES will automatically be linked into the workfile when it is run.

```
(*$L CONSOLE:*)
PROGRAM SPECIALDEMO;

 USES SPECIALFEATURES;

 VAR CHARNUM,XCOOR,YCOOR,COUNT,
       LEFT,RIGHT,TOP,BOTTOM: INTEGER;
      COLOR,COLOR2: COLORS;

 PROCEDURE DELAY(TIME: INTEGER) ;
  VAR COUNT1,COUNT2: INTEGER;
  BEGIN
   FOR COUNT1: = 1 TO TIME DO
    FOR COUNT2: = 1 TO 50 DO (* WAIT A WHILE *) ;
  END;

 PROCEDURE WHISTLE;
  VAR FREQUENCY,INC: INTEGER;
  BEGIN
   SOUNDON; (* TURN SOUND OPTION ON *)
   FREQUENCY: = 256;
   FILLCOLOR(BLUE);
   XCOOR: = 0; YCOOR: = 1; INC: = 1;
   REPEAT
    TONE(FREQUENCY);
    FREQUENCY: = FREQUENCY + 2;
    PLOTCOLOR(INVBLUE,XCOOR,YCOOR) ;
    IF INC = 1 THEN
     IF XCOOR< 31 THEN
      XCOOR: = XCOOR + INC
     ELSE
      BEGIN
       INC: = -1;
       YCOOR: = YCOOR + 1;
      END
    ELSE
     IF XCOOR>0 THEN
      XCOOR: = XCOOR + INC
     ELSE
      BEGIN
       INC: = 1;
       YCOOR: = YCOOR + 1;
      END;
   UNTIL FREQUENCY = 2048;
   INC: = -1;
   REPEAT
    TONE(FREQUENCY) ;
    FREQUENCY: = FREQUENCY - 2;
    PLOTCOLOR(BLUE,XCOOR,YCOOR) ;
    IF INC = 1 THEN
     IF XCOOR< 31 THEN
      XCOOR: = XCOOR + INC
     ELSE
      BEGIN
       INC: = -1;
       YCOOR: = YCOOR - 1;
      END
    ELSE
     IF XCOOR>0 THEN
      XCOOR: = XCOOR + INC
     ELSE
      BEGIN
       INC: = 1;
       YCOOR: = YCOOR - 1;
      END;
  UNTIL FREQUENCY = 256;
 END;


BEGIN (* PROGRAM SPECIALDEMO *)
 INITOPTIONS; (* INITIALIZE OPTIONS *)
 CLEARGRAPHICS; (* CLEAR GRAPHICS DISPLAY *)
 CLEARCOLOR; (* CLEAR COLOR DISPLAY *)
 COLORON; (*TURN COLOR OPTION ON*)
 COLOR: = YELLOW;
 FOR CHARNUM: = 0 to 47 DO
  BEGIN
   FILLGRAPHICS(CHARNUM) ;
   SCR32 x 64;
   FILLCOLOR(COLOR) ;
   DELAY(25) ;
   COLOR: = SUCC(COLOR) ;
   SCR32 x 32;
   FILLCOLOR(COLOR) ;
   DELAY(25) ;
   COLOR: = SUCC(COLOR) ;
  END;
 CLEARGRAPHICS;
 COLOR2: = YELLOW;
 REPEAT
  FILLCOLOR(COLOR2) ;
   (* DISPLAY COLOR CHECKBOARD SPIRALING OUT *)
   LEFT: = 15; RIGHT: = 16; BOTTOM: = 15; TOP: = 16;
   REPEAT
    FOR YCOOR: = BOTTOM TO TOP DO
     BEGIN
      PLOTCOLOR(COLOR,LEFT,YCOOR) ;
      COLOR: = SUCC(COLOR) ;
     END;
    FOR XCOOR: = LEFT + 1 TO RIGHT DO
     BEGIN
      PLOTCOLOR(COLOR,XCOOR,TOP) ;
      COLOR: = SUCC(COLOR) ;
     END;
    FOR YCOOR: = TOP - 1 DOWNTO BOTTOM DO
     BEGIN
      PLOTCOLOR(COLOR,RIGHT,YCOOR) ;
      COLOR: = SUCC(COLOR) ;
     END;
    FOR XCOOR: = RIGHT - 1 DOWNTO LEFT + 1 DO
     BEGIN
      PLOTCOLOR(COLOR,XCOOR,BOTTOM) ;
      COLOR: = SUCC(COLOR) ;
     END;
    LEFT: = LEFT - 1; RIGHT: = RIGHT + 1;
    BOTTOM: = BOTTOM - 1; TOP: = TOP + 1;
   UNTIL LEFT = 2;
   (* DISPLAY GRAPHICS CHARACTERS SPIRALING IN
   LEFT: = 3; RIGHT: = 28; TOP: = 28; BOTTOM: = 3;
   CHARNUM: = 0;
   REPEAT
    FOR YCOOR: = BOTTOM TO TOP DO
     BEGIN
      PLOTCHARACTER(CHARNUM,LEFT,YCOOR) ;
      CHARNUM: = CHARNUM + 1;
     END;
```

OHIO SCIENTIFIC 1333 S. Chillicothe Road • Aurora, Ohio 44402 • (216) 831-560

```
FOR XCOOR: = LEFT + 1 TO RIGHT DO
 BEGIN
  PLOTCHARACTER(CHARNUM,XCOOR,TOP) ;
  CHARNUM: = CHARNUM + 1;
 END;
FOR YCOOR: = TOP - 1 DOWNTO BOTTOM DO
 BEGIN
  PLOTCHARACTER(CHARNUM,RIGHT,YCOOR) ;
  CHARNUM: = CHARNUM + 1;
 END;
FOR XCOOR: = RIGHT - 1 DOWNTO LEFT + 1 DO
 BEGIN
  PLOTCHARACTER(CHARNUM,XCOOR,BOTTOM) ;
  CHARNUM: = CHARNUM + 1;
 END;
LEFT: = LEFT + 1; RIGHT: = RIGHT - 1;
TOP: = TOP - 1; BOTTOM: = BOTTOM + 1;
UNTIL LEFT = 16;
DELAY(50) ;
(* ERASE GRAPHICS CHARACTERS SPIRALING OUT *)
LEFT: = 15; RIGHT: = 16; BOTTOM: = 15; TOP: = 16;
REPEAT
 FOR XCOOR: = LEFT TO RIGHT DO
  ERASECHARACTER(XCOOR,BOTTOM) ;
 FOR YCOOR: = BOTTOM + 1 TO TOP DO
  ERASECHARACTER(RIGHT,YCOOR) ;
 FOR XCOOR: = RIGHT - 1 DOWNTO LEFT DO
  ERASECHARACTER(XCOOR,TOP) ;
 FOR YCOOR: = TOP - 1 DOWNTO BOTTOM + 1 DO
  ERASECHARACTER(LEFT,YCOOR) ;
 LEFT: = LEFT - 1; RIGHT: = RIGHT + 1;
 TOP: = TOP + 1; BOTTOM: = BOTTOM - 1;
UNTIL LEFT = 2;
(* ERASE COLORS SPIRALING IN *)
LEFT: = 3; RIGHT: = 28; TOP: = 28; BOTTOM: = 3;
REPEAT
 FOR XCOOR: = LEFT TO RIGHT DO
  ERASECOLOR(XCOOR,BOTTOM) ;
 FOR YCOOR: = BOTTOM + 1 TO TOP DO
  ERASECOLOR(RIGHT,YCOOR) ;
 FOR XCOOR: = RIGHT - 1 DOWNTO LEFT DO
  ERASECOLOR(XCOOR,TOP) ;
 FOR YCOOR: = TOP - 1 DOWNTO BOTTOM + 1 DO
  ERASECOLOR(LEFT,YCOOR) ;
 LEFT: = LEFT + 1; RIGHT: = RIGHT - 1;
 TOP: = TOP - 1; BOTTOM: = BOTTOM + 1;
UNTIL LEFT = 16;
COLOR2: = SUCC(SUCC(COLOR2)) ;
COLOR: = SUCC(COLOR) ;
NTIL COLOR2 = OLIVE;
LEARGRAPHICS;
HISTLE;
NITOPTIONS; (* REINITIALIZE OPTIONS *)
ND.
```

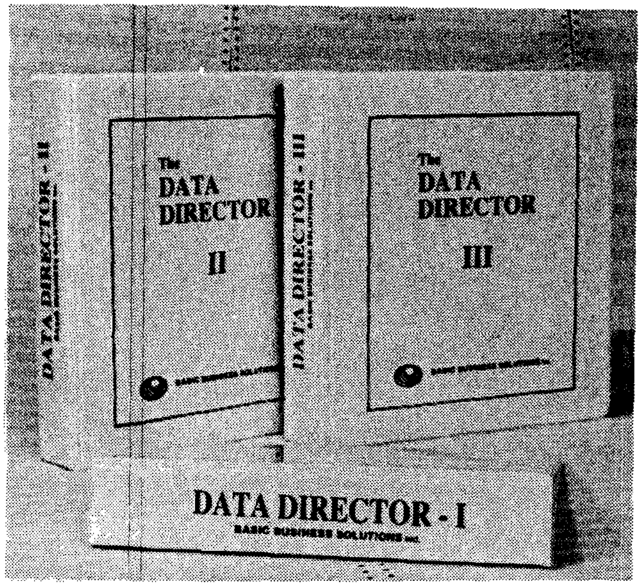
ibliography

. Bowles, Kenneth L., *Beginner's Guide to the UCSD Pascal System,* Peterborough: Byte Books, 1980.

. Fox, David & Waite, Mitch, *Pascal Primer,* Indianapolis: SAMS.

. *UCSD Pascal User's Manual,* San Diego: Softech microsystems, 1978.

. *UCSD Pascal Supplemental User's Document for Use with the Ohio Scientific C3, C4 and C8,* San Diego: Softech microsystems, 1980.



The Data Director

easy to use Professional Software for easy to get Professional Results

THE DATA DIRECTOR PROJECT

The Data Director is a file management system designed for associations, direct mailers, sales & marketing departments, recruiters, and companies that have the following profile:

- Maintain 1 to 5 files on paper or through a service bureau.
- *Files are the heart of your operation* (membership lists, subscription lists, sales inquiries, resumes, etc.).
- First time computer user.
- Staff characterized by high turnover.
- One major task associated with each file. Renewals, mailing labels, inquiry response, regular peak periods, etc.

The system is based upon work that began on Burroughs mainframe computers in the mid-1970s. In 1979 it was installed on OSI computers at four carefully selected alpha-test sites. These sites ranged from a church in a Gary, Indiana ghetto to the fourth-ranked graduate business college in the nation. One site, a direct mail service in Chicago, entered over 300,000 names and addresses into their system during 1979 and 1980.

Before a line of code was written, we profiled the average operator. He/she was a high school graduate, an average typist, low paid, low seniority, filling a high-turnover position. How could we work with this person?

We began by isolating the disk operating system, thereby reducing the training time and the chances of a costly mistake. The system includes a diskette formatter, file-to-file copier, directory report, file create, file rename, file delete, disk packer, and diskette copier. It traps disk errors and displays error messages in English. For example, an open disk drive prompts the message, "Drive door open. Close door and try again."

Next we developed a machine language terminal controller to simplify data entry. Displays are paged, not scrolled. Records are presented as forms automatically. If your terminal offers it, we use full and half intensity to highlight data, and cursor control keys to move around the display. We emulate all the features found on the most expensive terminals—character insert/delete, forward/reverse tab, field erase, strike-over, rubout, etc. Existing data is edited, not retyped.

The bottom line of each display is reserved as a status and command line. The operator uses a vocabulary of 30 English command words to begin each task. For example, ADD adds a new record to a file, REBUILD reconstructs an existing file into a new format. SORT sorts a file on up to 5 keys in ascending or descending order for each key.

Help is available on-line through a HELP dictionary which explains the purpose and function of each command in the vocabulary.

A 225-page manual thoroughly documents system operation, and supplies additional information to get you started properly.

HARDWARE REQUIREMENTS

Computers: C2-OEM, C2-D, and all C3 models running under the OS-65U operating system. Data Director II and III support up to 8 users.

Terminals: All 24 by 80 cursor addressable terminals are supported. Models are selected from a menu. Multi-user systems may use different terminal models at each station.

Printers: All printers supported by the OS-65U V1.2 DOS. Printers are selected from a menu. Line and page lengths are establish globally.

THE DATA DIRECTOR I

Our base system is optimized for floppy disk systems. Records may contain up to 99 fields of information. Each field may contain up to 71 characters. Three field types are recognized alpha, numeric, and MM/DD/YY dates.

Existing OS-DMS compatible files can be read and maintained by the system (although the reverse is not true). We hope that OS-DMS users will consider upgrading to our system.

The REPORTS command offers an inquiry report that can be sent to the console or printer, a mailing label generator, and a conditional report writer with statistical analysis. All reports, and most of the utilities, feature a program halt on CTRL-C which allows you to halt the report and abort or continue at your leisure.

THE DATA DIRECTOR II

Although it runs on a floppy disk system, our second system is optimized for a hard disk system. It supports up to 8 users (16 upon request), and was designed for files up to 20,000 records long. All version I features are incorporated.

The operating system utilities are extended to include a fast floppy dumper to back up hard disk files to floppy diskette. (Mag tape support is available separately.)

A duplicates report scans files for duplicate records. As an option, it can count all the occurrences of a duplicate field, like breaking down zipcode distributions.

The report saver captures report definitions and saves them by name. Our users have defined reports with exotic names like "In Work," "Delinquents," "Approved Loans," "Past Due," and "Prod Work Orders." The reports offer conditional selec tion and statistical analysis.

The mailing label generator is expanded into ; complete subsystem aimed at professional mailers The operator defines a label definition, giving it ; name, the label's size, fields which are to appear o it, messages like "After 5 days return to:" and eve default values like "Occupant" that are to appear i the data is missing. By selecting a definition, th operator can print on pressure sensitive labels envelopes, 3 by 5 cards, stationery, etc. Prin options include printing labels 1 to 5 across an repeating labels up to 99,999 times.

THE DATA DIRECTOR III

Our top of the line system is designed for ver large files, 20,000 to 100,000 records long. I incorporates all of the features of versions I and I and adds a new "linked list" storage technique.

From the operator's viewpoint, a linked fil appears to be resorted automatically whenever record is added or deleted. A file can be ordered i up to 5 different ways. For example, you coul order a membership roll by (1) zipcode, (2) zipcod and name, (3) name, (4) renewal date, and (5) se age, and marital status.

Seek times are vastly reduced. You could fin the first occurrence of renewal date 11/01/81 in 100,000 record file in under 6 seconds, the switch keys and browse forward or backwar through the file by name.

CONSIDER THESE FACTS

It costs money to put data into a computer. As rule of thumb, expect to pay $125 per 100 records in labor just to gather and key in your dat Ask any data processing professional; in less tha two years your data will be worth more than th cost of your computer and software combined. Th is without consideration of cost savings or ac ditional income you can generate with the use c this information. You need the best software on th market to look after that asset.

Data Director I	$ 495.00
Data Director II	$ 995.00
Data Director III	$1995.00
Manuals	$ 30.00

Dealer inquiries invited.

Basic Business Solutions, Inc.
831 N. 3rd Ave.
St. Charles, IL 60174
Bus. Offices (312) 377-0813
Support Center (312) 746-8736

Please take a moment to complete this questionnaire. It will help MICRO and its advertisers to serve you better.

You

A. Age: ________ B. Occupation: ________

C. Professional Computer Experience: ________

D. Computer Courses/Training: ________

E. Microcomputer Hardware Level: Novice ________ Intermediate ________ Expert ________

F. Microcomputer Software Level: Novice ________ Intermediate ________ Expert ________

Your System(s)

G. Indicate which systems you have access to on a regular basis by placing an "H" for Home, "W" for Work, "O" for Other.

AIM ________ Apple ________ Atari ________ KIM ________ OSI ________ PET/CBM ________ SYM ________

Other 6502 Microcomputers (list): ________ 6809 Microcomputers: ________

H. Please answer for your own personal system(s): Type: ________

RAM Memory: ________K ROM/EPROM: ________K Printer: ________

Disk drives: 5¼": ________ 8": ________ Video Monitor: ________ Modem: ________

Other Peripherals: ________

I. Estimate dollars you will spend in coming year for hardware: $ ________ For: ________

Software and Applications

J. Estimate percent of software regularly used in each language:

Assembly: ________% BASIC: ________% Pascal: ________% FORTH: ________% Other: ________%

K. Estimate percent of time spent in each type of application:

Games: ________% Education: ________% Business: ________% Programming: ________%

Scientific: ________% Engineering: ________% Other: ________%

L. Estimate dollars you will spend in coming year for software: $ ________ For: ________

M. How many hours per month do you spend in the following microcomputer activities?

Using Computer: ________ Programming: ________ Reading Computer Material: ________

Computer Clubs: ________ Computer Store: ________ Computer Shows: ________

Other Computer Activities: ________

You and MICRO

N. Rate the value of MICRO departments on a scale of 1 (most valuable) to 5 (least):

Bibliography: ________ Club Circuit: ________ New Publications: ________

Software Catalog: ________ Hardware Catalog: ________ PET Vet (Column): ________

Micros in Medicine (Column): ________ Challenges (OSI Column): ________ Editorial: ________ Letterbox: ________

O. Rate your preference for types of articles from 1 (most valuable) to 5 (least):

General Hardware: ________ Microcomputer-specific Hardware: ________

General Software: ________ Microcomputer-specific Software: ________

Applications: ________ Tutorials: ________ Programming Techniques: ________

BASIC Programs: ________ Assembly Programs: ________ Pascal Programs: ________

Other: ________

P. What kind of 6809 coverage would be most useful to you? ________

Q. Where did you get your current copy of MICRO? Subscription: ________

Computer Store: ________ Library: ________ Computer Club: ________ Borrowed: ________

R. If you are a subscriber, when did your subscription start? ________ / ________

S. If you buy MICRO at a computer store, how regularly do you buy it?

Monthly: ________ Occasionally: ________ Rarely: ________

T. What would make MICRO more valuable and/or interesting?

____

U. Please indicate what other microcomputer magazines you normally read:

S = Subscriber, P = Purchase regularly, O = Buy occasionally

BYTE ________ Compute! ________ Creative Computing ________ Interface Age ________

KB Microcomptuing ________ Nibble ________ On Computing ________ Personal Computing ________

Others (list): ________

V. Please use this space and space on back for any additional comments and/or suggestions. ________

Return address not necessary.

Please fold here.

PLACE
STAMP
HERE

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
USA

MICRO Classified

Timestack — A Programmable Controller

Expand your KIM-1 into a general-purpose machine. 80-page manual documents Clock/Port/RAM/PROM Expansion Board and controller software. Subroutine library includes user interaction routines, I/O, and clock controls. Complete manual — $15.00. SASE for more information and newsletter.

Hunter Services
P.O. Box 359
Elm Grove, Wisconsin 53122

Atari Game Owners

Turn your Video Game Console into a 6502 microcomputer with our MagiCard. Write programs using your keyboard controllers, with full access to Atari video capabilities. Includes 1K bytes RAM, ROM monitor, disassembler, instruction manual, cassette interface plans. Send $49.88 (Illinois add 6%) to:

Computer Magic Inc.
P.O. 3383M
Fox Valley Center
Aurora, Illinois 60505

Turnkey Medical Billing System

Interactive data entry. Automated file management. Outputs: Patient statements, Universal Claim Forms, financial reports. Customized by user-developed text files. Requires Apple, Applesoft, printer. One disk drive manages 150 accounts; 2 drives—400 accounts. $350 for programs and 25 pages documentation.

Jerome B. Blumenthal, M.D.
7500 E. Hellman
Rosemead, California 91770

Used Micro Listing Service

Save time, money, mistakes, frustration. Buyer/Seller — Apple, PET, OSI, CP/M systems, floppies, printers — all equipment $300 and up. Pay only for results. Get thorough advice and listings over the phone. Call now: 800-327-9191 x 61 or 703-471-0044.

Used Computer Exchange
2329 Hunters Woods Plaza
Reston, Virginia 22091

PET/CBM Owners

Real world software at low cost. 2114 RAM adapter and 4K Memory Expansion for "old" 8K PETs. Write for free catalog!

Optimized Data Systems
Dept. M, Box 595
Placentia, California 92670

Spanish Hangman

2,000 SPANISH words and sentences taught in a fun way on the Apple. Send for your school's free 30-day evaluation diskette, from:

George Earl
1302 South General McMullen
San Antonio, Texas 78237

AIM-65 Newsletter * * Target

Target provides hardware and software information useful for AIM-65 and 6502 users. The 1979 and 1980 back issues are available for $12.00 while a continuing subscription costs $6.00. Just write to:

Target
Donald Clem
Route 2
Spenserville, Ohio 45887

C1P Extended Monitor

2K EPROM has 14 cursor control/editing functions, improved keyboard decoding. Machine language save, load, display, modify, move, breakpoint processing and much more. For 24, 32, 64 char/line. $39.95 plus $1.00 shipping. $1.00 for complete information.

Bustek
P.O. Box A
St. Charles, Missouri 63301

Ohio Scientific C1P, C4P COLOR

Earthship has GREAT programs. C1P, C4P — Animated Lunar Lander, Catchword, real-time Scrabble, graphics designer, analytical plotter, single disk copier; C1P — animation and shape table graphics, BASIC tutor, add and multiply tutor, information processing simulation and tutor. Send for catalog.

Earthship
17 Church Street #28
Nutley, New Jersey 07110

we carry it all....

everything for Commodore and Atari

Atari® Software

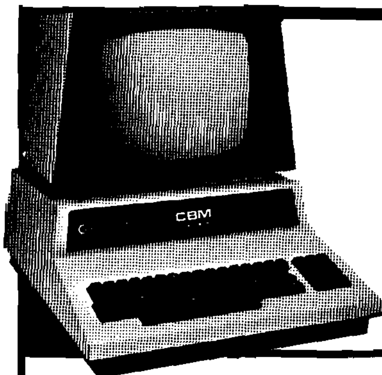
VisiCalc	149
CX4101 Invitation to Programming 1	17
CX4104 Mailing List	17
CX4102 Kingdom	13
CX4103 Statistics	17
CX4105 Blackjack	13
CX4106 Invitation to Programming 2	20
CX4107 Biorhythm	13
CX4108 Hangman	13
CX4109 Graph It	17
CX4111 Space Invader	17
CX4110 Touch Typing	20
CX4115 Mortgage & Loan Analysis	13
CX4116 Personal Fitness Program	13
CX4117 Invitation to Programming 3	20
CX4118 Conversational French	45
CX4119 Conversational German	45
CX4120 Conversational Spanish	45
CX4121 Energy Czar	13
CX4125 Conversational Italian	45
CX8108 Stock Charting	20
CXL4001 Educational System Master	21
CXL4002 Basic Computing Language	46
CXL4003 Assembler Editor	46
CXL4004 Basketball	30
CXL4005 Video Easel	30
CXL4006 Super Breakout	30
CXL4007 Music Composer	45
CXL4009 Chess	30
CXL4010 3-D Tic-Tac-Toe	30
CXL4011 Star Raiders	33
CXL4015 TeleLink	20
Talk & Teach Courseware: CX6001 to CX6017	23

Atari® Peripherals:

400 16K	$349
410 Recorder	59
810 Disk	469
815 Disk	1199
822 Printer	359
825 Printer	629
830 Modem	159
850 Interface Module	139

Atari® Accessories

CX853 16K RAM	89
CX70 Light Pen	64
CX30 Paddle	18
CX40 Joystick	18
CX86 Printer Cable	42
CO16345 822 Thermal Printer Paper	5
CAO16087 825 80-col. Printer Ribbon (3/box)	17
Microtek 16K RAM	79
Microtek 32K RAM	179

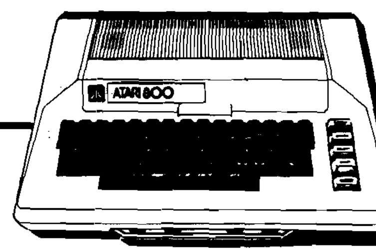


commodore

VIC-20	$ 279
4032N	1080
8032	1499
CBM 4022 Printer	669
CBM 4040 Drive	1039
CBM 8050 Drive	1449
CBM C2N Drive	87
PET-IEEE Cable	37
IEEE-IEEE Cable	46

Disks

Maxell Disks	10 for $36
Syncom Disks	10 for 29
Atari Disks	5 for 22



ATARI 800™ with 32K RAM only $759

Software

EBS Accounts Receivable Inventory System	$595
OZZ Information System	329
BPI General Ledger	329
Tax Package	399
Dow Jones Portfolio Management	129
Pascal	239
WordPro 3 (40 col.)	186
WordPro 4 (80 col.)	279
WordPro 4 Plus (80 col.)	339
Wordcraft 80	319

Printers

NEC 5530	2495
Diablo 630	2195
Trendcom 100	299
Starwriter	$1495
Trendcom 200	489
Paper Tiger 445G	769
Paper Tiger 460G	1219
Epson MX-80	499
Tally 8024	1699

Please Call Between 11AM & 6PM (Eastern Standard Time)

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

Prepaid Orders Receive Free Shipping

(800) 233-8950

* on all in stock units

Computer Mail Order **501 E. Third St., Williamsport, PA 17701 (717) 323-7921**

MICRO Hardware Catalog

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **Micro-Courier**
System: Apple II, Apple II Plus
Memory: 48K
Language: Integer BASIC or Applesoft
Hardware: Apple II or Apple II Plus, Monitor, Disk II with controller, DOS 3.3, DC Hayes Micromodem, a "clock card," printer with interface card.
Description: Allows owners of Apple II and Apple II Plus desktop computers to rapidly transmit charts, graphs, correspondence, VisiCalc® reports and entire programs to other Apple computer owners. The transmissions can be sent automatically, day or night, to take advantage of late night phone rates. Maintains phone lists and sorts messages by individual user. Exchanges data with time-sharing systems and larger computers.
Price: $250.00 (suggested retail, FOB Boston)
Available: Microcom, Inc.
89 State St.
Boston, Massachusetts 02110

Name: **Hayes Stack Smart-modem**
System: Machine independent — RS-232C compatible
Language: Program controlled in any language
Hardware: Low speed modem
Description: RS-232C compatible, 300 baud data communications system for small computers. Features program control in any language, switch selectable options, full or half duplex and LED status indicators.
Price: $279.00
Available: Hayes Microcomputer Products, Inc.
5835A Peachtree Corners East
Norcross, Georgia 30092
(404) 449-8791
(Contact above address for nearest retail dealer.)

Name: **Bytewriter-1**
Memory: One line buffer capacity
Language: BASIC
Description: 7 × 7 dot matrix printer, friction feed, 80 c.p.s., 60 l.p.m., interfaces Apple, Atari and TRS-80. 80-columns per line and double wide character set.
Price: $299.00
Available: Microtek, Inc.

Name: **The PEAR System**
System: Apple
Memory: 48K
Language: Applesoft
Hardware: Dual 5¼" disk drives, DC Hayes micromodem, 32-column printer.
Description: PEAR is a multiple portfolio recordkeeping and reporting system for stockbrokers. Its unique file structure means that securities information is entered only once and can be changed on all portfolios with a single entry. PEAR includes automatic pricing from Dow Jones, matching of proceeds and cost basis by tax lot, automatic adjustment of positions for stock splits, portfolio appraisals, unrealized gain and loss, realized gain and loss, investment income reports, and a full cross reference listing of client holdings by security.
Price: $500.00 includes documentation and program disk.
Author: Gregg Wilson
Available: PEAR Systems
27 Briar Brae Road
Stamford, Connecticut 06903

Name: **PSSBC-A**
System: AIM 65
Hardware: AIM 65 with BASIC and Assembler ROMs
Description: Power supply built to the specs for the AIM 65 including case power cord cable to computer, switch, fuse, pilot light, overvoltage protection.
Price: $64.95 plus shipping (5 lbs)
Available: CompuTech
Box 20054
Riverside, California 92516

Name: **Micromodem 100**
System: S-100 Bus Computers
Hardware: Low speed modem
Description: Direct connect data communications system for S-100 bus computers. Features 110 and 300 baud, full or half duplex and programmable auto dial and auto answer capabilities.
Price: $379.00
Available: Hayes Microcomputer Products, Inc.
5835A Peachtree Corners East
Norcross, Georgia 30092
(404) 449-8791
(Contact above address for nearest retail dealer.)

Name: **MEM 4 and MEM 8**
System: AIM 65
Memory: 4K and 8K
Description: This is a low-power memory board that is plug-compatible with the AIM 65 expansion connector and requires no motherboard or other hardware.
Price: $169.00 introductory price for MEM 8 and $109.00 introductory price for MEM 4.
Available: System Peripherals
P.O. Box 971, Dept. M
Troy, Michigan 48099

Name: **Datasouth DS180 Matrix Printer**
Description: 180 cps dot matrix impact printer; bi-directional logic-seeking printing for throughput from 75-425 lines per minute; standard features include serial and parallel interfaces, top of form, perforation skipover, horizontal and vertical tabs, non-volatile format retention, expanded print and self-test. Options include graphics and APL.
Price: $1595 (OEM discounts up to 40%)
Available: Datasouth Computer Corp.
4740-A Dwight Evans Rd.
Charlotte, North Carolina 28210
and our distribuors

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **Biostatistics**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft BASIC
Hardware: Two Disk II; Optional: Printer and Watanabe Miplot
Description: This is a collection of programs aimed at the researcher who requires graphical representation and analysis of data. The package performs the following tests: Linear Regression, Exponential Regression, Curvilinear Regression, Data Plotting, Student t Tests (paired and unpaired with calculated probability), Mann-Whitney U Test and Wilcoxon Paired Test. A significant optional feature enables the user to generate graphical output on the Watanabe Miplot plotter. The package includes both program and data disks (DOS 3.2) as well as documentation.
Price: $40
Available: A2Devices
P.O. Box 2226
Alameda, California
94501
(415) 527-7380

Name: **Hebrew II™**
System: Apple II
Memory: 48K
Language: Applesoft in ROM or Language System
Hardware: Apple II with one disk drive
Description: The first foreign language word processor for the Apple II in America. This program puts Hebrew characters on the screen from right to left (and numbers left to right in their natural order) and allows full cursor movement and character editing. Text can be printed, saved to disk, and recalled for further editing, which makes it ideal for independent student work. It is particularly useful for labeling any Apple Hi-Res page such as charts, maps, and pictures. Hebrew II can produce graph labels, press-on labels, memos, posters, and, of course, practice in learning Hebrew.
Price: $60
Available: Aurora Systems, Inc.
2040 E. Washington Ave.
Madison, Wisconsin
53704

Name: **DOW2000**
System: Apple II
Memory: 32K
Language: Applesoft
Hardware: Disk 3.3/3.2 Printer Option
Description: Stock Market Analysis will determine price projections based on a stock's BETA coefficient or Relative Strength number and the Dow Jones Average. Projections are made as you vary the DOW. (What if....) On 1 stock or entire portfolio with single scan, quick scan, or variable scan of values. Included is the booklet "The Art of Timing Your Stock's Next Move." Author in market 17 years and former registered Investment Advisor with S.E.C.
Copies: Just released
Price: $29.00 with booklet (booklet alone $6.00).
Author: CIAC: Calabrese
Available: BIT'N PIECES SERIES
P.O. Box 7035
Erie, Pennsylvania 16510

Name: **C1P Animation and Shape Table Graphics**
System: OSI C1P cassette or PICO DOS
Memory: 8K cassette, 20K disk
Language: BASIC and assembler
Description: The animation package contains a BASIC program for drawing from the keyboard, without any numbers or programming, any number of single page pictures which are catalogued and POKEd into an indexed shape table. They may be saved to tape for later use. The following three assembler routines are organized by a short BASIC executive to give the user the ability to do complex high speed graphics and animations through simple BASIC programming. CLEAR: Clear or fill any portion of the screen in one page increments. PUTPIC: Call any catalogued picture to any part of the screen. FLASH: Flash any portion of the screen, or alternate between two pictures.
Price: $22.95 cassette, $24.95 disk fully documented
Author: Ken Madell
Available: Earthship
17 Church St. #28
Nutley, New Jersey
07110

Name: **Disk Bowling System**
System: PET/CBM
Memory: 32K (16K for smaller version)
Hardware: PET with disk and printer
Description: A complete scoring system for bowling league secretaries. Scratch and handicap bowling leagues with up to 24 teams (smaller version handles 12 teams). Features include disk records, accuracy, and extensive editing giving the secretary complete control of the data. Provisions are included for forfeits, blinds, partial absences, snapout errors, postponements, team ties, individual ties, subs, name changes, drops, ineligibles, messages, display of secretary's lane, and lane assignments anywhere in a 98-lane house. It is designed to be complete and yet save paper costs. The Epson option produces compacted printing saving another 25%. A year-end sweeper program that runs off of the final data disk is available, as is a complete archive program that will read each week's disk record for data on each individual.
Price: Starts at $40.00
Available: Harry H. Briley
P.O. Box 2913
Livermore, California
94550

Name: **5 Great Games!**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II Plus, Disk II
Description: Includes Animal Bingo, Jungle Safari, Space Defense, Sky Watch, and the unforgettable Air Traffic Controller. These are our most popular games — every one is Hi-Res, chock full of shape tables, and full of great machine language sound effects — some like you've never heard before. There's enough action and intrigue to keep you going for months!
Copies: Many
Price: $29.95 (or $9.95 for any one of the above games). Includes game cards, two disks, instructions.
Available: Avant-Garde Creations
P.O. Box 30161
Dept. MCC
Eugene, Oregon 97403

Name: **Mini-Count**
System: PET/CBM
Memory: 8K
Language: BASIC and machine code
Hardware: Connector and clip leads
Description: Uses the PET/CBM parallel user port to measure frequency and time intervals. Can also count pulses. Many sophisticated features such as auto-ranging, averaging, and external stop/start signals. Frequency limit of 17 Khz and pulse widths of 45 usec to 65.53 msec.
Price: $19.95 includes cassette and manual
Author: Ralph D. Goff
Available: Optimized Data Systems
P.O. Box 595
Placentia, California
92670

Name: **The Ultimate Catalog**
System: Apple II/Apple II Plus
Memory: Minimum 20K (ROM Applesoft)
Language: Applesoft and machine RWTS
Hardware: Apple II, Disk II, DOS 3.2
Description: Now you can format your directory to appear any way you wish. Block similar programs together; write headers mid-directory; separate by sections. This 5K, menu-driven utility is easy to use and performs the following functions: Alphabetize any portion or all of directory, move any file, exchange any two files, highlight or remove highlighting from any file name, insert blank line(s), delete any file, lock or unlock all files, delete or restore all files.
Price: $6.50 for listing and instructions
Author: Larry Abrams
Available: Aries Software
P.O. Box 58
Los Altos, California
94022

Name: **Apple Alarm**
System: Apple II with Firmware Card or Apple II Plus
Memory: 48K RAM
Language: Applesoft DOS 3.2, 3.3
Hardware: Disk Drive, Paddles, Sensors (switches)
Description: Apple Alarm is a program that converts your computer into a sentry, keeping track of fire, smoke, intrusion, motion, moisture and other on/off sensory inputs. Attach your floor mat, door-window switch, fire alarm or other sensor to the paddle buttons and your Apple will sound an alarm or quietly keep time from the moment triggered. Have your Apple guard your home, tell you when the kids came home...or left. Know when your night janitor arrived.
Copies: Just released
Price: $20.00 includes 12-page manual
Author: Andent Inc.
Available: Andent Inc.
1000 North Ave.
Waukegan, Illinois 60085

Name: **COMCON Disk**
System: OSI Challenger (C2 and C3 series)
Memory: 32K or 48K
Language: BASIC/6502 Assembly under OS65D
Hardware: Disk drive, modem, CRT, optional printer; (video and serial versions available).
Description: A telecommunications interface program providing smart terminal facilities via modem. Useful for transferring software or data files and saving them on disks. Allows communication with mainframes or other micros, uploading and downloading and printing. Control key initiation of LOGON messages. User-controlled tailoring of protocol and system characteristics, including port and output device, half or full duplex, parity, checksums, baud rate, and line control.
Price: $45.00 on 8" disk postpaid. Includes documentation (specify 32K or 48K version, and whether serial or video).
Author: Sid Brounstein
Available: Responsive Computer Technology, Inc.
P.O. Box 719
Silver Spring, Maryland
20901

Name: **Laser Wars**
System: OSI C1P or Superboard
Memory: 8K
Description: Maneuver your space craft to line enemy fighters in your cross-hairs and destroy them with your lasers. A fast action arcade-type game with machine language graphics for one player.
Price: $7.95 ppd.
Author: Brian and Craig Zupke
Available: BC Software
9425 Victoria Drive
Upper Marlboro,
Maryland 20870

Name: **Perception 3.0**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Apple II, Disk Drive, Game Paddles
Description: Seven High-Resolution activities will challenge the user's visual perception and hand-eye coordination. Activities are Length Perception; Shape Memory; Size Comparison; Star Trace; Centering a Falling Line; Visual Pursuit; and Tilt Maze. Each of the activities offers a wide range of parameter settings for both the skilled and unskilled user.
Price: $24.95 includes documentation and diskette.
Available: All computer dealers, or Edu-Ware Services, Inc.
22222 Sherman Way,
Suite 102
Canoga Park, California
91303

Name: **A-2a. Moving Averages**
System: PET
Memory: 8K
Language: BASIC
Hardware: PET/CBM
Description: Computes centered moving averages for 3 span lengths and prints values and/or differences. Discloses cyclic movements in a time series such as stock prices. Includes logical file input and modification to update and delete old data.
Price: $15.00 for cassette and documentation
Author: Claud E. Cleeton
Available: Claud E. Cleeton
122-109th Ave., S.E.
Bellevue, Washington
98004

Name: **AIM Video-Trek**
System: AIM 65
Memory: 12K
Language: BASIC
Hardware: Video terminal
Description: A new Trek game designed to run on *any* AIM 65 with 12K memory and a video terminal. You command the Enterprise in its search to destroy the invading Klingons. You have superior weaponry, but they have a cloaking device. Sound effects are provided by using CB2 output of the User 6522 VIA (CB2 sound instructions included).
Copies: Just released
Price: $12.00 on cassette, ppd.
Author: J.S. Wahlquist
Available: J.S. Wahlquist
1643 N. Formosa Ave., #4
Los Angeles, California
90046

MICRO

Microbes and Updates

Mike Rowe
Microbes & Updates
P.O. Box 6502
Chelmsford, MA 01824

J.G. Wendel, of Ann Arbor, Michigan, sent this microbe:

For some time I've been using Mr. B.E. Baxter's fine routine in the January 1980 MICRO (20:30) for direct writing to the Apple screen. Just now I've discovered a small bug in it, because I happened to fill up line #16, apparently for the first time. What happened was that the last character of the line was lost, because the file should be saved with length $3D0 rather than $3CF. The correction consists in changing the code at $0396/7 in your program to C4 B0.

Edward H. Carlson, Okemo, Michigan, sent us this update to his article:

I have received some phone calls about my article, "A 6502 Assembler in BASIC," in MICRO (34:7). If you are having trouble making the program run, rest assured that it does work on OSI C2 and C4 machines, as is. Dale Mayers pounded it into his C4P and found no real errors. However, he did point out that the 56 in line 124 should really be a 14. He also pointed out that a cleaner logic is possible in this region and the program will then run slightly faster and use less memory. The changes are:

```
124 FOR I = 1 TO 4:FOR J = 1
TO 56 STEP 4

130 IF L$ = MID$(C$(I),J,3)
THEN N = 14*(I - 1) + (J + 3)/4:
GO TO 161

155 delete

163 OP = VAL(MID$(F$(I),J,3))
```

If you are having trouble, you have made a key-in error. Check out the program using PRINTs, and check every possible op code and addressing combination. A lot of work? You bet, but worth it! Finally, if you have a C1, you will need to change the screen display to fit it into 24 characters, probably using PRINTs rather than POKEs. It would be much appreciated by readers of MICRO if anyone who makes the conversion of this program to a C1 or other machine will write a letter describing the modifications.

John G. Ruff of Plymouth, Minnesota sent us the following update:

I read with great interest the March 1981 article, "A 6502 Assembler in BASIC," by E. H. Carlson (34:7). After only a short time I began the translation into my 24K OSI C1P with 64 × 32 video. During the process I discovered items worth commenting on.

1. Although spaces on lines are convenient for casual reading (especially when used to an editor/compiler), a user with 4K RAM cannot afford the luxury; there are 104 spaces (bytes) in lines 2000 - 2027! By removing all spaces and REMark statements there will be about two pages available above BASIC. Line 2030 should be changed to point to the beginning *Non-BASIC* location to prevent overwriting the BASIC vectors in page 2. After removing all spaces (lines 2000 - 2027), change the following lines:

```
124 FORI = 1TO4:FORJ = 1TO14:
N = 3*J - 2

163 OP = VAL(MID$(F$(II),
JJ*3 - 2,3))
```

Be sure to run the program (without doing any assembly) before attempting to determine the highest location used by BASIC, since variable and string space is allocated at RUN time.

2. The following addressing modes are not documented by the author, although they are included in the program:

a. Indirect:	JMP (*****)
b. Indexed Indirect:	ADC (**;X)
c. Indirect Indexed:	ADC (**);Y

Note: ** equals Hex digit.

3. To allow the conversion of hexadecimal numbers with 1,2, 3 or 4 digits change lines 4000 - 4050 to the following:

```
4000 N = 0:LL = 16:FORI = 1TOL

4010 M = ASC(MID$(C$,I,1))
- 48:M = M + 7*(M    9)

4020 N = N + M*(LL    (L - I)):
NEXT:C$ = STR$(N):N = Q + 23
- (LEN(C$))
```

The above will also right-justify the decimal output to allow alignment with the ASCII output.

I have used the above assembler to build several small device handlers and find the program most successful. Should there be any questions feel free to contact me at Weldon Electronics, Inc., 14010 23rd Ave. No., Plymouth, MN 55441 (612/559-1984).

Lee Meador of Arlington, Texas wrote to us with this tip:

The article entitled "Create a Data Disk for DOS 3.2 and 3.2.1" in the June 81 issue is indeed interesting for someone who needs to save space for data on Apple II disks. There is one related item that needs to be made known about the use of track 0. The Apple DOS (3.2 or 3.3) does not allow the use of track zero. Consider how the track/sector list is used by the DOS. (See pages 128-129 of the DOS manual.) In the list two bytes hold the track (1 byte) and the sector (1 byte) of the appropriate sector of the file. The first item in the list for the first 256 bytes of the file, the second item for the next 256 bytes, etc. If the first of the two bytes is zero, then it is assumed by DOS that that block of 256 bytes is not used in the file. A sector is not allocated for that group of 256 bytes. Perhaps this is a design error in the DOS, or perhaps they thought no one would ever try to use track 0 so they could cut out a few bytes of code to speed things up a little. (Obviously, only track 0, sector 0 should be off limits.) Anyway, when that first byte is zero, the DOS, rather than looking on track zero for the sector, will assume that the sector doesn't exist.

This isn't a problem if all your files are created and read by DOS. DOS will never allocate a sector on track zero, whether you free up the space or not. *But...* some file copy programs, in particular, FID, MUFFIN and its derivatives, DEMUFFIN, and Niffum, and other similar programs, will put parts of files into track 0. The problem is only noticed afterwards when you try to use DOS to access the file. It isn't there.

I suggest this change to Mr. Sogge's article to solve the problem. Change the line three up from the bottom of the middle column of page 49 from "(11,0,38) to FF E0 00 00" to read "(11,0,38) to 00 00 00 00". This will leave track 0 marked as in use and the file copy programs won't be tempted to allocate space there.

MICRO

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

6502 Bibliography: Part XXXVIII

985. Abacus II 2, Issue 11/12 (November/December, 1980)

Anon., "IAC Apnote: Serial Handshake Modification with Tabs," pg. 4-5.
Using the Apple High Speed Serial Interface Card with printers and using the existing data input line to sense if the printer is busy.

Anon., "IAC Apnote: Upper/Lower Case and Special Characters," pg. 9-15.
A method for using the language card on the Apple so that control of upper and lower case is controlled by the shift key.

Sokal, Dan, "IAC Apnotes: Pascal PEEKs and POKEs," pg. 13-15.
A program for your Pascal library.

Anon., "IAC Apnote: Text Screen Mapping and Use," pg. 16-17.
All about text pages, screen maps, and character display values, including an example of use.

Davis, James P., "Savings," pg. 23-24.
A program to calculate interest on savings with your Apple.

Davis, James P., "Printer On — Says-a-Me," pg. 24.
A printer control program for the Apple/Trencom 200/AII-g combination.

Davis, James P., "Print Catalogs," pg. 25.
An easy to use catalog printing routine for the Apple.

Robbins, Greg, "DOS Tricks for DOS 3.2.1," pg. 26.
Several techniques for users of Apple DOS 3.2.1.

Anon., "How to Obtain Those Special Characters," pg. 27.
A machine language routine that allows several extra characters to be printed on the Apple II.

Davis, James P., "Two M/L Sound Effects Programs Revisited," pg. 29-31.
Tutorial with two example routines for the Apple.

986. Peek(65) 1, No. 12 (December, 1980)

Stevenson, Greg, "U2," pg. 2-5.
Tips for OSI users including an addition to BEXEC to add flags.

McMurray, C. Eugene, "Something for Nothing," pg. 5, 16.
How to avoid confusion between variables and BASIC function labels on OSI micros.

Jones, Davis A., "Cassette Corner," pg. 6.
Some hardware and software assists for cassette operation.

Hooper, Phil, "CALL for OSI BASIC," pg. 7-8.
How to provide a CALL routine for the OSI machines to invoke a machine language program.

Williams, Jim, "How to Edit Programs and Keep Variables," pg. 8.
Tips on the use of OSI BASIC variables.

Anon., "Location of Routines," pg. 10-11.
A listing of location of routines in Microsoft BASIC Ver. 1.0, Rev. 3.2 in OSI C1P and Superboard II.

Lundberg, Charles " 'PRINT AT' Hides in BASIC," pg. 11.
A formatting technique for OSI users.

Goodman, Kelsey, "OSI Files," pg. 14-15.
Discussion on handling OSI files.

Dennis, Neil, "Graphics Program," pg. 16.
A graphics program to draw patterns on the OSI screen.

987. Stems from Apple 3, No. 12 (December, 1980

Stein, Dick, "Review of Pascal Version 1.1," pg. 4, 9, 13
Version 1.1 of Apple Pascal has had many changes, reviewed in this article.

Anon., "Renumber Problem — DOS 3.2 and 3.3," pg. 8.
How to fix a bug in the Applesoft Renumber program.

Robinson, Alan H., "A Look at Fortran," pg. 10-12.
Comments on a user's experience with Apple Fortran. Some pitfalls to be avoided are discussed.

Dulk, G.A., "Use of Apple as a Word Processor," pg. 15-19.
The Apple Pascal system has many of the desirable features of a Word Processor.

Warren, John W., "Ballistic," pg. 20-22.
This program will calculate and print a complete ballistics table, bullet flight path, etc.

988. The Apple Peel 2, No. 12 (December, 1980)

Brown, Tom, "POKE Salad," pg. 4-5.
Discussion of a malfunction of the VAL function which is memory dependent, for the Apple.

Graham, Johnny, "13/16 Sector Switch Modification," pg. 6.
Add a switch to your Apple disk controller card to switch from 13 to 16 sectors (DOS 3.2/3.3).

Donahue, Tom, "13/16 Sector Switch," pg. 7.
Another approach to switch between 13 and 16 sectors on the Apple disk system.

989. MICRO No. 31 (December, 1980)

Carlson, Ron, "Graphing Rational Functions," pg. 7-9.
A discussion and listing of a general-purpose graphing program for the Apple hi-resolution screen.

Elm, Robert L., "A C1P User's Notebook," pg. 11-13.
Secrets of the Challenger and notes on ACIA, graphics, tape control, etc. for OSI users.

Davis, Harvey S., "Drawing a Line on PET's 80 × 50 Grid," pg. 15-19.
A collection of flexible machine language routines for graphing.

Weiner, Eugene V., "A Random-Character Morse Code Teacher for the AIM 65," pg. 21-23.
Program your AIM to generate code sounds at 13 words per minute and up.

Tibbetts, Gregory L., "An Apple Flavored Lifesaver," pg. 25-30.
An Apple game.

Wright, Loren, "MICRO PET Vet," pg. 33.
Several new Commodore products are discussed.

McBurney, N.R., "Creating an Applesoft BASIC Subroutine Library," pg. 37-40.
Using EXEC instead of RUN offers increased flexibility and can link Applesoft programs from a common disk-resident library.

Staff, "Microscope," pg. 43.
PBASIC-DS Version Two is reviewed.

Crites, Roger C., "Stuffit," pg. 45-47.
A time-saving utility program for PET BASIC files.

Rowe, Mike (Staff), "New Publications," pg. 51.
Two new books are reviewed.

Froelich, Jerry W., M.D., "Microprocessors in Medicine: The 6502," pg. 53.
Discussion of 6502 computer programs in medical education.

Lindsay, Len, "Atari Bits," pg. 57-59.
Discussion of Atari keyboard buffer, screen protect feature, dynamic keyboard, hi-resolution graphics, etc.

Taylor, William L., "Relocating OSI ROM BASIC Programs," pg. 61-63.
This BASIC program will assist OSI users to understand how their Microsoft BASIC and monitor are used.

Vrtis, Nicholas J., "Cassette I/O for SYM BASIC," pg. 65-69.
Expand the capabilities of SYM BASIC with this cassette I/O handler, allowing access of the cassette as a data file.

Boering, Brooke W., "Multiplying on the 6502," pg. 71-74.
Here are five routines to speed up multiplication on any 6502 system.

Dial, Wm. R., "6502 Bibliography: Part XXVII," pg. 89-93.
Over 150 new references to the voluminous and growing 6502 literature.

990. The Apple-Dillo (December, 1980)

Teas, George, "Pascal Primer," pg. 5.
Pascal routines providing ready access to inverse and flash functions on the Apple.

Beal, Bob, "Using Parameters with the Control-Y Monitor Command," pg. 6-8.
A discussion of the Apple Control-Y with two listings as demos.

991. The Seed 2, No. 12 (December, 1980)

Nelson, Rod, "Apple Speed," pg. 7.
An interesting experiment comparing the speeds of routines.

Anon., "IAC APNOTE: The Apple II Cassette Interface," pg. 20-23.
A good discussion of the operation of the cassette interface on the Apple II.

992. O.S.I. Users Independent Newsletter No. 7 (December, 1980)

Curley, Charles, "OSI Resources," pg. 1-13.
Entire issue devoted to a catalog of OSI resources including hardware, utilities, education and business programs, word processors, etc.

993. Southeastern Software Newsletter No. 23 (December, 1980)

McClelland, George, "Update on Word Processing," pg. 4-5.
Discussion of Super Text II and 80-character Apple Pie and format.

Christman, John, "Printer Controls for the Paper Tiger," pg. 8-9.
This Applesoft program prints out the number of columns for each size of print on the Paper Tiger.

Sander-Cederlof, Bob, "Character Codes," pg. 9.
A short routine to print out a chart of the complete Apple screen codes.

Powell, David, "Apple Escher," pg. 9.
A graphics program for the Apple Hi-Res screen.

994. From the Core (December, 1980)

Budge, Joe, "Natterings from the Nabob," pg. 3.
A fix for the renumber program in DOS 3.3, disk centering problems, a mod for Apple disk analog cards to reduce errors in going from disk to disk and 3.2 to 3.3, how to identify disk drives made by Shugart and an alternate supplier.

Anon., "Copyone," pg. 4.
An improved Pascal single disk copy.

Anon., "Oligopoly Simulation," pg. 8-9.
A program to model a simplified economic system and determine the most profitable pricing strategy given a number of alternatives.

995. The Cider Press (December, 1980)

Weiglin, Peter C., "Formatting — Part Three."
More on formatting techniques for the Apple.

Roxburgh, Alastair, "Tape to Tape Copy," pg. 6.
How to copy difficult programs on the Apple cassette interface.

Fields, Randy, "Extensions to Print Using," pg. 7.
Additional information on the Print Using function.

Thompson, C.J., "Niffum," pg. 8-9.
A reverse MUFFIN (DOS 3.3 to 3.2 converter) for the Apple.

Anon., "Try These Patches on DOS 3.2: Part II," pg. 9.
A second installment of some handy DOS modifications for the Apple.

Schaffer, Jay, "FRE(x) Modifications," pg. 10-12.
A sixteen-sector version of a program to find the free space remaining on a diskette.

Nareff, Max J., "Beginner's Notes on Pascal," pg. 15-16.
A program in Pascal demonstrating the use of the "String" intrinsic functions (COPY, POS, CONCAT). Also a split-screen demo.

996. Washington Apple Pi 2, No. 12 (December, 1980)

Anon., "The Case of the Disappearing Directory," pg. 7-11.
Discussion of the Pascal handling of directories.

Wurzel, Bill, "Linear Scaling in Hi-Res Graphics," pg. 17-18.
A tutorial in Apple Hi-Res graphics.

Wurzel, Bill, "A Bug in the Toolkit Assembler," pg. 18.
A bug in the 6502 assembler on the Toolkit disk and a fix for the bug.

Schwartz, Dana J., "Dana's Hex Loader," pg. 18-20.
A utility for the Apple.

Crosby, Mark L., "Questions, Questions, Questions," pg. 21.

Some common questions on Apple programming or operation.

Mitchell, Howie, "Printing Out the Hi-Res Screen," pg. 22-24.
A program for the Apple and the Anadex DP-9501 printer.

Cottrell, C., "Equations for Some Common Bessel Graphs," pg. 24-27.
Equations and listing to print Bessel function graphs.

997. AppleGram 2, Issue 12 (December, 1980)

Sander-Cederlof, Bob, "Word-Search Puzzle Maker," pg. 3-7.
Routines to develop matrices of letters and to find hidden words therein, for the Apple.

Firth, Mark, "Short Cut to Common Routines," pg. 8.
How to get a common routine into several programs using the Renumber program and the EXEC function on the Apple.

Firth, Mike, "MID$ vs. LEFT$ and RIGHT$ and Other Routines," pg. 13-14.
A series of handy routines and techniques for the Apple.

998. SoftSide 3, No. 3 (December, 1980)

Pence, Fred, "Christmas Card," pg. 20-21, 50-51.
An Apple program using Lo-Res graphics.

Pelczarski, Mark, "The Developing Data Base," pg. 30-33.
Part 4 of a continuing series for the Apple and Atari.

Barts, Duane, "Connect-A-Dot," pg. 34-37.
A sketching program for the Apple Hi-Res graphics.

Ward, Dennis and Osborne, Leon A., "One-Liners," pg. 51.
Several programs for the Apple.

Bohlke, Dave, "Baseball," pg. 65-68.
A game for the Apple.

McKenna, Michael, "Space Dodge," pg. 70-71.
A game for the Atari.

Bohlke, David, "States and Capitals," pg. 80-81.
An educational game for the Atari.

Bohlke, Dave, "Speedello," pg. 88-89.
An Othello-like game for the Atari.

999. G.R.A.P.E. 1, No. 11 (December, 1980)

Wasson, Philip, "Fast Hi-Res Scroll," pg. 4.
An Apple program for a machine language fast scroll.

1000. Softalk 1 (December, 1980)

Wagner, Roger, "Assembly Lines," pg. 14-16, 22.
Part 3 of a continuing tutorial on Assembly language, for the Apple.

1001. Apple Assembly Line 1, Issue 3 (December, 1980)

Laumer, Mike, "Integer BASIC Pretty Lister," pg. 3-8.
An Apple program to make pretty listings of Integer BASIC programs.

Sander-Cederlof, Bob, "S-C Assembler II Notes," pg. 9-14.
Discussion and patch for .da directive; block move and copy for Version 4.0; etc.

Sander-Cederlof, Bob, "Handling 16-Bit Comparisons," pg. 16.
How to compare two double-byte numbers on the Apple for branching routines.

1002. T.A.R.T. 1, No. 1 (March, 1980)

Koerin, Sidney, "Ditty," pg. 2.
A fix to DOS 3.2.1 of the Apple to make the INIT program go faster.

Shanes, John, "Faster Than a Speeding Bullet!!", pg. 8.
Speed up your Apple cursor with this hardware mod.

1003. T.A.R.T. 1, No. 2 (May, 1980)

Rivers, Jerry, "Lower Case from Your Apple," pg. 2.
Two routines to allow you to use both upper and lower case in your Apple programs.

1004. T.A.R.T. 1, No. 3 (October, 1980)

Anon., "Disk Labeling," pg. 3-4.
A BASIC program to label your Apple diskettes.

1005. T.A.R.T. 1, No. 4 (December, 1980)

Hubbard, Bill, "A Striking Article," pg. 2-3.
Add a typewriter-like sound to your Apple keys.

1006. Apple Bits 2, No. 10 (December, 1980)

Anon., "Apple Disk II Card DOS 3.2/3.3 Switch Modification," pg. 4.
A convenient hardware mod for the Apple disk controller card.

Koehler, John, "BASIC Basics," pg. 5.
A common denominator program for the Apple.

Kovalik, Dan, "Taking the Mystery and Magic Out of Machine Language," pg. 8-10.
An Apple Hi-Res graphs left/right flip program.

1007. The Apple-Dillo (January, 1981)

Clardy, Robert C., "Converting Integer BASIC Programs to Applesoft," pg. 5-6.
A useful utility for the Apple programmer.

1008. OSIO Newsletter 3, No. 1 (January, 1981.)

Sand, Paul A. and Morganstein, David, "Prettylisting," pg. 1, 2.
Improve the appearance of your 6502 program listing with this routine. For OSI computers.

Kirshner, Joe, "OS-65 Notes," pg. 3-5.
Some discussion of the handling of files on the OSI system.

Compton, Radford, "Assignment: Format," pg. 6-7.
Format a report with this OSI program.

1009. The Harvest 2, No. 5 (January, 1981)

Stadfeld, Paul, "Toccata and Fugue in CTRL-D," pg. 1-3.
A tutorial on Apple keyboard logic, modifications to the keyboard, etc.

1010. The Apple Peel 3, No. 1 (January, 1981)

Jenkins, Jerry, "Space Saver," pg. 6.
Get more storage area on that diskette for your Apple Hi-Res pictures.

Jenkins, Jerry, "APTYPE/MX-80," pg. 6.
Improve the compatibility of the APTYPE/MX-80 combination on the Apple.

1011. The Seed 3, No. 1 (January, 1981)

White, Harry, "Move On, String Writer," pg. 3-4.
A tutorial for Hi-Res graphics on the Apple, with a Hi-Res page move demo.

Anon., "Apple Pi Conventions," pg. 6.
A utility to set up program REM statements, etc.

1012. Nibble No. 8 (January, 1981)

Capella, Mark, "Will 'O The Wisp," pg. 9-21.
A fantasy game for the Apple.

Riley, Kevin D., "Cassette Tape Visual Display Monitor," pg. 22-23.
A mod to make tape loading more reliable.

Laird, Alexander, "Fun with Apple's Assembler," pg. 27.
Some insight into the Apple Monitor's graphics.

Darr, Robert W., "Apple and the 3.3 DOS," pg. 31.
A review of the new DOS and it's feature utilities.

Berman, Andrew, "Blast Away!", pg. 35-39.
A shooting gallery program for the Apple.

Harrell, Keith, "Pascal Pointers and Principles," pg. 41-45.
The filer of the Pascal system and the compound statements.

Reynolds, William III, "String Function for Integer BASIC Programs, pg. 53.
A subroutine allowing for a string variable to be set equal to the printed string of a numeric variable on the Apple.

Szetela, David P., "BASIC/Machine Language Subroutine Creator," pg. 53.
A BASIC POKE creator for the Apple BASIC.

Reynolds, William III, "Deleting Files Absolutely," pg. 53-57.
Defeat the recovery of a deleted file on the Apple diskette.

Thompson, C.J., "Niffum," pg. 61.
A reverse muffin for the Apple DOS 3.3/3.2 systems.

Abrams, Larry, "Loan Reduction Analysis/Display," pg. 63.
A financial program for the Apple.

1013. KB Microcomputing No. 49 (January, 1981)

Baker, Robert W., "Potpourri: New PET Monitor," pg. 10-13.
A well-documented monitor ROM called Mojana/1, BASIC 4.0/DOS 2.1, etc.

Baker, Robert, "Real-Time Spectrum Analyzer," pg. 48-50.
A PET program for audio signal analysis.

Chamberlin, Hal, "Simulation of Musical Instruments," pg. 53-58.
Computer music synthesis for 6502 machines.

Rager, Edward, "Scramble," pg. 78-80.
A PET program demonstrating the utility of nested subroutines.

Deininger, Rolf A. and Tujaka, Don, "Apple Connections," pg. 122-123.
Put connectors on the back panel of your Apple for convenience in connecting peripherals.

Hirbernik, Robert M., "Space Race," pg. 126-128.
A graphics game for the Apple.

Baker, Donn Burke, "Reverse Video for the OSI C1P," pg. 176-182.
A $10 hardware mod for the C1P.

Hutchinson, Thomas E., "Second Cassette Interface with One IC," pg. 188-190.
Improve the flexibility of your PET with this mod.

1014. Byte 6, No. 1 (January, 1981)

Crawford, Chris and Winner, Lane, "An Introduction to Atari Graphics," pg. 18-32.
A tutorial on Atari graphics with two listings.

Roybal, Phil, "The Picture-Perfect Apple," pg. 226-235.
An Apple program in Assembly language for the Qume Sprint Micro 3 printer.

1015. Softalk 1, No. 5 (January, 1981)

Wagner, Roger, "Assembly Lines, Part 4," pg. 22-27.
Incrementing, decrementing and loops in assembly language for the Apple.

1016. Atari Computer Enthusiasts 2, Issue 1 (January, 1981)

De Groot, Bill, "Business Program," pg. 2.
An Atari program to calculate interest and payments on loans.

1017. Interface Age 6, No. 1 (January, 1981)

Baker, Al, "Game Corner," pg. 22-26.
A game for the Atari called "Cannon Duel."

Zant, R.F., "File Cabinet and Ampersosrt II," pg. 94-96.
Improve the sort routine in the Apple File Cabinet.

1018. The G.R.A.P.E. Vine (January, 1981)

Ude, Art, "Neon Sign," pg. 3.
A program of the crawler or banner type for the Apple.

Ude, Art, "Throttle," pg. 4.
Applesoft and Integer BASIC listings for slow list on the Apple.

Lawson, Steve, "Screen Position," pg. 5.
An Apple program to find the screen position given row and column parameters.

Lawson, Steve, "Binary to Decimal to Binary Conversion," pg. 6-7.
An assist to converting numbers on the Apple.

1019. From The Core (January, 1981)

Budge, Joe, "King Kluge," pg. 3.
A hardware mod for the Apple to restore singlestep and other Old ROM features on your Autostart machine.

Whittaker, Alec, "Timer Subroutine," pg. 5.
An inexpensive clock for the Apple.

Holzworth, Paul, "The Secrets in Your Apple, ...Maybe," pg. 7.
An examination of the latest Apple motherboard seems to predict things to come.

Budge, Joe, "UPPER/lower Case Pascal," pg. 8.
Modify your Apple BIOS to allow U/L in Pascal.

Anon., "DOS to Pascal Transfer Program," pg. 8-9.
A program which will transfer Apple files from DOS to Pascal.

Anon., "L/C System Startup for Pascal 1.1," pg .13.
A program which calls an assembly language routine to set up various startup options of the Apple.

1020. The Michigan Apple-Gram (August, 1980)

Rivers, Jerry, "Technical Tidbits," pg. 6.
Fix for the fix for the DOS Append on 3.2 and 3.2.1; garbage collection to free up space, etc. for the Apple.

Anon., "IAC Apnote: Applesoft Array Eraser," pg. 16.
A program for the Apple.

Anon., "IAC Apnote: Converting Integer BASIC Programs to Applesoft," pg. 19.
A discussion of a useful procedure for the Apple.

Anon., "IAC Apnote: Out of Memory Errors," pg. 20.
Reasons for getting "Out of Memory" errors on the Apple.

Anon., "IAC Apnote: VTAB and HOME Converter," pg. 21.
Some useful routines for the M&R SUP-R-Terminal on the Apple.

Anon., "IAC Apnote: Modifying the LISA Assembler," pg. 22.
Modification of the Apple utility to handle user functions.

Anon., "IAC Apnote: DEL Character Killer," pg. 26.
A routine for the Apple system.

Anon., "IAC Apnotes: Misc. Apnotes for Apple Pascal Systems," pg. 28-37.
A series of Pascal Notes on GETREM, TAKE 280, TRANSFER, FOREIGN, LONG INTEGER FIX, LOAD/SAVE to DISK, etc.

1021. The Michigan Apple-Gram (September, 1980)

Tuttleman, Roger, "Disk Inform," pg. 7-10.
An Apple assembly language program for printing information about a diskette.

Rivers, Jerry, "Technical Tidbits," pg. 11-14.
A tutorial on the 6502 operation and the LISA Assembler.

Hall, Lennis L. and Ankofski, Tom, "Select By Number," pg. 17-19.
A Hello program for the Apple Disk system.

1022. The Michigan Apple-Gram (October, 1980)

Neuhauser, Robert, "I/O Port and Joysticks," pg. 5.
A hardware article for improving the game port I/O of the Apple.

Holderby, Michael, "Integer BASIC Token Scheme," pg. 6-7.
A tutorial for the Apple.

McClaren, Mac, "Catalog Free Sectors Revisited," pg. 7.
A listing that works in either Applesoft or Integer BASIC, together with notes on just *how* this machine language routine works.

Smith, Paul, "Catalog List," pg. 8-9.
A tutorial for the Apple.

Tuttleman, Roger, "Integer BASIC Append Methods," pg. 11-12.
Several routines for the Apple, including 'To Text Create,' a program to create an EXEC file to convert BASIC programs to Text files.

Tuttleman, Roger, "Fix for Applewirter," pg. 12.
A fix for using Applewriter with the Paymar Lower Case Adapter.

Tuttleman, Roger, "Integer Info Create," pg. 13.
A program to create an EXEC file to return HIMEM, LOMEM, start of program and end of variable address.

Anon., "My Disk Runneth Over," pg. 15-16.
Several routines including one that allows the Apple to use graphics programs written for the TRS-80, a fast text-copy program, etc.

1023. The Michigan Apple-Gram (November, 1980)

Holderby, Mike, "Programmer's Corner," pg. 5.
A tutorial on how to add beeps, buzzes, etc. to your Apple programs.

Walker, Carl, "DOS Patch for Single Stroke Entry," pg. 6.
A Single Stroke Entry to use with DOS.

Tuttleman, Roger, "Remove Lisa," pg. 8.
A utility for Apple users of the LISA assembler.

Rivers, Jerry, "Text File Reader," pg. 9.
A program to read any sequential text file into memory.

McLaren, Mac, "Disk Zap Conversion with DOS 3.3," pg. 9.
Mods for the popular Disk Zap utility to adapt it to the new Apple DOS 3.3.

Tuttleman, Roger, "Introductory Fortran Program," pg. 10-11.
A short program showing the forms of various FORTRAN statements and how to get the Apple clear of the bugs.

Smith, Paul and Rivers, Jerry, "Serial Interface Card Tabbing," pg. 11.
How to tab past column 40 using the Apple serial interface card.

Tuttleman, Roger, "FORTRAN Turtle Graphics Demo," pg. 14-15.
A simple Apple program demonstration of Fortran Turtle Graphics.

Tuttleman, Roger, "POKE Writer," pg. 15.
A program to convert assembly language routines to POKEs for BASIC programs.

Macdowell, Mac, "My Disk Runneth Over," pg. 16-17.
A software mod for Apple sound, and a telephone dialing routine that yields fast dialing capability.

Paul, L., "Un-Muffining Routine," pg. 17.
A procedure for converting a program from DOS 3.3 to DOS 3.2.

Rivers, Jerry, "The FORTRAN Format," pg. 18-19.
Notes from an Apple Fortran user with a Fortran listing of TEXTPRT, a routine to print any 'Text' file to your printer.

Tuttleman, Roger, "Free Sectors," pg. 20.
An Integer BASIC program to print the volume number and number of free sectors on an Apple disk.

1024. The Michigan Apple-Gram (December, 1980/ January, 1981)

Holderby, Mike, "Programmer's Corner," pg. 6-7.
A look at Apple's error trapping and input editing techniques.

Rivers, Jerry, "Fortran Format," pg. 7.
Notes by an Apple Fortran user shows the pitfalls in this language.

Lea, Diane, "Beginner's Corner," pg. 10-11.
Some tips for new Apple owners, including a graphics listing.

Tuttleman, Roger, "RWTS Disk I/O From BASIC," pg. 12-16.
A guide to using the RWTS disk utility, with several programs and routines for the Apple.

Rivers, Jerry, "Maybe You Didn't Know That...," pg. 17-18.
Some interesting notes on the Apple HIMEM:, LOMEM:, the speed of interpreters vs. compilers, Pascal and Fortran speed, etc.

Wiggington, Randy, "Read/Write Track-Sector," pg. 20-35.
Listing for this major Apple utility and a description of its internal workings.

ADVERTISERS' INDEX

JULY 1981

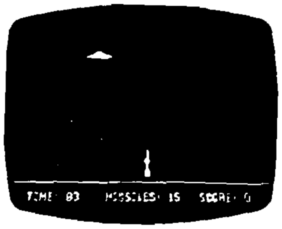
Apple II

sensational software

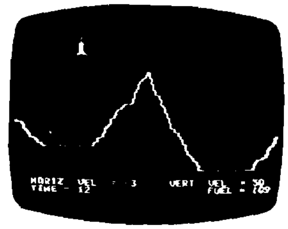
creative computing software

Space Games-I

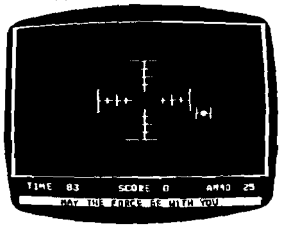
Cassette CS-4001 $11.95 4 programs Requires 16K Apple II or Apple II Plus



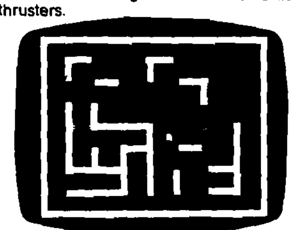
Saucer Invasion. Fire missles to destroy the invaders who fly at different speeds and altitudes.



Rocket Pilot. Maneuver your spaceship over the mountain using horizontal and vertical thrusters.



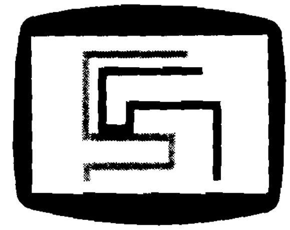
Star Wars. Shoot down as many TIE fighters as possible in 90 seconds.



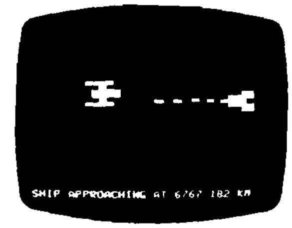
Dynamic Bouncer. A colorful, ever-changing graphics demonstration.

Strategy Games

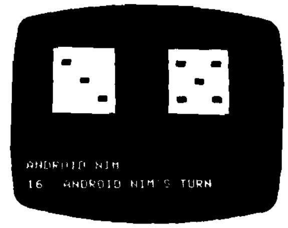
Cassette CS-4003 $11.95 4 Programs Requires 16K Apple II or Apple II Plus



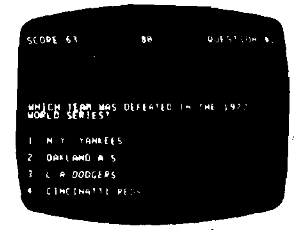
Blockade. Build a wall to trap your opponent, but don't hit anything.



UFO. Use lasers, warheads or guns to destroy an enemy spacecraft.



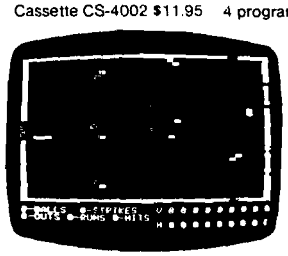
Skunk. A 2-player strategy game played with dice, skill and luck.



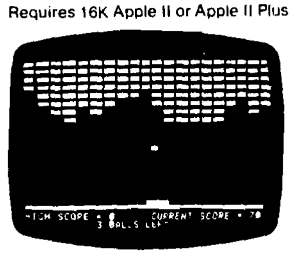
Genius. A fast-moving trivia quiz with scores of questions.

Sports Games-I

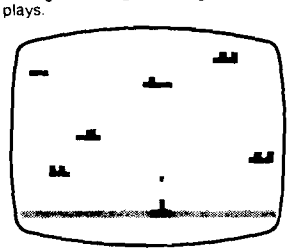
Cassette CS-4002 $11.95 4 programs Requires 16K Apple II or Apple II Plus



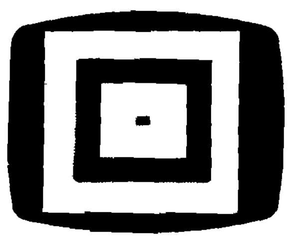
Baseball; A 2-player game with pitching, batting, fielding, stealing and double plays.



Breakout. Four skill levels and improved scoring make this the best breakout ever.



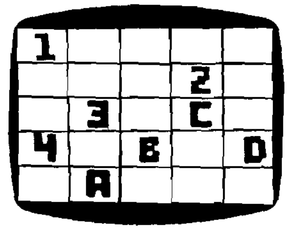
Torpedo Alley. Sink as many warships as possible in 2 minutes.



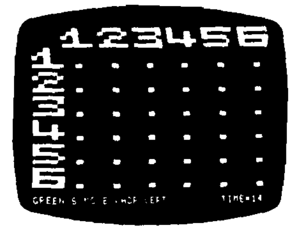
Darts. Use game paddles to control the throw of 6 darts.

Brain Games

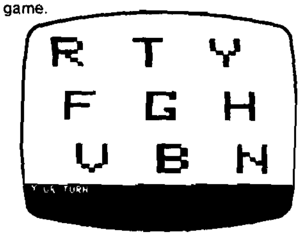
Cassette CS-4004 $11.95 7 programs Requires 16K Apple II or Apple II Plus



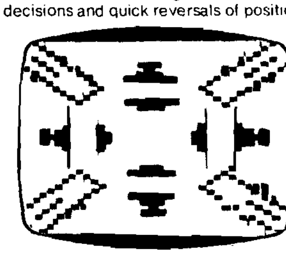
Dodgem. Be the first to move all your pieces across the board in this intriguing strategy game.



Nuclear Reaction. A game of skill, fast decisions and quick reversals of position



Parrot. A Simon-type game with letters and tones. **Dueling digits** is a version with numbers.



Midpoints and Lines. Two colorful graphic demonstrations. **Tones** lets you make mus and sound effects

Space & Sports Games

Disk CS-4501, $24.95
Requires 32K Apple II or Apple II Plus

This disk contains all eight games from cassettes CS-4001 and CS-4002.

Strategy & Brain Games

Disk CS-4502, $24.95
Requires 32K Apple II or Apple II Plus

This disk contains all 12 games and programs from cassettes CS-4003 and CS-4004.

Order Today

To order any of these software packages, send payment plus $2.00 postage and handling per order to Creative Computing, Morris Plains, NJ 07950. Attn: Clarice

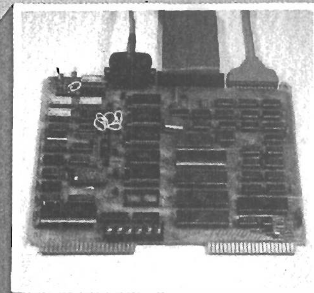
Visa, MasterCard and American Express orders may be called in toll-free.

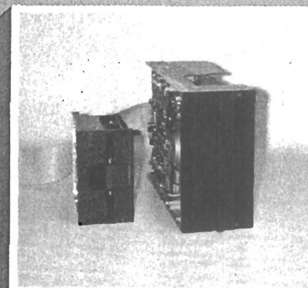
Order today at no risk. If you are completely satisfied, your money will promptly and courteously refunded.

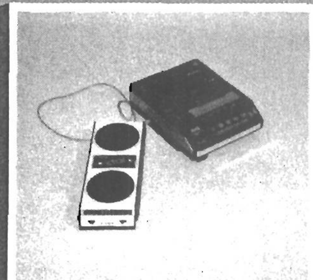
Creative Computing Software
Morris Plains, NJ 07950
Toll-free **800-631-8112**
In NJ, 201-540-0445

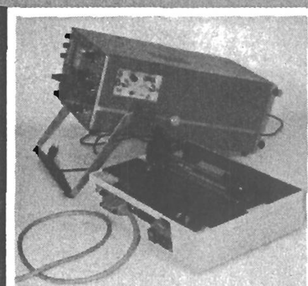
creative computing software

You Can Do It All with FLEXI PLUS™









Build a complete system or expand your Apple or other 6502-based system.

6809-BASED MICROCOMPUTER

FLOPPY DISK CONTROLLER

RS-232 COMMUNICATIONS

IEEE 488 BUS CONTROLLER

A remarkably flexible microcomputer board

FLEXI PLUS is a **6809-based single board microcomputer** with up to 56K of on-board memory, extensive serial and parallel I/O capability and a cassette interface. It may be used without the 6809 as an expansion board for most 6502, 6800 and 6809 systems. The **Floppy Disk Controller** supports up to four 8" drives or three 5¼" drives and provides IBM compatible formats. The fully buffered **RS-232 Communications Port** features programmable data formats and baud rates from 50 to 19,200. The **IEEE 488 Bus Controller** supports interfacing to sophisticated instrumentation and test equipment.

Microcomputer Features:
- State-of-the-art Motorola 6809E microprocessor
- Supports seven memory devices; Up to 56K bytes 2K, 4K and 8K RAMs, EPROMs or ROMs
- Cassette port handles many formats
- 20 mA current loop TTY port
- 6522 VIA for parallel/serial I/O
- Directly expandable with VIDEO PLUS and DRAM PLUS

Communications Features:
- Programmable baud rates from 50 to 19.2K baud
- Parity generation and checking
- Programmable word length and stop bits
- Full or half-duplex operation
- Full buffering on all lines

IEEE 488 Instrumentation Bus:
- Full implementation of IEEE standard
- Uses Motorola 68488 controller and 3448 buffers
- Standard 24-pin edge connector

Floppy Controller:
- WD 1791 supports IBM and other formats
- Up to four 8" Shugart compatible drives
- Up to three 5¼" Shugart compatible drives
- Includes fundamental disk operating software

Software Support:
- Includes a system monitor, device drivers and other basic software support
- We will be selling FLEX™, OS-9™ and/or other operating systems that support BASIC, Pascal, FORTH, word processing, assemblers, and many commercial software packages

FLEXI PLUS Base Price	**TCB-108**	**$320**
Floppy Disk Controller Option	TCX-931	125
6809 Microprocessor Option	TCX-932	75
RS-232 Communications Option	TCX-933	75
IEEE 488 Bus Controller Option	TCX-934	125

Add option prices to Base Price to obtain system price. FLEXI PLUS must be ordered with at least one option. Prices quoted are for US only. Add $3.00 surface postage in US. Please write for foreign pricing. Massachusetts residents add 5% sales tax.

OEM inquiries invited.

(FLEX is a trademark of Technical Systems Consultants)
(OS-9 is a trademark of Microware Systems Corporation)

Let us build your custom system.

THE COMPUTERIST®
34 Chelmsford St., Chelmsford, MA 01824
617/256-3649

MasterCard
VISA

Hayes Stack™

Microcomputer Component Systems

At Hayes, we don't believe in second best. Or planned obsolescence. We believe in taking the state of the art to the limit. Our new Smartmodem, for example, is the most sophisticated 300-baud originate/answer modem you can buy. And yet, it is perhaps the easiest-to-use modem ever.

RS-232C Compatible. Smartmodem lets any RS-232C compatible computer or terminal communicate by phone with other computers and time-sharing systems located *anywhere in North America*. You get full and half-duplex operation with both Touch-Tone* *and* pulse dialing.

Auto-Answer/Dial/Repeat. Smartmodem can answer the phone, dial a number, receive and transmit data, and then hang up the phone – automatically! If desired, Smartmodem will even repeat the last command. You can depend on Smartmodem for completely unattended operation.

Completely Programmable. Smartmodem can be controlled using *any* programming language. Over 30 different commands can be written into your programs or entered directly from your keyboard.

Smartmodem also includes several switch-selectable features that let you tailor performance to your exact needs. You can "set it and forget it" for the ultimate in convenience.

Built-in Audio Monitor. Thanks to an internal speaker, you can actually listen to your connection being made. You'll know immediately if the line is busy or if you reached a wrong number – *and you don't even need a phone!*

Status at a Glance. Seven LED's indicate Smartmodem's current operating mode: auto-answer, carrier detect, off hook, receive data, send data, terminal ready and modem ready. You're never left in the dark!

Direct-Connect Design. Smartmodem is FCC registered for direct connection to any modular phone jack – there's no acoustic coupler to cause signal loss and distortion.

Smartmodem, Smart Buy. Professional quality features. Versatile performance. A full two-year limited warranty. A suggested retail price of only $279.

What more could you want? Perhaps the matching Hayes Stack Chronograph, an RS-232C compatible calendar/clock system.

Check out the Smartmodem wherever fine computer products are sold. And don't settle for anything less than Hayes.

Hayes

Smartmodem. The ultimate concept in modems is now a reality.

Hayes Microcomputer Products Inc. 5835 Peachtree Corners East, Norcross, Georgia 30092 (404) 449-8791