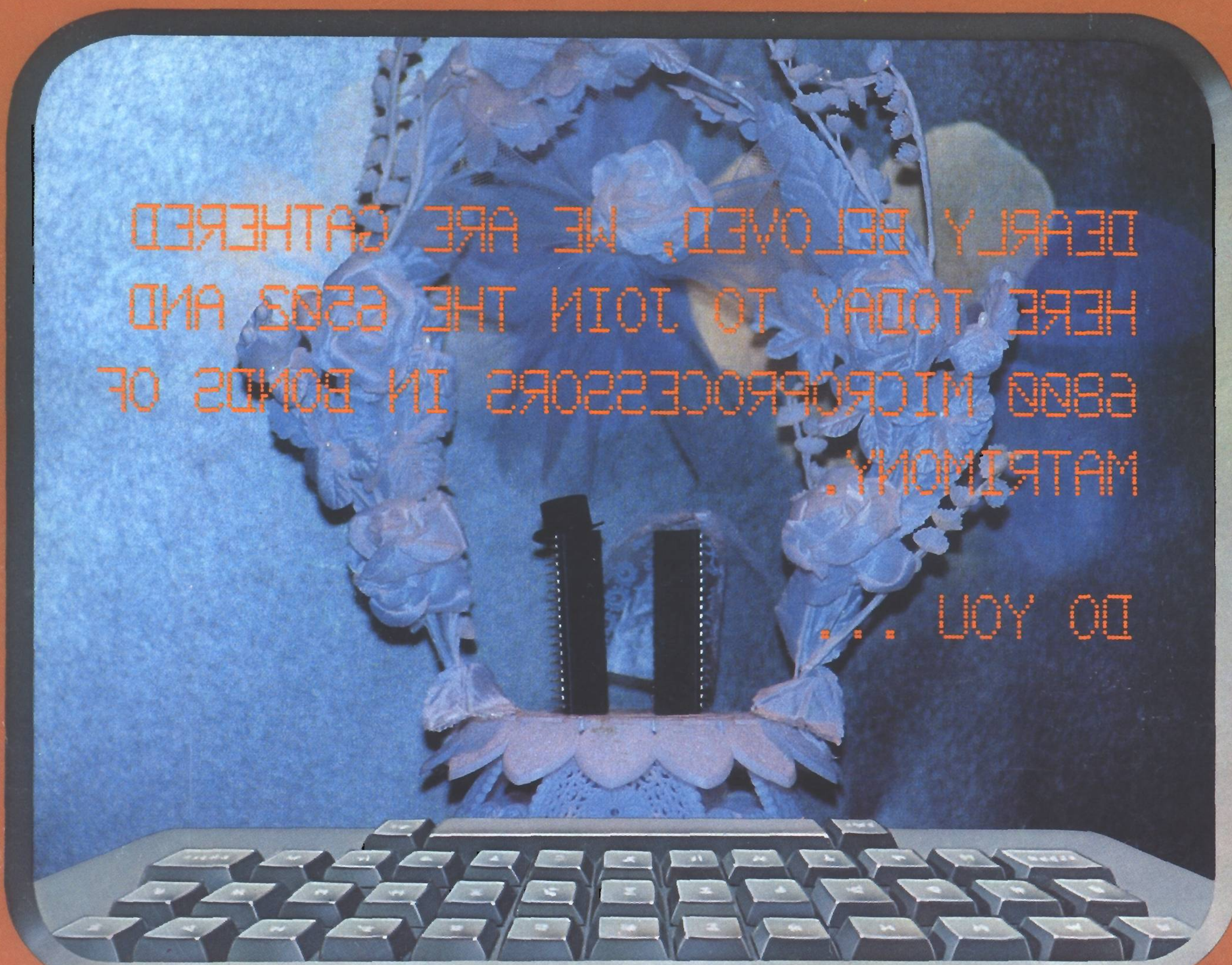
NO. 37 $2.50 JUNE 1981

MICRO™

THE 6502/6809 JOURNAL



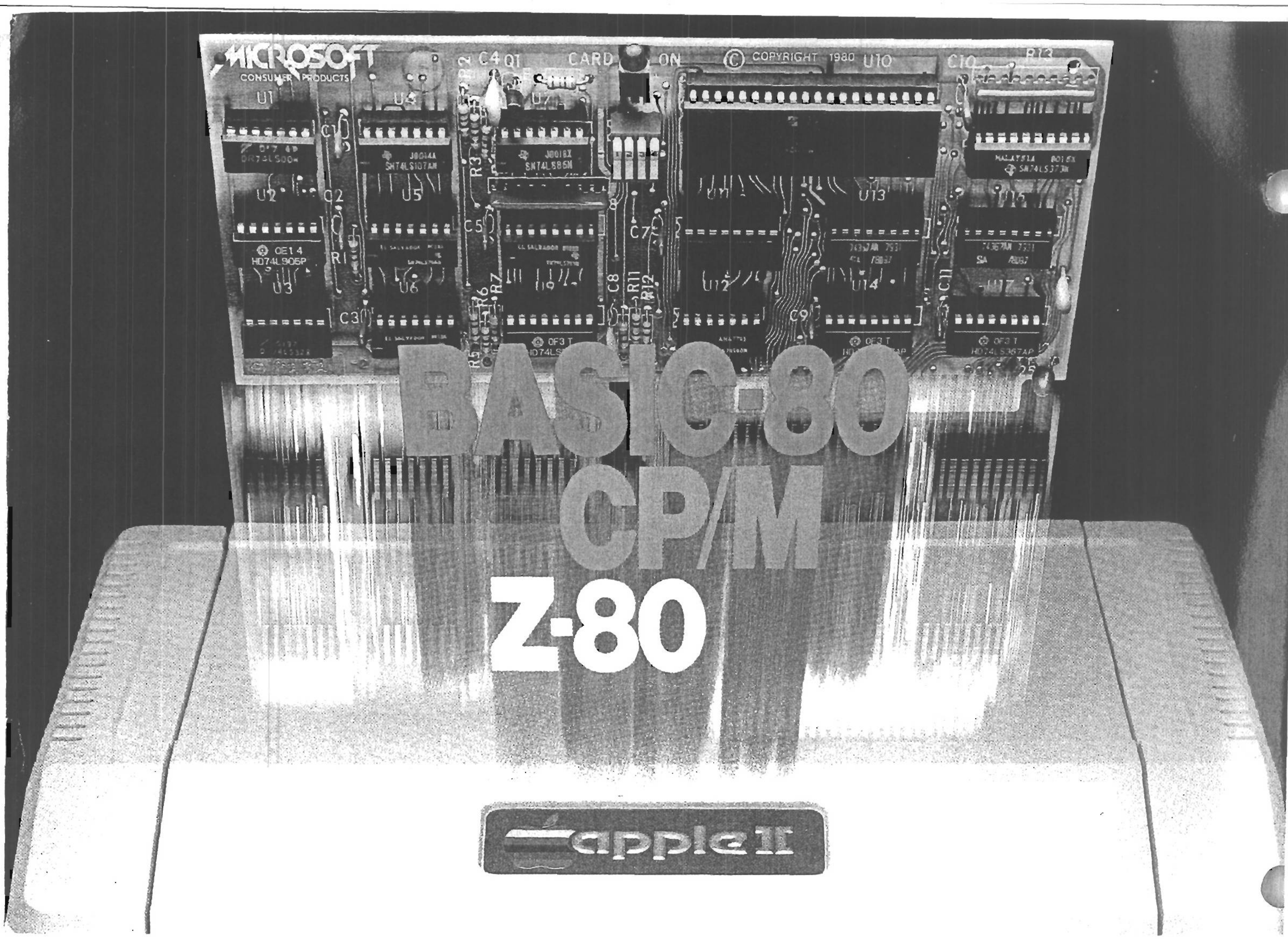
16-page bonus for Apple Computer users!

Macros for Micros

A C1P Dump Utility

Telephone Directory/Dialer for the AIM

Horizontal Screen Scrolling on the CBM/PET



Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

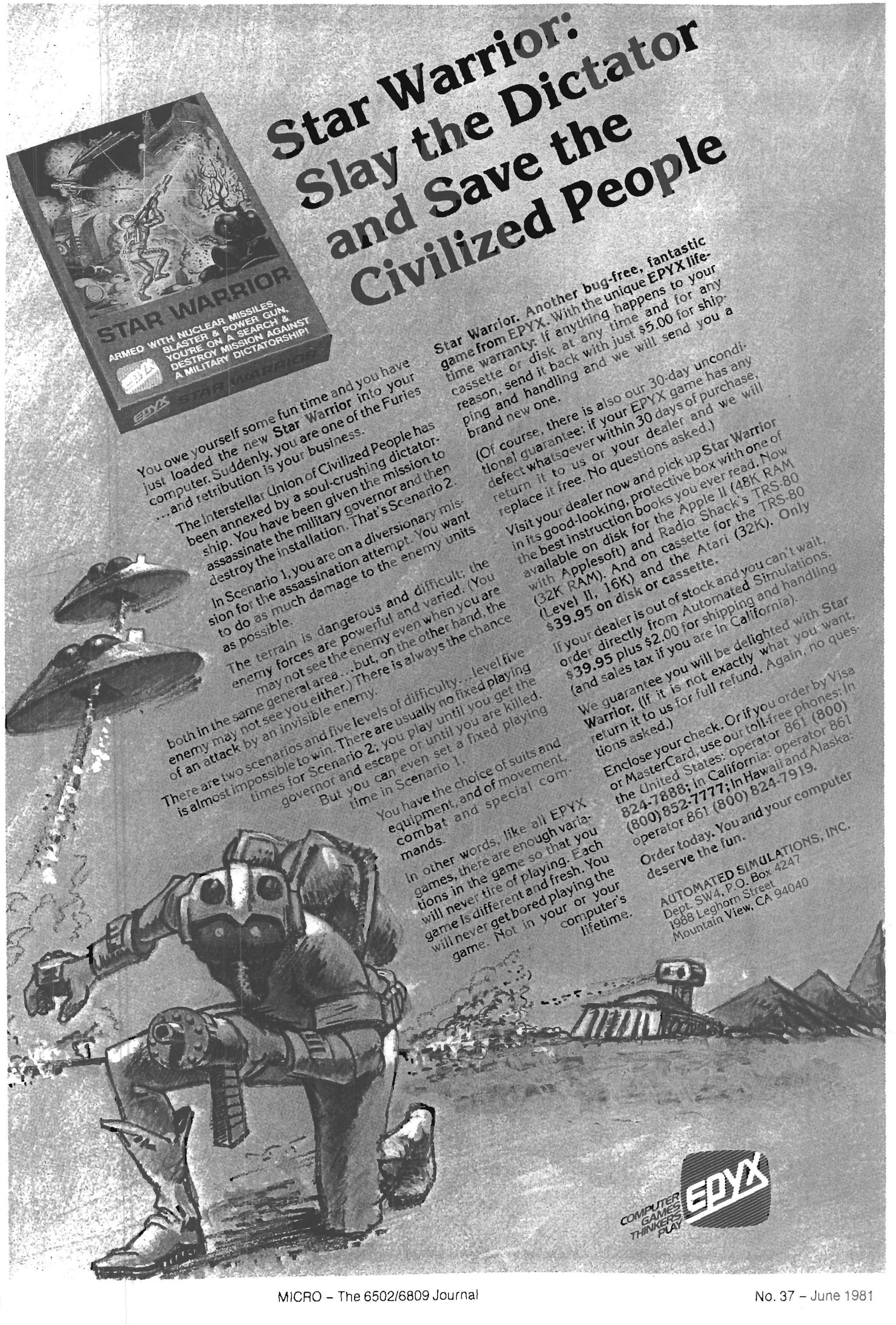
Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT
CONSUMER PRODUCTS

Microsoft Consumer Products, 400 108th Ave. N.E., Bellevue, WA 98004. (206) 454-1315


Star Warrior:
Slay the Dictator and Save the Civilized People
STAR WARRIOR
ARMED WITH NUCLEAR MISSILES, BLASTER & POWER GUN, YOU'RE ON A SEARCH & DESTROY MISSION AGAINST A MILITARY DICTATORSHIP!
EPYX
You owe yourself some fun time and you have just loaded the new Star Warrior into your computer. Suddenly, you are one of the Furies ...and retribution is your business.
The Interstellar Union of Civilized People has been annexed by a soul-crushing dictatorship. You have been given the mission to assassinate the military governor and then destroy the installation. That's Scenario 2.
In Scenario 1, you are on a diversionary mission for the assassination attempt. You want to do as much damage to the enemy units as possible.
The terrain is dangerous and difficult; the enemy forces are powerful and varied. (You may not see the enemy even when you are both in the same general area...but, on the other hand, the enemy may not see you either.) There is always the chance of an attack by an invisible enemy.
There are two scenarios and five levels of difficulty....level five is almost impossible to win. There are usually no fixed playing times for Scenario 2; you play until you get the governor and escape or until you are killed. But you can even set a fixed playing time in Scenario 1.
You have the choice of suits and equipment, and of movement, combat and special commands.
In other words, like all EPYX games, there are enough variations in the game so that you will never tire of playing. Each game is different and fresh. You will never get bored playing the game. Not in your or your computer's lifetime.
Star Warrior. Another bug-free, fantastic game from EPYX. With the unique EPYX lifetime warranty: If anything happens to your cassette or disk at any time and for any reason, send it back with just $5.00 for shipping and handling and we will send you a brand new one.
(Of course, there is also our 30-day unconditional guarantee: if your EPYX game has any defect whatsoever within 30 days of purchase, return it to us or your dealer and we will replace it free. No questions asked.)
Visit your dealer now and pick up Star Warrior in its good-looking, protective box with one of the best instruction books you ever read. Now available on disk for the Apple II (48K RAM with Applesoft) and Radio Shack's TRS-80 (32K RAM). And on cassette for the TRS-80 (Level II, 16K) and the Atari (32K). Only $39.95 on disk or cassette.
If your dealer is out of stock and you can't wait, order directly from Automated Simulations. $39.95 plus $2.00 for shipping and handling (and sales tax if you are in California).
We guarantee you will be delighted with Star Warrior. (If it is not exactly what you want, return it to us for full refund. Again, no questions asked.)
Enclose your check. Or if you order by Visa or MasterCard, use our toll-free phones: In the United States: operator 861 (800) 824-7888; In California: operator 861 (800) 852-7777; In Hawaii and Alaska: operator 861 (800) 824-7919.
Order today. You and your computer deserve the fun.
AUTOMATED SIMULATIONS, INC.
Dept. SW4, P.O. Box 4247
1988 Leghorn Street
Mountain View, CA 94040
COMPUTER GAMES THINKERS PLAY
EPYX

MICRO™

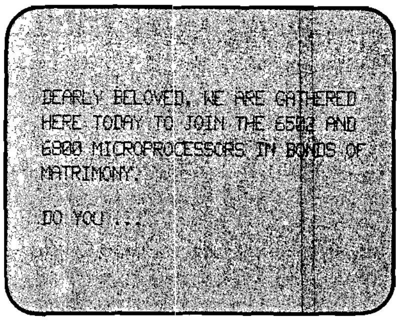
THE 6502/6809 JOURNAL

ARTICLES

APPLE BONUS

DEPARTMENTS

About the Cover



A Marriage Made in Arizona

This cover depicts the joining of the 6502 and the 6800. The offspring, the 6809, combines the second accumulator, the 16-bit index register and the 16-bit stack of the 6800 with the second index register and improved addressing modes of the 6502. It then adds its own unique new capabilities, including an additional 16-bit stack pointer, a multiply instruction, a number of 16-bit operations, a fantastic Load Effective Address instruction, and many other improvements which make it superior to either of its parents. Hopefully, the generation gap is minimal and can be overcome. It will take willingness to invest a little time in learning how the new generation "thinks" and in getting familiar with its "slang."

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

MICRO to Cover the 6809

The first four volumes of MICRO were devoted strictly to covering the 6502 microprocessor, and microcomputers based upon the 6502. Starting with this issue, which is the beginning of volume 5, MICRO will expand its range to include the Motorola 6809 microprocessor and microcomputers based upon it. The reason for this expanded coverage is simple. While the 6502 is a very good microprocessor and will continue to be a major force in the micro world for some time to come, it does have certain limitations, and over a period of time will become less and less competitive. For years we have hoped that MOS Technology, Synertek or Rockwell International, the three manufacturers of the 6502, would produce an improved 6502. At this time it seems unlikely that this will happen. None of the three have announced any new 8-bit upgrade of the 6502, and to do so at this late date would probably be a mistake. It takes a great deal of time and effort to produce a new microprocessor, and even more time to generate the most basic support required: editors, assemblers, language compilers and interpreters, business packages and so on. MICRO feels that it is simply too late for a new 6502-based product. So, what is the alternative? Do MICRO and its readers sit helplessly, watching the rest of the world move on to better micros? We think not. There is a very viable alternative — the 6809.

This microprocessor is very closely related to the 6502. Both are direct descendents of the 6800. They have a very similar basic architecture, compatible instructions, almost identical address, data and control signals, and much more. In fact, if someone had designed a "better 6502," it would probably have come out looking very much like the 6809. The first of a series of articles written to introduce the MICRO readership to the 6809 appears in this issue. Subsequent articles will go into greater detail about this device.

The 6809 is not "brand new." It has been around for a year or two and does have a reasonable amount of support. It is very quickly finding its way into the 6502 world. Synertek Systems has announced an update kit that converts a SYM-1 to run with the 6809. The kit includes a 6809 version of the SYM monitor in ROM as well as the 6809 and supporting circuitry. Stellation II has announced an add-on for the Apple which permits the Apple to run with both the 6502 and the 6809. Commodore has just announced a new product, "Micro-Mainframe", which is a 6809-based system with extensive software packages including interpreters for BASIC, Pascal, FORTRAN and APL; an editor; operating system; and an assembly language development system. The Computerist Inc. has announced a system which may use the 6502, 6809, or both.

We expect that this is just the start of a whole new generation of microcomputers, based on the 6809, but related to the current 6502 system. MICRO readers should keep abreast of these developments and should become familiar with the 6809. MICRO will do its part by presenting introductory articles about the 6809 and by keeping you informed on all related developments. If you are working on a 6809-based system already, we are interested in reviewing articles about your system.

A Quick Reference

I told you things were happening fast in the 6809 world. Just today, as this issue goes to the printer, I received a new book: *6809 Microcomputer Programming & Interfacing With Experiments*, by Andrew C. Staugaard, Jr. It is published by Howard W. Sams & Co., Inc. and lists at $13.95. I have not had time to give it more than a quick "once-over", but it looks very informative.

The Perfect MICRO

Since MICRO has grown so much in physical size over the past year, and since we expect more growth in the coming year, especially with the Bonus Sections, we have had to go to a different binding technique: Perfect Bound. This should provide a better product with less chance of covers tearing off. The three-hole punch will be maintained.

Robert M. Tripp

MICRO Letterbox

The following letters are in response to the March editorial (34:5).

Dear Editor:

Your March editorial concerning "copyright/copywrong" was an articulate plea for honesty and fairness in the use and abuse of "protected" material. While I personally agree with nearly everything the editorial stated, I emphatically *do not* agree with the conclusion you arrived at and I wholeheartedly disagree with the position you have taken.

I am appalled by the assumption you make that anyone who has a program that can copy a protected disk, tape, (whatever), will rush out and run off numerous copies for his friends and relatives (thereby reducing the potential market for the protected material). Where do you get the moxy to demean the *large majority* of your readers by suggesting they would act in such a manner? That theft exists I am willing to admit. Like you I condemn it unequivocally! It *does* and *has* forced vendors to increase the price to cover "copy wrong" losses. Your statement that theft "may" increase prices is generous to a fault. Those hidden costs (including the added cost in programming time and design effort to "protect" the program) are *already* included in the price. Valid users are already paying for the thieves' practice and for the disregard by vendors and editors who who protect themselves at the expense of the utility of the program(s).

I suggest the only real threat to the growth of the software market is the usability and convenience withheld from the end user. Programs that ignore the honest needs of the end user *ought* to face competition from a product that will provide that service to the user. To restrain that sort of competition is the worst disservice a magazine and its editor can do to its readership, its advertisers and the marketplace in general.

Dear Editor:

I am a computer dealer, and as such a software salesman. My own personal computer is an Apple II. Believe me, if I had had to buy every piece of software I have for the Apple, I would very likely never have become a dealer. I wasn't born with 1's and 0's for brain cells as so many computerists I know! My background is electronics. To "get up to speed" in the world of computers, I have worked my tail off through trial and error, reading what I was able to digest on the subject, but most of all running programs other people had written and observing what did and did not work. I freely admit there are many copyrighted programs in my library which I obtained through software swaps and from friends. If I were using any of these for commercial gain or was reselling them through any means, I should be locked up. The fact is that I, and every other computer acquaintance I have, uses whatever kind of quality programs available to learn more about how to write programs. Often as not, what is learned is how *not* to do something. There are some unbelievably atrocious programs out there which are advertised in your magazine and every other computer magazine. Why don't all these self-righteous people who had such a damned fit about your running the ad, get equally worked up about "programmers" asking and getting money for sheer junk?

There are some very good programs available for the Apple and, fortunately, they seem to be increasing in number. Trouble is, the advertisements look just the same whether the programs are any good or not. Since it is almost never possible to try a program before stocking it or buying it for personal use, I for one, will *never* buy a program which cannot be copied either with normal means or, at least, with a bit copier. I think anyone who spends good money for a piece of software should have the right to modify it, customize it, and put it on any number of disks he wishes. I *want* programmers to make money. I also want to own what I pay money for.

Thank you for running the ad and thank you for putting out one of the best computer magazines available today.

It's Time to Stop Dreaming

Since there is apparently not going to be an enhanced version of the 6502, it is time to stop dreaming about it. The 6809 is closely related to the 6502 and has many features which make it worth considering as an improved micro.

Robert M. Tripp
Editor/Publisher
MICRO

This is the first part of a MICRO series on the 6809 microprocessor. Part I covers an overview. Here we'll focus on the "new" chip's characteristics and merits. Future articles will discuss the chip in greater detail, including how to convert 6502-based hardware and software to 6809 systems.

A good programmer is never totally satisfied with his program. He always wonders if there are more improvements that could be made. Therefore, it is not surprising that ever since the first successful microprocessor was introduced, the 8080, computerists have been seeking improved devices. The Motorola 6800 was one direction of improvement, followed by its fairly direct descendent, the MOS Technology 6502. Even though MICRO was started to help promote the 6502 at a time when it was being virtually ignored by the microcomputer industry, we have always thought about the next generation, an improved 6502. Articles and letters in issues 23, 24, 26 and 34 of MICRO, plus numerous other material which never got into print, indicate that many of our readers are actively interested in the "dream machine," an improved microprocessor based on the 6502.

The time for dreaming has ended. There is now a microprocessor in the 6502 tradition with many of the improvements requested in the articles, and in our own considerations. It is *not* being made by MOS Technology, Synertek or Rockwell International, the three manufacturers of the 6502. None of these companies has announced any advance development based on the 6502. However, Motorola, the inventor and primary manufacturer of the 6800, has produced a microprocessor which can be considered the 6502 dream machine. The 6809 is based conceptually on the 6800 8-bit microprocessor. But then, so was the 6502. Since 6502 manufacturers do not seem interested in producing an improved version of the 6502, we suggest that the 6809 be seriously considered as the eventual successor to the 6502. This does not mean the 6502 is in any danger of disappearing overnight. It is a firmly established product with a lot of support and is actively being used by thousands of computerists. It will be around for quite a while. But, in this business, change and improvement are the standard, not the exception.

Why should we consider the 6809? Because it is very similar to the 6502 in its architecture and in many of its principles of operation. It is as much an extension of the 6502 as of the 6800, so let's examine its main features.

Architecture

The 6809's architecture is very similar to the 6502's. It has a 16-bit address space (64K bytes) and uses an 8-bit data bus. Its timing and control signals are almost identical to those of the 6502, so that most expansion boards will be compatible between the 6502 and the 6809 with little or no modification. Figure 1 — the registers of the 6502 and 6809 — shows the similarity between the two chips and some of the improvements in the 6809. The 6502 has one 8-bit accumulator (A) and the 6809 has two (A and B). The 6502 has two 8-bit index registers (X and Y); the 6809 has two 16-bit registers (also X and Y). The 6502 has a single stack located in page one, the 6809 has two stacks. One stack, like the 6502, services hardware requirements (interrupts, JSRs). A second stack is not affected by any hardware conditions. Each stack has a 16-bit register so that it may be located anywhere in memory, and is not limited to a single page in length.

Several of the 6809's logical improvements include:

1. 16-bit X and Y index registers (8-bit on 6502) permitting the various indexing operations to operate anywhere in memory over the full 16-bit addressing range.

2. 16-bit stack register (9-bit on 6502) permitting the stack to be anywhere in memory and to be any size. The 6502 stack can only be 256 bytes maximum and must be on page one.

3. A second 16-bit stack is available for the user and is not affected by hardware operations such as interrupts and subroutine calls. The 6502 does not have a second stack.

The 6502 has a single 8-bit accumulator. The 6809 has two 8-bit accumulators which may be used as a single 16-bit accumulator for particular 16-bit operations. These operations include add, subtract, compare, load, store, transfer between registers and exchange between registers. This 16-bit capability makes the 6809 extremely powerful without adding 16-bit data bus hardware overhead.

The 6502 has a page zero addressing mode which permits fast addressing with one byte of address for data on the zero page. The 6809 has the same type of fast addressing but permits any page of memory to be the target page (direct page). A direct page register contains the address of the page to be accessed as the direct page. Any page can be made to act like the 6502 page zero, effectively providing 256 "page zeros."

Instruction Set Improvements

With a few minor exceptions, the 6809 has all of the instructions of the 6502. It has a number of new instructions and is more consistent and uniform in its instruction/addressing structure. A number of instructions have been added to the accumulator operations for both A and B accumulators:

1. INC/DEC — increment or decrement either accumulator.

2. One's Complement (COM) and Two's Complement (NEG).

3. Multiply A times B with the result in A and B. This is an 8-bit unsigned multiply with a 16-bit result.

4. Add and Subtract without carry or borrow, as well as the normal add and subtract with carry or borrow.

5. Exchange (EXG) or Transfer (TFR) between any 8-bit registers.

6. Clear either accumulator.

The 16-bit accumulator operations are all new, and work on the combined A and B accumulators in what is addressed as the D register. The operations include:

1. Add and Subtract 16-bit.

2. Compare to memory.

3. Load and Store 16-bits from or to memory.

4. Transfer or Exchange between any 16-bit registers: X, Y, S, U or PC.

5. Push and Pull from either the S or U stacks.

The operations available to the six 16-bit registers offer great potential in developing more efficient programs. These operations include:

1. Compare X, Y, S or U with memory.

2. Exchange or Transfer any 16-bit register with any other 16-bit register.

3. Load or Store any 16-bit register except PC.

4. Push and Pull any 16-bit register to either stack.

5. And a very useful new instruction which loads the effective address of an operation into the X, Y, S or U register.

(This new function opens up a vast number of possibilities for position-independent code and other advanced techniques.)

All of the branches provided by the 6502 are included in the 6809, as well as signed and unsigned branches, a branch to subroutine and a branch always. These branches support position-independent code (PIC) and are therefore important. There is also a branch never, which I haven't figured out a use for yet. The branches may be limited, as on the 6502, to branch forward or back about 128 locations (short) or they may be double byte addresses which permit branching to any location in memory. No more "Branch out of Range" assembly errors!

Miscellaneous Instructions

Instead of having a number of independent operations to set or clear the condition codes as the 6502, the 6809 uses an ANDCC or ORCC to logically AND or OR the condition code register to set and clear bits. This permits any set of condition codes to be cleared or set in one instruction. The 6502 has one software interrupt (BRK) command. The 6809 has three separate software interrupts which may be used at different levels of the program and for debugging.

Addressing Modes

Probably the most significant improvements made in the 6809 are in the addressing modes. Many of the 6502 modes have been maintained, which is not too surprising since many of them are rather fundamental: Inherent, Immediate, Absolute (16-bit address), and others. Some have been modified, such as the Relative, which was limited to 8-bit on the 6502 but which can be 8- or 16-bit on the 6809. Some of the 6502 index/indirect modes have been eliminated in their 6502 form, but most can be easily generated by the new 6809 indexed modes. The indexed address modes include:

1. Zero offset in which the 16-bit index value is used as the complete address: LDA X would load the A register with the contents of the memory address contained in the 16-bit X register.

2. Constant offset in which the 16-bit index value plus a 5-, 8- or 16-bit immediate value is used as the effective address: LDA TEST,X would add the value of TEST to the contents of X and use this as the effective address.

3. Accumulator-Offset Indexed ad the contents of a specified accum lator to the contents of the specifi index register to form the effecti address: LDA B,X adds the 8-bit register to the 16-bit X register form the effective address.

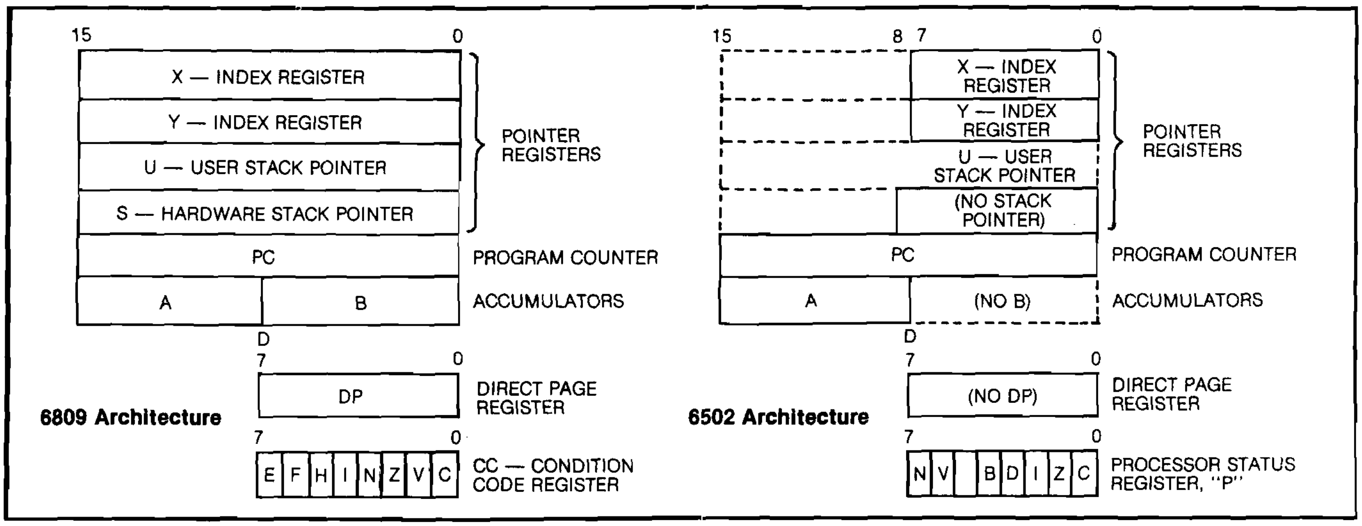
4. Auto Increment/Decrement Index is a form of the Zero Offset, but al increments or decrements the inde register one or two. This is useful scanning tables, data, and mar other operations on organized dat This mode permits the X and Y i dex registers to be used as addition software stacks.

5. Indexed Indirect — Most of th index modes permit a level of in direct addressing. The indexing o curs first and the effective address the indexing operation is used t determine the location in memor which contains the final addres There is no simple Indirect Indexe as on the 6502, but this is easil accomplished by the indexin modes mentioned above.

As mentioned in the Branchin instructions, relative addressing ma be short (1 byte offset), as on the 6502 or long (2 byte offset). This greatly ex pands the capabilities of the branchin instructions. Another important new addressing mode is Program Counte Relative. One of the difficulties in writing position-independent code (PIC) on the 6502 is that when the code moves, any tables or other data which move with the code lose their absolute addresses. With Program Counter Relative addressing, the addresses of the table or data are calculated relative to the current Program Counter, so that the addresses' relationship between the instruction and the table or data is preserved when they are moved together.

6809 Support

No matter how fantastic a microprocessor chip is, it is virtually useless without hardware and software support. The success of the 6502 has been due in part to the success of the Apple II, PET, and other 6502-based microcomputers. While the 6809 is the "new chip in town," it does have some solid initial support. Although the average MICRO reader may want to wait awhile longer before seriously considering a 6809-based system, the paragraphs below provide some insight into what is currently available.



Hardware

There are a number of hardware devices available. Two are add-ons to existing 6502-based systems. Synertek Systems has a plug-in module which converts the standard SYM-1 into a 6809-based system. It has a monitor equivalent to the 6502 version. This is perhaps the cheapest way to experiment with a 6809 system, particularly if you already own the SYM-1. Stellation Two has "THE MILL," an add-on to the Apple II which permits you to use both the Apple on-board 6502 and the additional 6809. To quote from Stellation's literature:

> The 6809 runs at its rated speed of 1MHz at the same time the 6502 is running at 20% of its rated speed. This allows the 6809 to perform time-critical tasks which are being controlled by the 6502. The control program can do all the slow speed operator interaction, and may even be written in the Apple's native BASIC.

Several complete systems are currently available. Motorola has an M6809 Monoboard Microcomputer and a Micromodule 19 (M68MM19) for the EXORcisor system. Canon's CX-1 is a 6809 video/floppy desktop computer with up to 96 kilobytes RAM, and supports DOS, BASIC, and has an assembler. Smoke Signal Broadcasting, long involved in the 6800, has a system — 9822 — based on the 6809. Percom Data Company offers the LFD-800. I am sure that there are other systems currently available; we will mention them in future articles as the information reaches us.

In addition to the currently available systems, there are other developments in the works. Rumor, unconfirmed at this time, has it that the new Radio Shack color computer will be 6809-based. I saw an Hitachi 6980 color system at the West Coast Computer Faire in April. It is 6809-based (the system number may have been a typo!) and looked very sophisticated. It may be available this fall. The Computerist will be offering a board this summer which will have a floppy disk controller, IEEE-488 controller, ACIA controller, multiple VIAs, RAM, EPROM, cassette interface and a 6809 microprocessor. This may be used, with some form of terminal, as a stand-alone system, or may be used in conjunction with MICRO PLUS as a video-based 6809 system.

Software

Although the 6809 is relatively new, it is upwardly compatible with the older 6800 at the source level, so that much of the existing 6800 software can be readily converted to run on the 6809. This means that the time required to produce support software has been considerably reduced and a fair amount is already available. Motorola offers a broad range of development and support software including BASIC-M, an interactive compiler, 6809 Cross Macro Assembler and Linking Loader, resident Pascal Interpreter and a 6809 Realtime Multitasking System.

Technical System Consultants, long a provider of 6800-based software packages offers: FLEX™ Disk Operating System for SWTPc, EXORciser and general systems; UniFLEX™ Operating System; a BASIC Precompiler; Sort/ Merge Package; BASIC and Extended BASIC; a Text Editor; Mnemonic Assembler System; Cross Assembler; Test Processing System; FLEX Utilities; a Debug Package; and FLEX Diagnostics.

Another broad support software house is Microware Systems Corporation, which has a number of offerings, including: OS-9 Operating System, BASIC09, Stylograph word processing, OS-9 Macro Text Editor, OS-9 Interactive Assembler and OS-9 Interactive Debugger. Smoke Signal Broadcasting offers, in addition to its hardware, the following software: Assembler, Pascal, Forth, COBOL, FORTRAN, and a large number of application packages including A/P, A/R, Payroll, Inventory, Medical and more. Some other companies who have been listed as vendors of 6809 software, but whose catalogs have not been received in time for this article, include: Phoenix Digital, Software Dynamics, and Softech Microsystems, Inc.

Summary

It may be a little bit early for most MICRO readers to rush out and buy a 6809-based system, but it is definitely not too early to become aware of the relatively new 8-bit microprocessor which may well be the successor, over time, to the 6502. Readers who are active in microcomputer hardware and software development will certainly want to keep abreast of the happenings in this area. MICRO will be generating a series of articles to help readers become more aware of, and understand, the 6809. We invite and encourage anyone who has experience in using the 6809, and particularly in converting from 6502 to 6809, to consider writing about his experiences.

Editor's note: All companies developing 6809-based systems, or 6809-based software, are urged to send us related information to be included in a future resource list.

Programmable Character Generator for the CBM 2022 Printer

The CBM 2022 printer allows programmable characters, but the method provided is tedious. With this BASIC program, a special character can be designed on the screen. The special character codes are generated and can be stored on tape or disk in "dictionary" form for future use.

Roger C. Crites
11880 Rio Grande
St. Louis, MO 63138

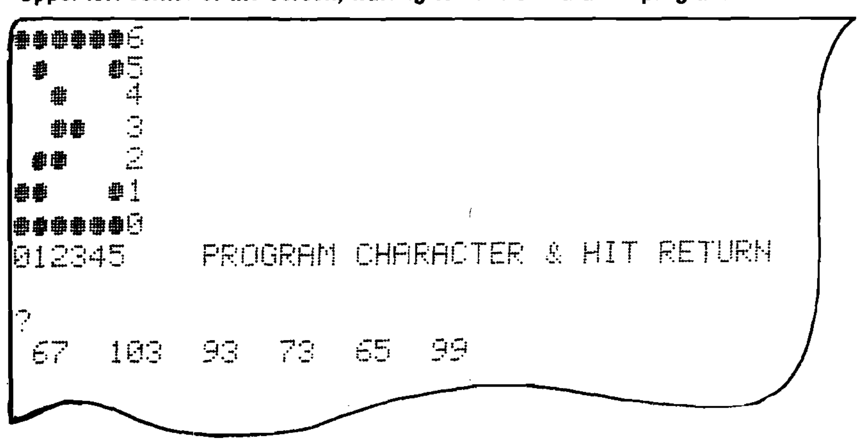
When I purchased my CBM 2022 printer I was impressed with the availability of programmable characters. I had visions of generating reports with the special math symbols, and charts with special plotting symbols. Text would be vertically, diagonally or otherwise aligned with chart axes. I would make dot plot printer art with subtle shading. I was going to really work the devil out of that programmable character.

Well, it always takes much longer to do anything than you think it will. When the new toy syndrome wore off and I was left with the work that I had bought the printer for in the first place, my enthusiasm over the programmable character fell. It was just too tedious to stop in the middle of a job and figure out the character code needed to achieve the effect desired. After all, it's more important to get the work out, plain, but finished, than to hit the deadline with a very snazzy half job. Before long I came to completely ignore the programmable character, but I never forgot it was there.

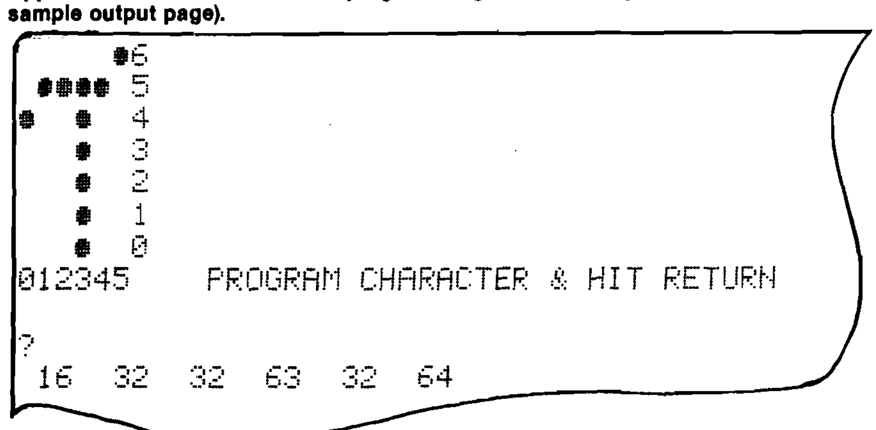
After a time I concluded that the bottleneck in the use of this capability was mostly due to the time required to figure out the special character codes. What I needed was an extensive dictionary of all the special codes that I expected to use. If all the character codes

Figure 1

Upper left corner of the screen, waiting to have a character programmed.



Upper left corner of screen after programming summation symbol (see 5th entry on sample output page).



Upper left corner of screen after programming greek letter Tau (see 4th entry on sample output page).

were known, they could be compiled into concise data sets—one for charts and one for text, etc. Stored on tape any "dictionary" could be merged with the current work file as a string array, PC$(I). From there it's down hill.

If a single special character is needed in a line, the required code is invoked by writing PC$(I) to printer secondary address 5, then inserting CHR$(254) in the print stream where needed. If multiple special characters are needed on a line it is a little more tricky. The printer only takes one programmable character at a time. To get more than that on the same line it is necessary to use a return without line feed. This is done by breaking the print string into several components. Each component must contain only one special character. Each component is output, inserting the required special character code in the correct place. The length of the output component is determined, the return code CHR$(141); is appended to the component and the resulting string printed. This prints the first component containing the first special character and returns without advancing the paper. The next special character is programmed as before, the length of this component determined, and CHR$(141); appended. Before outputting this component, however, it is necessary to prefix SPC(CL) to the output string.

CL is the sum of all previous component lengths. When this is output, the printer will space over the previous components, print the current component, and return without advancing the paper. This process is repeated until all components have been output. A blank print then advances the paper, ready for the next line. Admittedly this procedure is somewhat cumbersome, but once the necessary subroutine is worked out it can be implemented in most programs without further effort.

```
100 REM**********************************
110 REM***                            ***
120 REM***   PROGRAMMABLE CHARACTER   ***
130 REM***                            ***
140 REM***        PROGRAMMER          ***
150 REM***                            ***
160 REM***                            ***
170 REM**********************************
180 REM THIS PROGRAM PROGRAMS PROGRAMMABLE
190 REM CHARACTERS FOR THE CBM 2022 PRINTER
200 REM
210 OPEN 4,4:OPEN 5,4,5
220 OPEN6,4,6:PRINT#6,CHR$(16)
230 PRINT#4,CHR$(1)+"PROGRAMMABLE CHARACTERS"+CHR$(10)+CHR$(10)
240 PRINT"⊐";
250 PRINT"       6"
260 PRINT"       5"
270 PRINT"       4"
280 PRINT"       3"
290 PRINT"       2"
300 PRINT"       1"
310 PRINT"       0"
320 PRINT"012345    PROGRAM CHARACTER & HIT RETURN"
330 INPUT A$
340 IF A$="END"GOTO510
350 FORI=0TO5:C(I)=0:NEXTI
360 FORI=0TO5
370 FORJ=0TO6
380 X=PEEK(32768+40*J+I)
390 IF X<>32 THEN C(I)=C(I)+2↑(6-J)
400 NEXTJ
410 NEXTI
420 PRINT"[illegible]";
430 FOR I=0TO5:PRINTC(I);:NEXTI
440 P$=""
450 FOR I=0TO5:P$=P$+CHR$(C(I)):NEXT
460 PRINT#5,P$
470 PRINT#4,"┌─┐"
480 PRINT#4,"| "CHR$(254)" |"C(0);C(1);C(2);C(3);C(4);C(5)
490 PRINT#4,"└─┘"
500 GOTO240
510 REM** RESET PRINTER & STOP **
520 PRINT#6,CHR$(24)
```

PROGRAMMABLE CHARACTERS

⧗	65	99	119	127	107	99
δ	0	2	37	89	73	38
∡	3	4	8	24	36	67
Ƭ	16	32	32	63	32	64
Σ	67	103	93	73	65	99
ø	1	30	37	41	30	32
ᗺ	127	65	69	125	73	127
α	12	18	18	30	18	18
b	2	2	14	18	18	14
c	24	36	4	4	36	24
o	28	36	36	36	36	28
E	60	4	28	4	4	60
F	60	4	28	4	4	4
⊐	18	18	30	18	18	12
∞	24	20	20	24	20	24
∩	12	18	16	16	18	12
⊙	0	28	34	42	34	28
▷	0	127	34	20	8	0
⊡	63	33	45	45	33	63
±	0	17	17	125	17	17
♦	0	54	127	127	54	0
↓	4	2	127	2	4	0
←	8	28	42	8	8	8
→	8	8	8	42	28	8
↑	0	16	32	127	32	16
↖	120	96	80	72	4	2
↗	2	4	72	80	96	120
↙	15	3	5	9	16	32
↘	32	16	9	5	3	15
≈	10	20	20	10	10	20

After I had decided all this, the major task was compiling the special character "dictionary." To aid in this process I called on my PET. The result is a program to compute programmable character codes. With this program anyone (with a PET) can quickly generate a special character dictionary.

Before walking through the program, it will be helpful to review the process of programming a special character for the CBM 2022. The print head produces a 6-column by 7-row dot matrix. The rows are binary weighted starting from the bottom; i.e., 1,2,4,8,16,32,64. The dots to be turned on to form the character are chosen. Then binary weights associated with the chosen dots are summed column-by-column. The result is 6 sums, one for each column. If this is the Ith character and S1, S2, ..., S6 represent the 6 column sums, then PC$(I) = CHR$(S1) + CHR$(S2) + ... + CHR $(S6). For a more detailed description of the process refer to the CBM 2022 printer manual.

Now for the program. Line 210 opens files to the printer. File 4 is a general print file and file 5 is the character programmer in the printer. Line 220 adjusts the line spacing and lines 230-320 print a heading on the printer and form a 6 by 7 blank matrix on the screen. Line 330 waits for an input. If the input, A$ = "END", the program jumps to line 510, resets the line spacing and stops. To program a character, home the cursor. Then use the cursor controls to position the cursor, marking the dots (I use a space ball—shift Q) to form the desired special character. That is, you simply draw a picture of the desired character on the screen in the matrix outlined (see the examples). When you have completed the character, hit return.

Since A$ will not be "END", the program drops through to line 350. Lines 350-410 PEEK the character drawn on the screen and calculate the column codes necessary to program the character. Lines 440-490 print out the new special character and its column codes—one more entry in the dictionary. Line 500 loops back to repeat the process.

It should be pointed out that if lines 220 and 520 are omitted this program should also work for the CBM 2023.

The output (as shown for a page of random characters) is a convenient hard copy suitable for filing. Characters needed for any purpose are quickly selected from the dictionary and assembled into character string arrays as previously discussed.

With the aid of this approach to the programmable character, my printouts are finally beginning to benefit. I must admit, however, the results still fall short of my first imaginations. This may be the fault of human nature — reality seldom equals the imagination. In any case the CBM 2022 is capable of producing excellent results.

I suspect that there are others with CBM systems who would like to put the programmable character to work, but like myself have found the process too tedious to be practical. It is for them that I offer these reflections and the character generating program.

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 3×3 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.

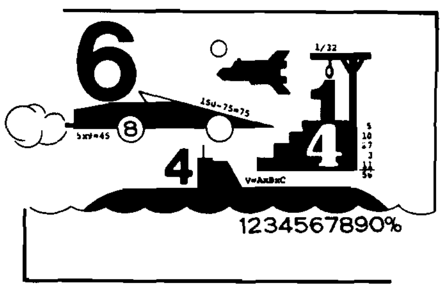
Car Jump—Make your stunt car jump the ramps. Each correct answer will increase the number of buses your car must jump over. These problems involve calculating the areas of different geometric figures.

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 Mountain, 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 opponent command opposing fleets of fighter-bombers armed with bombs and missiles. Your orders? Fly over the mountain and bomb the enemy blockhouse into dust!

Flying a bombing mission over that innocent looking mountain is no milk run. The opposition's aircraft can fire missiles at you or you may even be destroyed by the bombs as they drop. Desperate pilots may even ram your plane or plunge into your blockhouse, suicidally.

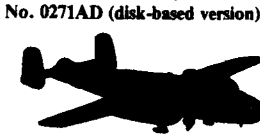
Flight personnel are sometimes forced to parachute from badly damaged aircraft. As they float helplessly to earth, they become targets for enemy missiles.

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

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

Minimum system requirements: An Apple II or Apple II Plus, with 32K RAM, one disk drive and game 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. 03458
603-924-7296

MICRO Club Circuit

Mike Rowe
Club Circuit
P.O. Box 6502
Chelmsford, MA 01824

The following club announcements are presented in zip code order.

Capital Area PET Enthusiasts (CAPE)
This group meets at the Patrick Henry Library, Route 123, in Vienna, Virginia, on the second Saturday of each month at 1:30 p.m. Robert C. Karpen is president, and membership now totals 40. The group's purpose is to exchange views, experiences and programs, and to discuss problems. For additional information, please write to:

CAPE
2054 Eakins St.
Reston, Virginia 22091

Basically Ohio Scientific Systems (B.O.S.S.)
This recently-formed club meets on the first Tuesday of each month at Sarasota Junior High School at 7:30 p.m. Its objectives include information sharing through the club's library, and demonstrations. B.O.S.S. is open to all current or prospective OSI owners. Dues are $12.00 per year. Area OSI owners interested in membership, and clubs interested in newsletter exchanges contact:

B.O.S.S.
P.O. Box 3695
Sarasota, Florida 33578

Rockford Area PET Users
Tom Storm is president of this 50-member group. It meets on the second Thursday of each month at 7:00 p.m. at Rock Valley College. The group's purpose is the general exchange of ideas on programming for the PET. If interested, please contact:

Mark J. Niggemann
912 St. Andrew's Way
Rockford, Illinois 61107

Sorbus Komputer Club (O.K.C.)
The purpose of this group is to help members learn programming techniques. Charles Olson is president and meetings are held every Thursday. For additional information contact:

Jim Johannes
1411 Classen Blvd.
Suite 348
Oklahoma City, OK 73106

New Braunfels 6502 Club
Informal meetings are held on the 4th Tuesday of each month at members' homes. David Sarkozi is the president, and membership stands at 15. The purpose of this club is to trade software and hardware ideas and to assist members having problems with either. For additional information, please contact:

David Sarkozi
171 Louisiana
New Braunfels, TX 78130

Bay Area Atari Users Group
Membership of this group now stands at 120, and Clyde H. Spencer is president. The group meets on the first Monday of each month at Foothill College. Newsletter is $12/year, and the aim of the group is to share and disseminate information about the Atari personal computer. For information write c/o:

Foothill College
12345 El Monte Road
Los Altos Hills, California 94022

Forth Interest Group
Meets on the fourth Saturday at Noon. Membership is over 1200. The club puts out a publication called "Forth Dimensions." for more information, contact:

Roy Martens, Publisher
FORTH Interest Group
P.O. Box 1105
San Carlos, CA 94070
(415) 962-8653

Santa Cruz Apple Users' Group
Jim McCaig is president of the Santa Cruz Apple Users' Group. The group's 15 members meet every 2nd Sunday in Felton. Its purpose is to lend programming assistance and to aid beginners. For additional information contact:

"Jay" Schaffer, Secretary
345-32nd Avenue
Santa Cruz, California 95062

Ohio Scientific Users Group North
This group, begun in 1979, now ha members. They meet on the secon day of each month at 7:30 p.m. a Data Systems Plaza. Mike Mahon president, and the group's goal share information and ideas about computers and to publish a newsle If interested, please contact:

Valerie J. Mahoney
P.O. Box 14082
Portland, Oregon 97214

Niagra Region '6502' Micro Users
This group's purposes are to bu software library that members can row from, conducting presentatio 6502 micros and their aspects, and moting the club Newsletter c '6502'. Meetings include demon tions, seminars, workshops, lect sharing ideas and programs . Mee are held at the College of Educatio Catharines, Ontario. For more info tion, contact:

Dr. R. Crane
College of Education
St. Catharines, Ontario L2S
(416) 684-7201 ext. 433

British Apple Systems User Grou
This newly-formed group already over 300 members. They meet nightly, just north of London, publish a bi-monthly newslette well as software disks. Martin Pe the Club's secretary. For more info tion, please contact him

c/o British Apple Systems User Group
P.O. Box 174
Watford, WD2 6NF
England

PET Users in West Lancashire
This group meets on the third T day of each month at 7 p.m. a Arnold School in Blackpool. The g has 32 members, with David Jo serving as president. For more contact:

David Jowett
PET Users in West Lancash
197 Victoria Road East
Thornton, Blackpool
FY5 3ST England

MICRO offers a free one year subs tion to all clubs registered with us registration form write to:
MICRO Club Circuit
Box 6502
Chelmsford, MA 01824

Musical Duets on the Apple II

Music generated by the Apple II, without extra firmware, is usually limited to one voice. Here are two Applesoft programs which, with the help of an ordinary amplifier, add a new dimension to Apple music — harmony.

Rick Brown
8903 Nogal Ave.
Whittier, California 90606

Anyone who has ever done any serious game-playing on the Apple II surely realizes how a catchy tune played through the Apple's speaker can enhance a program. A short machine language program is all that is needed to generate notes with a wide range of frequencies and durations. Such a tone-generating program is very nice, but it has the drawback of generating only one voice, which is to say, only one note at any given time can be played through the speaker. The usual way to acquire extra voices is to open the piggy bank and buy a music board or some other peripheral device designed for synthesizing music. For the serious music lover, it may be that nothing less will do. But can anything be done to satisfy the rest of us, whose standards (or finances) may not be as high? I chose to try to add, through software, a second voice to the Apple.

Now, before we go further, a little information about how a tone-generating program works is in order.The assembly language instruction LDA $C030 will toggle the Apple's speaker once every time it is executed, resulting in a little "click." Any sound whatsoever coming from the speaker is nothing but a series of such clicks, and the nature of the sound depends only on the interval of time between one click and the next. In the simplest case, this time interval is constant, and a steady, single-frequency, "pure" tone is generated. One convenient way to control the length of the pause between clicks is to use a "do-nothing" loop in the program, which generates a pause that is proportional to the number of times the loop is executed. The longer the pause between clicks, the lower the frequency of the resultant tone.

It occurred to me that it might be possible, by interleaving two such "do-nothing" loops, to superimpose one tone upon another and thus create the Apple's second voice. Consider two tones, one with a frequency of 500 Hz, and the other with a frequency of 300 Hz. To generate the first, we make the speaker click at intervals of 0.002s (s = seconds); that is, at these instants: 0.000s, 0.002s, 0.004s, 0.008s, 0.010s, etc.

Similarly, the 300 Hz tone would click at these instants: 0.0000s, 0.0033s, 0.0067s, 0.0100s, etc. Now, to generate both tones simultaneously, we should (it would seem) click the speaker at these instants: 0s, 0.002s, 0.0033s, 0.004s, 0.0067s, 0.008s, 0.01s, and so on. The problem of the two tones "clicking" at the same instant (e.g., at 0s and at 0.01s) is taken care of by a sort of "phase shift" inherent in the way the two "do-nothing" loops are interleaved.

Well, it all looks good on paper, and it might even work, were we using sinusoidally varying pulses instead of instantaneous clicks. But in fact, what results from the above technique is one of the most awful noises I've ever heard coming from the Apple speaker.

A More Promising Technique

All is not lost. There is another assembly language instruction, LDA $C020, which toggles not the speaker, but the cassette output. This produces a "click" on a cassette recording, or, if the output jack is connected to an amplifier, an audible click is produced. This is the secret to the second voice. There are several ways to amplify the signal. Perhaps the simplest is to plug an external speaker into your cassette recorder, and set the recorder in the "record" mode. Then, any input to the microphone jack will be amplified through the external speaker. Alternatively, you could patch from the cassette output jack to the computer to the auxiliary input of a stereo set. This method will probably give you more control over volume and tone. Now, by clicking the Apple speaker at a fixed interval, and clicking the alternate speaker at a different fixed interval, we can produce two distinct simultaneous tones. The Apple now harmonizes with itself!

Making Music

The core of the programs presented here is a machine language routine which generates two simultaneous notes of different pitches (P1 and P2), and different durations (D1 and D2). These notes are stored in two tables: one contains the melody and the other contains the harmony. After a note (either melody or harmony) is completed, the routine fetches the next pitch and duration from the appropriate table, and plays the next note. When a duration of zero is encountered in either table, the song is considered to be complete, and the machine language routine terminates. A listing of this routine is given in figure 1.

For each note, the pitch and duration take up one byte apiece. Thus there are 256 variations of pitch, and 255 possible durations (recall that a duration of zero will end the song). The value of P (the pitch) is proportional to the time delay between two successive "clicks" of the speaker, so that the highest values of P will produce the lowest notes. Because of this, P should be considered proportional to the wavelength, rather than to the frequency, of the note.

Although we have 256 wavelengths to choose from, most of them produce notes which are "between the keys of a piano." In other words, in order to make use of the isotonic scale to which we are accustomed, and in which music is commonly written, we must use only twelve notes per octave, and discard those values of P which produce non-isotonic notes. The range of 256 wavelengths available to us covers exactly eight octaves, and so the maximum number of isotonic notes we can use is 8 × 12, or 96. (In practice, the number is limited still further, as explained below.)

The ratio of wavelengths of two consecutive notes on the isotonic scale is a constant 2 ^ (1/12), or about 1.059, so that the ratio of wavelengths of two notes an octave apart is always 2:1. Thus wavelengths 128 and 64 are an octave apart, as are wavelengths 20 and 10, 2 and 1, and so forth. This fact imposes an obvious limitation on the higher notes.

Suppose we have a very high note—say of wavelength 4. The note one octave higher, then, has a wavelength of 2. Now, since the program uses only integers to represent wavelengths, it cannot generate the 11 isotonic notes between these two wavelengths (in fact, it can only generate one, corresponding to wavelength 3).

Another problem arising out of the use of integers for wavelengths is that the higher notes have an unavoidable tendency to go off-key. Suppose that the exact isotonic wavelength of a particular note (a low note, in this example) is calculated to be 154.43 on a scale from 1 to 256. This is rounded off to 154, creating a relative error of 0.29%. Consider now, a much higher note, whose exact wavelength is 15.43. This is rounded to 15, causing a much higher relative error of 2.8%, and it is this *relative* error (rather than the absolute error), which is detected by the ear.

Taking into account the limitations discussed earlier, I designed the program to use the lowest 65 isotonic notes available, covering a little more than five octaves, and using wavelengths from 6 to 256 (the latter wavelength is represented by zero in the routine). The highest notes are still a bit off-key, but generally they are rarely used and so won't create much of a problem. As far as the durations of the notes are concerned, they remain, as far as the ear can tell, faithfully proportional to their numerical values, throughout the range from 1 to 255.

***Figure 1:* The Two-Tone Generating Routine.**

```
0800             ;******************************
0800             ;*                            *
0800             ;* TWO-TONE GENERATING ROUTINE *
0800             ;*                            *
0800             ;*        BY RICK BROWN       *
0800             ;*                            *
0800             ;******************************
0800             ;*
0800             INDX1L EPZ $06
0800             INDX1H EPZ $07
0800             INDX2L EPZ $08
0800             INDX2H EPZ $09
0800             ;
0800             I      EQU $300
0800             P1     EQU $301
0800             D1     EQU $302
0800             P2     EQU $303
0800             D2     EQU $304
0800             I1L    EQU $305
0800             I1H    EQU $306
0800             I2L    EQU $307
0800             I2H    EQU $308
0800             ;
0309                    ORG $309
0309                    OBJ $800
0309             ;
0309 AD0503             LDA I1L             ;INITIALIZE
030C 8506               STA INDX1L          ;POINTERS
030E AD0603             LDA I1H             ;TO
0311 8507               STA INDX1H          ;BEGINNING
0313 AD0703             LDA I2L             ;ADDRESSES
0316 8508               STA INDX2L          ;OF
0318 AD0803             LDA I2H             ;NOTE
031B 8509               STA INDX2H          ;TABLES
031D A900               LDA #$00
031F 8D0003             STA I
0322 206003             JSR READ1           ;FETCH FIRST NOTE OF MELODY
0325 208403             JSR READ2           ;FETCH FIRST NOTE OF HARMONY
0328 CA          LBL1   DEX
0329 F007               BEQ TONE1
032B EA                 NOP                 ;THESE TWO INSTRUCTIONS CAUSE
032C AD1111             LDA $1111           ;A 6-CYCLE TIME DELAY
032F 4C3803             JMP LBL2
0332             ;
0332 AD30C0      TONE1  LDA $C030           ;CLICK SPEAKER AFTER P1 LOOPS
0335 AE0103             LDX P1              ;RESET X-REGISTER
0338 88          LBL2   DEY
0339 F007               BEQ TONE2
033B EA                 NOP                 ;THESE TWO INSTRUCTINS CAUSE
033C AD1111             LDA $1111           ;A 6-CYCLE TIME DELAY
033F 4C4803             JMP LBL3
0342             ;
0342 AD20C0      TONE2  LDA $C020           ;CLICK SPEAKER AFTER P2 LOOPS
0345 AC0303             LDY P2              ;RESET Y-REGISTER
0348 CE0003      LBL3   DEC I               ;AFTER 256 LOOPS, CHECK FOR END OF NOTE
034B D0DB               BNE LBL1
034D CE0203             DEC D1              ;END OF MELODY NOTE?
0350 D003               BNE LBL4            ;NO, CHECK HARMONY NOTE
0352 206003             JSR READ1           ;YES, FETCH NEXT NOTE OF MELODY
0355 CE0403      LBL4   DEC D2              ;END OF HARMONY NOTE?
0358 D0CE               BNE LBL1            ;NO, LOOP AGAIN
035A 208403             JSR READ2           ;YES, FETCH NEXT NOTE OF HARMONY
035D 4C2803             JMP LBL1            ;THEN LOOP AGAIN
0360             ;
0360 A200        READ1  LDX #$00
0362 A506               LDA INDX1L
0364 D002               BNE LBL5
0366 C607               DEC INDX1H
0368 C606        LBL5   DEC INDX1L
036A A106               LDA (INDX1L,X)
036C 8D0103             STA P1
036F A506               LDA INDX1L
0371 D002               BNE LBL6
0373 C607               DEC INDX1H
0375 C606        LBL6   DEC INDX1L
0377 A106               LDA (INDX1L,X)
0379 8D0203             STA D1              ;DURATION OF MELODY NOTE
037C D002               BNE LBL7
037E 68                 PLA                 ;IF D1=0, POP RETURN ADDRESS
037F 68                 PLA                 ;OFF STACK, SO RTS WILL END PROGRAM
0380 AE0103      LBL7   LDX P1
0383 60                 RTS
0384             ;
0384 A000        READ2  LDY #$00
0386 A508               LDA INDX2L
0388 D002               BNE LBL8
038A C609               DEC INDX2H
038C C608        LBL8   DEC INDX2L
038E B108               LDA (INDX2L),Y
0390 8D0303             STA P2              ;PITCH (WAVELENGTH) OF HARMONY NOTE
0393 A508               LDA INDX2L
0395 D002               BNE LBL9
0397 C609               DEC INDX2H
0399 C608        LBL9   DEC INDX2L
039B B108               LDA (INDX2L),Y
039D 8D0403             STA D2              ;DURATION OF HARMONY NOTE
03A0 D002               BNE LBL10
03A2 68                 PLA                 ;IF D2=0, POP RETURN ADDRESS
03A3 68                 PLA                 ;OFF STACK, SO RTS WILL END PROGRAM
03A4 AC0303      LBL10  LDY P2
03A7 60                 RTS
```

The Programs

Two programs are presented here, either of which can be used to play duets. However, the main purpose of the first program is to assemble the note tables from the data input by the user and to save the song on tape, while the second program is used only to load and play previously-recorded songs.

The Note-Table Assembler Program

This program provides an easy way to input a song, listen to it, edit it according to taste, and finally to save it on tape for later use. The song is input to the program through the use of DATA statements, which are typed in by the user each time the program is run. All such DATA statements must have line numbers greater than 690. The elements in these DATA statements will indicate the key signature (if any), the name and relative duration of each note, and the end of each part (melody or harmony) of the song. In order to facilitate the entry of these data, the notes are called by their alphabetic names (A,B,C,D,E,F,G) and converted by the program to the appropriate numerical values. The key signature, by default, determines whether a given note is to be played sharp, flat, or natural, but the signature may be overridden by appending the character "#" (sharp), "&" (flat), or "N" (natural) to the note's name.

Notes of different octaves are indicated by a single digit appended to the note name. If no such digit appears, octave 0 (zero) is assumed (this is the lowest octave which can be notated). Thus, G3 is one octave above G2, and D#1 is one octave above D#. The lowest letter-name within an octave is A, and the highest is G. Thus A2 is just a little above G1, while G#4 and A&5 designate the same note. A detailed description of the formats of the data elements is given below:

1. *Key Signature (optional):* If the music is written in a key other than C, the first two data elements should indicate the key signature. The first element should consist of the word "SHARP" or "FLAT", and the second element should be a string consisting of the letter names (in any order) of the notes to be sharped or flatted. Example:

   730 DATA FLAT,ADBE

2. *Note Names:* Each note name is an alphanumeric data item of the form XYM, where:

   X is one of the letters A, B, C, D, E, F, G, or R (rest)...

   Y is an optional character indicating sharp (#), flat (&), or natural (N). Any of these characters will override the key signature...

   M is a number from 0 to 9, indicating which octave the note belongs to. (However, the range within one song is limited to 65 notes, or about 5½ octaves.) M can be omitted if it equals zero.

   If X equals "R", then Y and M are omitted. Each note name must be followed by its note-duration.

3. *Note Duration:* This is a numerical quantity indicating the *relative* duration of the note that precedes it (the absolute duration will be calculated later). For example, if a

Figure 2: "Blue Bells of Scotland"

```
800  DATA G,1
801  DATA C1,2,B1,1,A1,1
802  DATA G,2,A1,1,B1,.5,C1,.5
803  DATA E,1,E,1,F,1,D,1
804  DATA C,3,G,1
805  DATA C1,2,B1,1,A1,1
806  DATA G,2,A1,1,B1,.5,C1,.5
807  DATA E,1,E,1,F,1,D,1
808  DATA C,3,G,1
809  DATA E,1,C,1,E,1,G,1
810  DATA C1,2,A1,1,B1,.5,C1,.5
811  DATA B1,1,G,1,A1,1,F#,1
812  DATA G,2,A1,1,B1,1
813  DATA C1,2,B1,1,A1,1
814  DATA G,2,A1,1,B1,.5,C1,.5
815  DATA E,1,E,1,F,1,D,1
816  DATA C,3
817  DATA END1
900  DATA R,1
901  DATA  R,1,E,1,F,1,F,1
902  DATA E,2,F,2
903  DATA G,1,C,1,D,1,F,1
904  DATA E,3,R,1
905  DATA R,1,E,1,F,1,F,1
906  DATA E,2,F,2
907  DATA G,1,C,1,D,1,F,1
908  DATA E,3,R,1
909  DATA C1,3,D1,1
910  DATA A1,2,F,1,G,.5,A1,.5
911  DATA D1,2,C1,2
912  DATA B1,1,D1,1,G,1,F,1
913  DATA E,2,F,1,F,1
914  DATA E,2,F,1,F,1
915  DATA G,1,C,1,D,1,F,1
916  DATA E,3
917  DATA END2
```

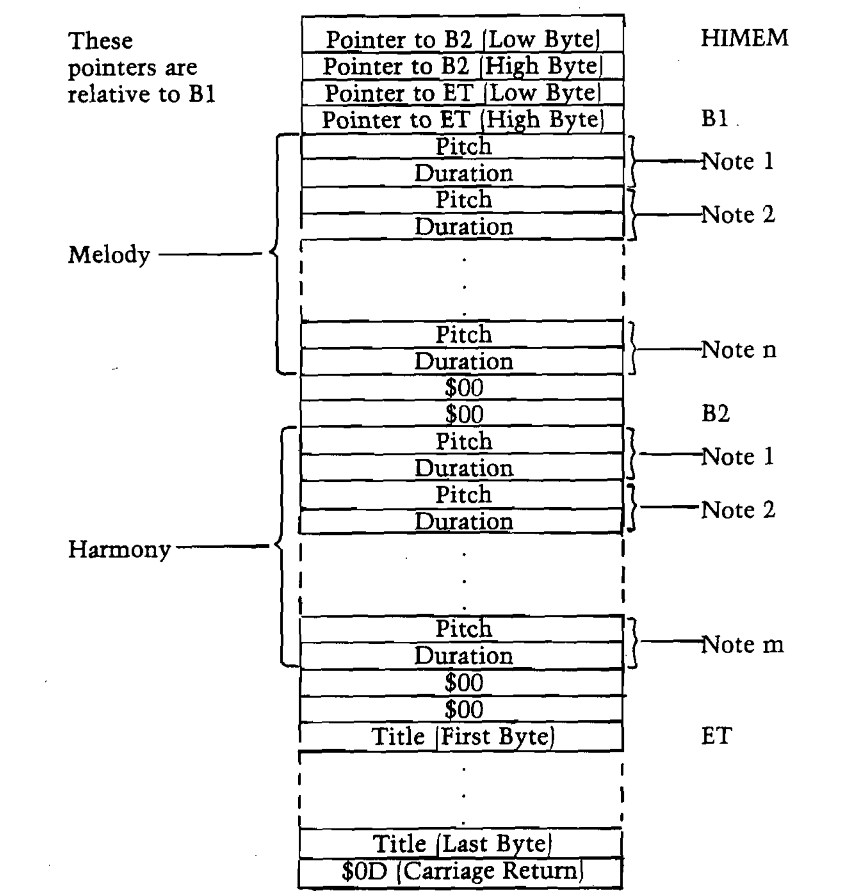


Figure 3: Structure of Note Tables for Duets

quarter-note is given a duration of 1, then a half-note would have a duration of 2, etc. Example:

740 DATA F1,.5,F#1,1,R,2,BN,1.5

4. *END1:* In a duet, the data element "END1" must follow the last note duration of the first part (melody) of the song.

5. *Second Part:* Note names and durations for the second part (harmony) of the song must follow "END1", in the format indicated in 2 and 3. The key signature (if any) is still in effect and should not be repeated here.

6. *END2:* The data element "END2" must follow the last note duration of the second part (harmony) of the song.

The above format applies to duets. There is also an option for entering and playing 1-part solos. To do this, enter key signature, note names and note durations for one part, as described above, but following the last note duration, enter the string "ENDSOLO" as the last data element. This will cause the same tune to be played through both speakers.

Running the Program

Before running the program as shown, you may find it necessary to change the value of M in line 10. HIMEM will be set to this value, which will be the highest byte occupied by the note tables, plus 1. The value shown in the listing is for a 32K system without DOS. Modify line 10 if necessary, then save the program on tape as shown (without any DATA statements).

Now, each time you load the program, type in the DATA statements according to the format explained above, remembering to give them line numbers higher than 690. Caution: for alphanumeric data, trailing blanks are considered to be part of the string, and may cause the data to be misinterpreted by the program. Avoid trailing blanks!

After all the necessary DATA statements have been entered, type "RUN". In a few seconds, you will see the prompt "TEMPO,KEY?" The tempo you input will be proportional to the *length* of the song, so that higher values will actually produce slower music. Notice that this is opposite from the usual interpretation of tempo. The tempo is multiplied by the relative note duration obtained from the DATA statement, the product is rounded to

MESSAGE	PROBABLE CAUSE
ILLEGAL QUANTITY ERROR	Tempo 0
BAD SUBSCRIPT ERROR	Illegal note name in DATA staten
OUT OF DATA ERROR	No "END2", or no "ENDSOLO"
SYNTAX ERROR	Bad DATA statement format; dat type mismatch
ERROR: KEY IS TOO HIGH	Key would cause notes to be outside of allowable range
ERROR: KEY IS TOO LOW	Key would cause notes to be outside of allowable range
ERROR: TEMPO IS TOO LONG	Tempo * Relative Duration 25 for some note
ERROR: INSUFFICIENT MEMORY FOR NOTE TABLES	DATA statements plus note table take up too much memory
WARNING: PART X IS XXX UNITS SHORTER THAN PART X. SONG WILL END EARLY.	The sums of the durations obtain from the DATA statements do no match. Song will play up to the end of the shorter part.
WARNING: DURATIONS OF SOME NOTES WERE ROUNDED TO THE NEAREST INTEGER. TUNES MAY NOT BE SYNCHRONIZED.	Tempo * Relative Duration does not equal an integer for some note(s).

***Table 1:* Error/Warning Messages**

```
0  REM      NOTE-TABLE ASSEMBLER
1  REM
2  REM
10 M = 32768: REM  M = SYSTEM'S CAPACITY
20 B1 = M - 4: HIMEM: M
30  DIM N%(65),P%(7)
40  DEF  FN HI(X) =  INT (X / 256)
50  DEF  FN LO(X) = X -  FN HI(X) * 256
55  REM  LOAD MACHINE LANGUAGE PROGRAM
60 P$ = "173005003133006173006003133007173007003133008173008003133009169O
    001410000030320960030321320032022400072341730170170760560031730481 92
    174001003136240007234173017017"
70  FOR I = 777 TO 830: POKE I, VAL ( MID$ (P$,3 * (I - 777) + 1,3)): NEX

80 P$ = "076072003173032192172003003206000003208219206002003208003032096O
    032060040032082060321320030760400031620001650062080021980071980O6161
    006141001003165006208002198"
90  FOR I = 831 TO 883: POKE I, VAL ( MID$ (P$,3 * (I - 831) + 1,3)): NEX

93 P$ = "007198006161006141002003208002104104174001003096160000165008208C
    021980091980081770081410030031650082080021980091980081770081410040O3
    208002104104172003003096"
95  FOR I = 884 TO 935: POKE I, VAL ( MID$ (P$,3 * (I - 884) + 1,3)): NEX

115 P$ = ""
120 N%(0) = 1:N%(1) = 0
125  REM   SET ISOTONIC WAVELENGTHS
130  FOR I = 2 TO 65
140 N%(I) = 256 / (2 ^ ((I - 1) / 12)) + .5
150  NEXT I
153  REM   ABCDEFG
```

```
155 P%(1) = 0:P%(2) = 2:P%(3) = 3:P%(4) = 5
156 P%(5) = 7:P%(6) = 8:P%(7) = 10
160 E = M -  FRE (0) + 200: HIMEM: E
170 B$ =  CHR$ (7) + "ERROR: "
180  RESTORE : INPUT "TEMPO,KEY? ";TM,K%:L = 0:F1 = 0
190  READ P$: IF P$ = "SHARP" OR P$ = "FLAT" THEN 680
200  RESTORE :LN = 0
210  FOR I = B1 - 1 TO E STEP  - 2
220  READ P$: IF  LEFT$ (P$,3) = "END" THEN 370
230  IF P$ = "R" THEN P = 0: GOTO 330
240 P = P%( ASC C(P$) - 64) + 12 *  VAL ( RIGHT$ (P$,1)) + K%
250 A$ =  MID$ (P$,2,1)
255  IF A$ = "N" THEN 310
260  IF A$ = "#" THEN P = P + 1: GOTO 310
270  IF A$ = "&" THEN P = P - 1: GOTO 310
280  IF LN = 0 THEN 310
290  FOR J = 1 TO LN
295  IF  MID$ (SF$,J,1) =  LEFT$ (P$,1) THEN P = P + Q: GOTO 310
300  NEXT
310  IF P < 1 THEN  PRINT B$;"KEY IS TOO LOW": GOTO 180
320  IF P > 65 THEN  PRINT B$;"KEY IS TOO HIGH": GOTO 180
330  READ DD:L = L + DD:DD = DD * TM:D =  INT (DD + .5)
340  IF D > 255 THEN  PRINT B$;"TEMPO IS TOO LONG": GOTO 180
350  IF D <  > DD THEN F1 = 1
355  REM  POKE PITCH,DURATION INTO NOTE TABLE
360  POKE I,N%(P): POKE I - 1,D: GOTO 390
370  POKE I,0: POKE I - 1,0
375  IF  LEFT$ (P$,7) = "ENDSOLO" THEN B2 = B1:ET = I - 2:L2 = L1: GOTO 4
     00
380  IF  LEFT$ (P$,4) = "END2" THEN ET = I - 2:L2 = L - L1: GOTO 400
385 B2 = I - 1:L1 = L
390  NEXT I: PRINT B$;"INSUFFICIENT MEMORY": PRINT "FOR NOTE TABLE  S": HIM
EM:
     M: END
400  POKE M - 1, FN LO(B1 - B2): POKE M - 2, FN HI(B1 - B2)
405  POKE M - 3, FN LO(B1 - ET): POKE M - 4, FN HI(B1 - ET)
410  IF L1 <  > L2 THEN SH = .5 * (3 -  SGN (L2 - L1)): PRINT : PRINT "WA
     RNING: PART ";SH;" IS "; ABS (L1 - L2);" UNITS SHORTER": PRINT "THAN
      PART ";3 - SH;".  SONG WILL END EARLY."
420  IF F1 THEN  PRINT : PRINT "WARNING: DURATIONS OF SOME NOTES WERE": PRI
NT
     "ROUNDED TO THE NEAREST INTEGER. TUNES": PRINT "MAY NOT BE SYNCHRONI
     ZED."
430  POKE 773, FN LO(B1): POKE 774, FN HI(B1)
440  POKE 775, FN LO(B2): POKE 776, FN HI(B2)
450  PRINT : INPUT COM$
460  IF COM$ <  > "GO" THEN 500
470  INPUT "REPETITIONS? ";R
480  FOR I = 1 TO R
490  CALL 777: NEXT I: GOTO 450
500  IF COM$ = "CHANGE" THEN 180
510  IF COM$ = "EDIT" THEN  HIMEM: M: LIST 691,: END
520  IF COM$ <  > "SAVE" THEN  PRINT "WHAT?": GOTO 450
530 J = ET - E: IF J > 255 THEN J = 255
535  PRINT "TITLE(1-";J;" CHARACTERS):"
540  FOR I = ET TO ET - J STEP  - 1
550  GET P$: IF P$ =  CHR$ (8) THEN I = I + 1: PRINT " "; CHR$ (8); CHR$
     (8);: GOTO 550
555  IF P$ =  CHR$ (21) THEN 550
557  IF P$ =  CHR$ (24) THEN  PRINT  CHR$ (92): GOTO 535
560  PRINT P$;: POKE I, ASC (P$): IF P$ =  CHR$ (13) THEN 580
570  NEXT I: PRINT : PRINT B$;"TITLE TOO LONG": GOTO 530
580  HOME : PRINT
590  PRINT "AFTER ADJUSTING VOLUME, PRESS 'RECORD',"
600  PRINT "THEN HIT ANY KEY.": GET P$
610  HOME : VTAB 12: FLASH : HTAB 12: PRINT "<<RECORDING>>": NORMAL
615  REM  ADDRESS -307 IS MONITOR WRITE ROUTINE:
620  REM  LOCATIONS 60-63 POINT TO BEGINNING
625  REM  AND ENDING ADDRESS OF WRITE.
630  POKE 6, FN LO(M - 1 - I): POKE 7, FN HI(M - 1 - I)
640  POKE 60,6: POKE 61,0: POKE 62,7: POKE 63,0: CALL  - 307
650  POKE 60, FN LO(I): POKE 61, FN HI(I)
660  POKE 62, FN LO(M - 1): POKE 63, FN HI(M - 1): CALL  - 307
670  HOME : GOTO 450
680 Q = 1: IF P$ = "FLAT" THEN Q =  - 1
690  READ SF$:LN =  LEN (SF$): GOTO 210
```

the nearest integer, and the final value is POKEd into the note table. So, for best results, you should input a tempo which, when multiplied by the note duration, always yields an integer (thus avoiding any rounding error). In no case may the product of the tempo and the relative note duration exceed 255. A product of 255 will produce a note about 3.0 seconds long. All other durations are proportionally shorter.

The KEY is an integer value (positive, negative, or zero) indicating how many semitones the song will be shifted up or down on the isotonic scale. Thus, for example, a key of 22 is one octave (12 semitones) higher than a key of 10. If the input key causes any note to fall outside the available range of 65 notes, an error message will be given.

After the tempo and key have been input, the program begins assembling the note tables. As the program processes the DATA statements, error or warning messages may be given, generated either by the program or by Applesoft. These messages are described in detail in table 1.

Program Commands

After the note tables are assembled, you will be prompted with a question mark. In response to this, you may type one of the following commands:

GO plays the song, in harmony and stereo, with as many repetitions as desired. (Be sure your amplifier is properly connected.)

SWAP causes parts 1 and 2 to switch speakers. Before this command is executed, part 1 plays through the Apple speaker, part 2 through your amplifier. Another SWAP will restore the original speakers.

CHANGE allows you to change the tempo and key, and reassemble the note tables.

EDIT lists the DATA statements and ends the program, allowing you to modify the song.

SAVE requests a song title, then saves the title and the note tables on tape. Since the program uses the GET command to input the title, any characters may be input, including colons, commas, and quotes. A carriage return terminates the input and causes recording instructions to be displayed.

quarter-note is given a duration of 1, then a half-note would have a duration of 2, etc. Example:

740 DATA F1,.5,F#1,1,R,2,BN,1.5

4. *END1:* In a duet, the data element "END1" must follow the last note duration of the first part (melody) of the song.

5. *Second Part:* Note names and durations for the second part (harmony) of the song must follow "END1", in the format indicated in 2 and 3. The key signature (if any) is still in effect and should not be repeated here.

6. *END2:* The data element "END2" must follow the last note duration of the second part (harmony) of the song.

The above format applies to duets. There is also an option for entering and playing 1-part solos. To do this, enter key signature, note names and note durations for one part, as described above, but following the last note duration, enter the string "ENDSOLO" as the last data element. This will cause the same tune to be played through both speakers.

Running the Program

Before running the program as shown, you may find it necessary to change the value of M in line 10. HIMEM will be set to this value, which will be the highest byte occupied by the note tables, plus 1. The value shown in the listing is for a 32K system without DOS. Modify line 10 if necessary, then save the program on tape as shown (without any DATA statements).

Now, each time you load the program, type in the DATA statements according to the format explained above, remembering to give them line numbers higher than 690. Caution: for alphanumeric data, trailing blanks are considered to be part of the string, and may cause the data to be misinterpreted by the program. Avoid trailing blanks!

After all the necessary DATA statements have been entered, type "RUN". In a few seconds, you will see the prompt "TEMPO,KEY?" The tempo you input will be proportional to the *length* of the song, so that higher values will actually produce slower music. Notice that this is opposite from the usual interpretation of tempo. The tempo is multiplied by the relative note duration obtained from the DATA statement, the product is rounded to

MESSAGE	PROBABLE CAUSE
ILLEGAL QUANTITY ERROR	Tempo 0
BAD SUBSCRIPT ERROR	Illegal note name in DATA statemen
OUT OF DATA ERROR	No "END2", or no "ENDSOLO"
SYNTAX ERROR	Bad DATA statement format; data type mismatch
ERROR: KEY IS TOO HIGH	Key would cause notes to be outside of allowable range
ERROR: KEY IS TOO LOW	Key would cause notes to be outside of allowable range
ERROR: TEMPO IS TOO LONG	Tempo * Relative Duration 255 for some note
ERROR: INSUFFICIENT MEMORY FOR NOTE TABLES	DATA statements plus note tables take up too much memory
WARNING: PART X IS XXX UNITS SHORTER THAN PART X. SONG WILL END EARLY.	The sums of the durations obtained from the DATA statements do not match. Song will play up to the end of the shorter part.
WARNING: DURATIONS OF SOME NOTES WERE ROUNDED TO THE NEAREST INTEGER. TUNES MAY NOT BE SYNCHRONIZED.	Tempo * Relative Duration does not equal an integer for some note(s).

***Table 1:* Error/Warning Messages**

```
0 REM      NOTE-TABLE ASSEMBLER
1 REM
2 REM
10 M = 32768: REM  M = SYSTEM'S CAPACITY
20 B1 = M - 4: HIMEM: M
30  DIM N%(65),P%(7)
40  DEF  FN HI(X) =  INT (X / 256)
50  DEF  FN LO(X) = X -  FN HI(X) * 256
55  REM  LOAD MACHINE LANGUAGE PROGRAM
60 P$ = "1730050031330061730060031330071730070031330081730080031330091690
    0014100000303209600303213200320224000723417301701707605600317304819
    2
    174001003136240007234173017017"
70  FOR I = 777 TO 830: POKE I, VAL ( MID$ (P$,3 * (I - 777) + 1,3)): NEXT


80 P$ = "0760720031730321921720030032060000032082192060020032080030320960
    032060040032082060321320030760400031620001650062080021980071980061
    61
    006141001003165006208002198"
90  FOR I = 831 TO 883: POKE I, VAL ( MID$ (P$,3 * (I - 831) + 1,3)): NEXT


93 P$ = "0071980061610061410020032080021041041740010030961600001650082080
    021980091980081770081410030031650082080021980091980081770081410040
    03
    2080021041041720030030 96"
95  FOR I = 884 TO 935: POKE I, VAL ( MID$ (P$,3 * (I - 884) + 1,3)): NEXT


115 P$ = ""
120 N%(0) = 1:N%(1) = 0
125  REM   SET ISOTONIC WAVELENGTHS
130  FOR I = 2 TO 65
140 N%(I) = 256 / (2 ^ ((I - 1) / 12)) + .5
150  NEXT I
153  REM   ABCDEFG
```

```
155 P%(1) = 0:P%(2) = 2:P%(3) = 3:P%(4) = 5
156 P%(5) = 7:P%(6) = 8:P%(7) = 10
160 E = M -  FRE (0) + 200: HIMEM: E
170 B$ =  CHR$ (7) + "ERROR: "
180  RESTORE : INPUT "TEMPO,KEY? ";TM,K%:L = 0:F1 = 0
190  READ P$: IF P$ = "SHARP" OR P$ = "FLAT" THEN 680
200  RESTORE :LN = 0
210  FOR I = B1 - 1 TO E STEP  - 2
220  READ P$: IF  LEFT$ (P$,3) = "END" THEN 370
230  IF P$ = "R" THEN P = 0: GOTO 330
240 P = P%( ASC C(P$) - 64) + 12 *  VAL ( RIGHT$ (P$,1)) + K%
250 A$ =  MID$ (P$,2,1)
255  IF A$ = "N" THEN 310
260  IF A$ = "#" THEN P = P + 1: GOTO 310
270  IF A$ = "&" THEN P = P - 1: GOTO 310
280  IF LN = 0 THEN 310
290  FOR J = 1 TO LN
295  IF  MID$ (SF$,J,1) =  LEFT$ (P$,1) THEN P = P + Q: GOTO 310
300  NEXT
310  IF P < 1 THEN  PRINT B$;"KEY IS TOO LOW": GOTO 180
320  IF P > 65 THEN  PRINT B$;"KEY IS TOO HIGH": GOTO 180
330  READ DD:L = L + DD:DD = DD * TM:D =  INT (DD + .5)
340  IF D > 255 THEN  PRINT B$;"TEMPO IS TOO LONG": GOTO 180
350  IF D <  > DD THEN F1 = 1
355  REM  POKE PITCH,DURATION INTO NOTE TABLE
360  POKE I,N%(P): POKE I - 1,D: GOTO 390
370  POKE I,0: POKE I - 1,0
375  IF  LEFT$ (P$,7) = "ENDSOLO" THEN B2 = B1:ET = I - 2:L2 = L1: GOTO 4
     00
380  IF  LEFT$ (P$,4) = "END2" THEN ET = I - 2:L2 = L - L1: GOTO 400
385 B2 = I - 1:L1 = L
390  NEXT I: PRINT B$;"INSUFFICIENT MEMORY": PRINT "FOR NOTE TABLE  S": HIM
EM:
     M: END
400  POKE M - 1, FN LO(B1 - B2): POKE M - 2, FN HI(B1 - B2)
405  POKE M - 3, FN LO(B1 - ET): POKE M - 4, FN HI(B1 - ET)
410  IF L1 <  > L2 THEN SH = .5 * (3 -  SGN (L2 - L1)): PRINT : PRINT "WA
     RNING: PART ";SH;" IS "; ABS (L1 - L2);" UNITS SHORTER": PRINT "THAN
      PART ";3 - SH;".  SONG WILL END EARLY."
420  IF F1 THEN  PRINT : PRINT "WARNING: DURATIONS OF SOME NOTES WERE": PRI
NT
     "ROUNDED TO THE NEAREST INTEGER. TUNES": PRINT "MAY NOT BE SYNCHRONI
     ZED."
430  POKE 773, FN LO(B1): POKE 774, FN HI(B1)
440  POKE 775, FN LO(B2): POKE 776, FN HI(B2)
450  PRINT : INPUT COM$
460  IF COM$ <  > "GO" THEN 500
470  INPUT "REPETITIONS? ";R
480  FOR I = 1 TO R
490  CALL 777: NEXT I: GOTO 450
500  IF COM$ = "CHANGE" THEN 180
510  IF COM$ = "EDIT" THEN  HIMEM: M: LIST 691,: END
520  IF COM$ <  > "SAVE" THEN  PRINT "WHAT?": GOTO 450
530 J = ET - E: IF J > 255 THEN J = 255
535  PRINT "TITLE(1-";J;" CHARACTERS):"
540  FOR I = ET TO ET - J STEP  - 1
550  GET P$: IF P$ =  CHR$ (8) THEN I = I + 1: PRINT " "; CHR$ (8); CHR$
     (8);: GOTO 550
555  IF P$ =  CHR$ (21) THEN 550
557  IF P$ =  CHR$ (24) THEN  PRINT  CHR$ (92): GOTO 535
560  PRINT P$;: POKE I, ASC (P$): IF P$ =  CHR$ (13) THEN 580
570  NEXT I: PRINT : PRINT B$;"TITLE TOO LONG": GOTO 530
580  HOME : PRINT
590  PRINT "AFTER ADJUSTING VOLUME, PRESS 'RECORD',"
600  PRINT "THEN HIT ANY KEY.": GET P$
610  HOME : VTAB 12: FLASH : HTAB 12: PRINT "<<RECORDING>>": NORMAL
615  REM  ADDRESS -307 IS MONITOR WRITE ROUTINE:
620  REM  LOCATIONS 60-63 POINT TO BEGINNING
625  REM  AND ENDING ADDRESS OF WRITE.
630  POKE 6, FN LO(M - 1 - I): POKE 7, FN HI(M - 1 - I)
640  POKE 60,6: POKE 61,0: POKE 62,7: POKE 63,0: CALL  - 307
650  POKE 60, FN LO(I): POKE 61, FN HI(I)
660  POKE 62, FN LO(M - 1): POKE 63, FN HI(M - 1): CALL  - 307
670  HOME : GOTO 450
680 Q = 1: IF P$ = "FLAT" THEN Q =  - 1
690  READ SF$:LN =  LEN (SF$): GOTO 210
```

the nearest integer, and the final value is POKEd into the note table. So, for best results, you should input a tempo which, when multiplied by the note duration, always yields an integer (thus avoiding any rounding error). In no case may the product of the tempo and the relative note duration exceed 255. A product of 255 will produce a note about 3.0 seconds long. All other durations are proportionally shorter.

The KEY is an integer value (positive, negative, or zero) indicating how many semitones the song will be shifted up or down on the isotonic scale. Thus, for example, a key of 22 is one octave (12 semitones) higher than a key of 10. If the input key causes any note to fall outside the available range of 65 notes, an error message will be given.

After the tempo and key have been input, the program begins assembling the note tables. As the program processes the DATA statements, error or warning messages may be given, generated either by the program or by Applesoft. These messages are described in detail in table 1.

Program Commands

After the note tables are assembled, you will be prompted with a question mark. In response to this, you may type one of the following commands:

GO plays the song, in harmony and stereo, with as many repetitions as desired. (Be sure your amplifier is properly connected.)

SWAP causes parts 1 and 2 to switch speakers. Before this command is executed, part 1 plays through the Apple speaker, part 2 through your amplifier. Another SWAP will restore the original speakers.

CHANGE allows you to change the tempo and key, and reassemble the note tables.

EDIT lists the DATA statements and ends the program, allowing you to modify the song.

SAVE requests a song title, then saves the title and the note tables on tape. Since the program uses the GET command to input the title, any characters may be input, including colons, commas, and quotes. A carriage return terminates the input and causes recording instructions to be displayed.

The Playback Program

After I wrote the program just described (the first version of which did not include the SAVE command), it occurred to me that you could spend a lot of time inputting a masterpiece, and lose it all when the computer was turned off. Of course, it's always possible to save the entire program, and thus preserve the DATA statements, but this can run into a lot of tape if you make a habit of it. Another drawback of this method is that every time the program is reloaded, the note tables have to be re-assembled, a process which can take several minutes for long songs. With all this in mind, I added the SAVE feature to the note-table assembler program, and wrote another program whose sole purpose was to load and play previously-recorded songs. Since this playback program loads note tables which are already assembled, we do not experience the delay associated with assembling, and of course a lot of time and tape is saved for anyone who wants to build up a library of songs.

Running the Program

As can be seen from the listing, line 10 of this program is the same as line 10 of the note-table assembler program. If necessary, modify this line as previously described before running the program.

After typing "RUN", you will be given brief instructions for loading a song from tape. After the song is loaded, its title will appear on the screen, and you will be prompted with a question mark. In response to the question mark, any of the following commands can be typed:

GO plays the song. Same as the GO command described earlier.

SWAP switches the speakers. Same as the SWAP command described earlier.

COPY allows you to copy the note tables to another tape. Similar to the SAVE command of the other program, but does not request a new song title.

LOAD allows you to load and play another song from tape.

It should be noted that there are no CHANGE or EDIT commands here; this is a "read-only" type program. When running the first program, then, you should be sure the tempo and key are adjusted to their most pleasing values before SAVEing the song.

```
0  REM  PLAYBACK PROGRAM
1  REM
2  REM
10 M = 32768: REM  M = SYSTEM'S CAPACITY
15  REM  LOAD MACHINE LANGUAGE PROGRAM
20 P$ = "1730050031330061730060031330071730070031330081730080031 33009169
    0014100000303209600303213200320224000723417301701707605600317304819
    1740010031362400072341730170 17"
30  FOR I = 777 TO 830: POKE I, VAL ( MID$ (P$,3 * (I - 777) + 1,3)): NE


40 P$ = "076072003173032192172003003206000003208219206002003208003032096
    032060040032082060321320030760400031620001650062080021980071980061 6
    006141001003165006208002198"
50  FOR I = 831 TO 883: POKE I, VAL ( MID$ (P$,3 * (I - 831) + 1,3)): NE


60 P$ = "00719800616100614100200320800210410417400100309616000016500820 8
    0219800919800817700814100300316500820800219800919800817700814100400
    20800210410417200300 3096"
70  FOR I = 884 TO 935: POKE I, VAL ( MID$ (P$,3 * (I - 884) + 1,3)): NE


80  DEF  FN HI(X) =  INT (X / 256)
90  DEF  FN LO(X) = X -  FN HI(X) * 256
100  HIMEM: M:B1 = M - 4
110  HOME : PRINT
120  PRINT "AFTER ADJUSTING VOLUME, PRESS 'PLAY',"
130  PRINT "THEN HIT ANY KEY.": GET P$
140  SHLOAD : REM  LOAD NOTE TABLES
150 B2 = B1 - ( PEEK (M - 1) + 256 *  PEEK (M - 2))
170 T = B1 - ) PEEK (M - 3) + 256 *  PEEK (M - 4))
180  HOME : PRINT : PRINT "TITLE:": PRINT
190  FOR I = T TO 0 STEP  - 1
200  PRINT  CHR$ ( PEEK (I));: IF  PEEK (I) = 13 THEN 215
210  NEXT
215 ET = I
217  REM  LOAD BEGINNING ADDRESSES OF NOTE TABLES
220  POKE 773, FN LO(B1): POKE 774, FN HI(B1)
230  POKE 775, FN LO(B2): POKE 776, FN HI(B2)
240  PRINT : INPUT COM$
250  IF COM$ <  > "GO" THEN 280
260  INPUT "REPETITIONS? ";R
270  FOR I = 1 TO R: CALL 777: NEXT I: GOTO 240
280  IF COM$ = "LOAD" THEN 100
290  IF COM$ <  > "SWAP" THEN 330
300  POKE 819,80 -  PEEK (819): POKE 835,80 -  PEEK (835)
310  GOTO 240
330  IF COM$ <  > "COPY" THEN  PRINT "WHAT?": GOTO 240
340  POKE 6, FN LO(M - 1 - ET): POKE 7, FN HI(M - 1 - ET)
350  POKE 60,6: POKE 61,0: POKE 62,7: POKE 63,0
360  HOME : PRINT : PRINT "AFTER ADJUSTING VOLUME, PRESS 'RECORD',"
370  PRINT "THEN HIT ANY KEY.": GET A$
380  HOME : FLASH : VTAB 12: HTAB 12: PRINT "<<RECORDING>>": NORMAL
390  CALL  - 307: REM  WRITE-TO-CASSETTE ROUTINE
400  POKE 60, FN LO(ET): POKE 61, FN HI(ET)
410  POKE 62, FN LO(M - 1): POKE 63, FN HI(M - 1)
420  CALL  - 307: HOME : GOTO 240
```

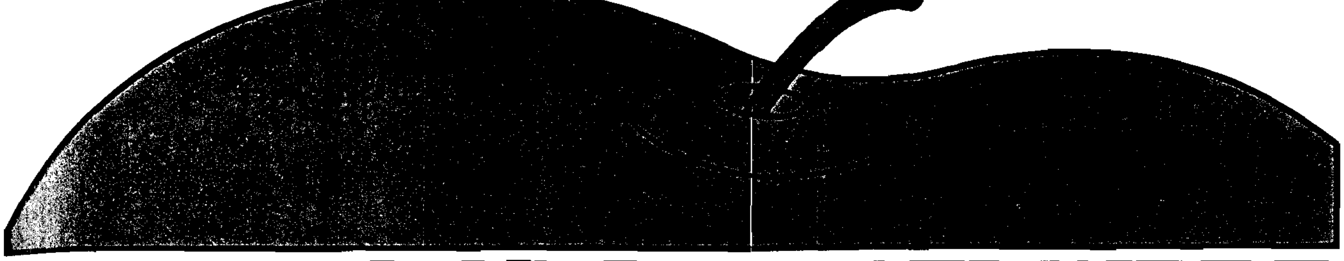
A Sample Song

In figure 2, the DATA statements for a short song are given. This is a folk song entitled "Blue Bells of Scotland." The recommended tempo and key for this song are 30, 20. These DATA statements illustrate several techniques which come in handy when you're inputting a song:

1. Input one measure per DATA statement. This way, if you get a warning that the two parts are not of the same length, you can simply check each DATA statement until you fi the measure that doesn't "add up This technique also helps you to rela the DATA statements to the she music.

2. Choose note durations whi will take the least amount of typing. this example, quarter notes a represented by 1, and eighth notes .5. If a song contains a preponderance eighth notes, on the other hand, might be wiser to represent eigh notes by 1, and quarter notes by 2, et so that you would not have to type

(Continued on page



(Continued from page 22)

so many decimal points. This would simply require a corresponding adjustment in the TEMPO when the program is run.

3. Number the DATA statements so that a measure in the melody can be easily related to the corresponding measure in the harmony. In the example, DATA statements of corresponding measures have line numbers separated by 100.

The Applesoft programs described provide a convenient method for transferring a song from sheet music to the computer. However, the assembly language routine can be used independently, as long as note tables are created, and the pointers to the beginnings of the note tables are initialized. Thus it is possible to experiment with more exotic kinds of music, using all 256 wavelengths instead of just the 65 to which my note-table assembler is limited. CALL 777 will start the song playing. If the song is interrupted (as with a RESET), CALL 840 will cause it to pick up where it left off.

When you create the note tables "by hand", (without the aid of the note-table assembler program), follow the structure illustrated in figure 3, POKEing the first note into the *highest* memory location, and working your way down. The first pointer (decimal locations 773,774) should be set to the location of the first pitch of the first part, *plus one.* Similarly, the second pointer (decimal locations 775,776) should be set to the location of the first pitch of the *second* part, plus one. In the case of solos, the first part *is* the second part, so both pointers are set to the same location. By judicious placement of these pointers, you can play duets, play solos, create a short delay between the two speakers for an "echo" effect, or even "listen" computer's ROM. For another in ing effect, execute the foll instruction:

```
POKE 835,80 - PEEK(835)
```

Then, when you do a CALL 777, parts of the song will be sent th the same speaker. This will provi excellent demonstration of why I to use two speakers instead of on

Whether you use the ma language routine independentl with the programs described in th ticle, or within your own BASIC grams, there is plenty of room fo perimentation, and I will be anxic hear about any enhancements or gestions from readers. In any ca think you will agree that two voic at least twice as good as one.

AI

MICRO

New Publications

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

General 6809

Using Microprocessors and Microcomputers: The 6800 Family by Joseph D. Greenfield and William C. Wray. John Wiley & Sons, 605 Third Avenue, New York, New York 10158, 1981, xiv, 460 pages, 7¾ × 9½ inches, hardbound.
ISBN: 0-471-02727-8 $22.95

This textbook for electronic technology and engineering students explains the uses and operation of the 6800 family of microcomputer components. Although only a few pages are devoted specifically to the 6809, the authors' comments are noteworthy: "The newer more powerful microprocessors, like the 6809, seem to be destined to replace the 6800 in new designs in the coming years.... A thorough introduction to the most promising of these microprocessors, the 6809, is presented so that the student may understand its advantages and incorporate it in new designs."

General 6502

Beyond Games: System Software for Your 6502 Personal Computer by Ken Skier. BYTE/McGraw-Hill, Book Division (70 Main Street, Peterborough, New Hampshire 03458), 1981, iv, 434 pages, diagrams and listings, 7½ × 9 3/16 inches, paperbound.
ISBN: 0-07-057860-5 $14.95

This book introduces newcomers to assembly-language programming in general, and of the 6502 in particular, and presents software tools for use in developing assembly-language programs for the 6502. The book's software runs on an Apple II, an Atari 400 or 800, an Ohio Scientific (OSI) Challenger 1-P, or a PET 2001. The author claims that with proper initialization of the System Data Block, the software should run on *any* 6502-based computer equipped with a keyboard and a memory-mapped, character-graphics video display.

CONTENTS: Introduction; Your Computer; Introduction to Assembler; Loops and Subroutines; Arithmetic and Logic; Screen Utilities; The Visible Monitor; Print Utilities; Two Hexdump Tools; A Table-Driven Disassembler; A General MOVE Utility; A Simple Text Editor; Extending the Visible Monitor; Entering the Software Into Your System. *Appendices: A.* Hexadecimal Conversion Table; ASCII Character Codes; 6502 Instruction Set — Mnemonic List; 6502 Instruction Set — Opcode List; Instruction Execution Times; 6502 Opcodes by Mnemonic and Addressing Mode. *B.* The Ohio Scientific Challenger 1-P; The PET 2001; The Apple II; The Atari 800. *C.* Screen Utilities; Visible Monitor (Top Level and Display Subroutines); Visible Monitor (Update Subroutine); Print Utilities, Two Hexdump Tools, Table-Driven Disassembler (Top Level and Utility Subroutines); Table-Driven Disassembler (Addressing Mode Subroutines); Table-Driven Disassembler (Tables); Move Utilities; Simple Text Editor (Top Level and Display Subroutines); Simple Text Editor (EDITIT Subroutines); Extending the Visible Monitor; System Data Block for the Ohio Scientific C-1P; System Data Block for the PET 2001; System Data Block for the Apple II; System Data Block for the Atari 800. *D.* Screen Utilities; Visible Monitor (Top Level and Display Subroutines); Visible Monitor (Update Subroutine); Print Utilities; Two Hexdump Tools, Table-Driven Disassembler (Top Level and Utility Subroutines); Table-Driven Disassembler (Addressing Mode Subroutines); Table-Driven Disassembler (Tables); Move Utilities; Simple Text Editor, Extending the Visible Monitor. *E.* Screen Utilities; Visible Monitor (Top Level and Display Subroutines); Visible Monitor (Update Subroutines); Print Utilities, Two Hexdump Tools; Table-Driven Disassembler (Top Level and Utility Subroutines); Table-Driven Disassembler (Addressing Mode Subroutine); Table-Driven Disassembler (Tables); Move Utilities; Simple Text Editor; Extending the Visible Monitor; System Data Block for the Ohio Scientific C-1P; System Data Block for the PET 2001; System Data Block for the Apple II; System Data Block for the Atari 800. *Index.*

Micro Chart: 6502 (65XX), Microprocessor Instant Reference Card by James D. Lewis (Micro Logic Corp., P.O. Box 174, Hackensack, New Jersey 07602), 1980: one 8½-×-11-inch plastic card, 2-color, 2-sided, 4-hole punched.
$5.95
(includes $1.00 for shipping)

This sturdy, plastic sheet for programmers, engineers, and students clearly and concisely lists significant and frequently referenced 6502 data.

CONTENTS: *Side I*—Hex to Instruction Conversion; Memory Map; Effect on Flags; Status Flags; Interrupts; Addressing Modes; ASCII Character Set; Hex and Decimal Conversion; 6502 Pins; Registers; Unsigned Comparisons; Abbreviations; Miscellaneous. *Side II*—Instruction Set; Instructions Notes; Shift Instructions; Added Cycle Time; Assembler Symbols.

Apple

MICRO/Apple, Volume 1, edited by Ford Cavallari. MICRO/Apple Series (ISSN: 0275-3537). Micro Ink, Inc. (34 Chelmsford Street, P.O. Box 6502, Chelmsford, Massachusetts 01824), 1981, 224 pages, listings and diagrams, 6 × 9 inches, cardstock cover with Wire-o binding. The inside back cover has a pocket containing a floppy disk.
ISBN: 0-938222-05-8 $24.95
(Including floppy disk)

This first volume of a new series on the Apple Computer contains 30 articles selected from *MICRO, The 6502 Journal,* 1977-1980, updated by the authors or MICRO's staff. Introductory material has been added and the 38 programs provided have been re-entered, listed, tested, and put on diskette (13-sector DOS 3.2 format, convertible to DOS 3.3).

CONTENTS: Introduction. BASIC Aids (4 articles); I/O Enhancements (4 articles); Runtime Utilities (4); Graphics (5); Education (4); Games (4); Reference (5). Language Index; Author Index (with biographies) Disk Information.

General Microcomputer

IEEE Micro is a new quarterly which began publication in February 1981. It is published by the IEEE Computer Society (10662 Los Vaqueros Circle, Los Alamitos, California 90720). It covers microcomputer design and applications and is edited for the practicing hardware and software engineer employed in design and application in areas such as communication; process control; consumer electronics; medicine; energy management; data acquisition; transportation; test, measurement, and instrumentation; navigation and guidance; military electronics; small business; microprocessor design and standardization; and education. An annual subscription to *IEEE Micro* is $8.00 in addition to society member dues ($14.00) or $23.00 for non-members.

(Continued on page 39)

A C1P Dump Utility

This article describes a debugging tool for machine language and BASIC programs.

Francois Faguy
P.O. Box 86
L'Islet-sur-mer
Quebec, Canada GOR 2B0

You have your C1P, have tried a few simple BASIC programs and want to get into more serious usage. You read magazines like MICRO and see all those great programs for Microsoft BASIC, as implemented for the Apple, PET or TRS-80 computers. They should run on your C1P since they use the same BASIC, but as soon as programs make use of machine-dependent features or BASIC flags and pointers, they don't work. The reasons are:

1. Although all these computers (and many more) use the same BASIC interpreter, they don't use the same version and release.

2. Microsoft 8K BASIC is only a BASIC interpreter. The I/O support routines are the responsibility of the system manufacturer.

3. Manufacturers add extensions to Microsoft BASIC.

4. All these systems include some kind of a monitor program; but they are all very different.

I wanted to use the technique discussed in Virginia Lee Brady's article (MICRO 27:7) for a program I am writing. I used the monitor to dump some of the page zero locations discussed and found that they did not match. So I tried dumping contiguous locations with the monitor. I wanted to check if the difference was due to a re-organization of work areas in page zero between OSI Microsoft BASIC Version 1.0, revision 3.2 and the Applesoft Version of Microsoft BASIC. But it could take years to find what I was looking for, dumping one byte at a time, and using the monitor. So I wrote the Dump program discussed in this article to get a better picture of the problem.

The Dump program is designed to be loaded at the high-end of RAM, where it can stay as long as the machine is powered-up, and as long as you enter the right memory size when you cold start. It uses 359 bytes (167 hex). On my 8K system, I set the start address to $1E00. If you wish to use Dump on a larger system, change the address in line 50 (listing 3) to the desired origin value and re-assemble the program.

Listing 1

```
10   REM   THIS PROGRAM COPIES
20   REM   THE LOADER FROM
30   REM   THE OSI ASM/EDIT TAPE
100   DIM A$(1000)
200   INPUT "READY INPUT";A$
205   REM   SET LOAD MODE
210   POKE 515,255
220   FOR I = 0 TO 239
230   INPUT A$(I)
240   NEXT I
245   REM   CLEAR LOAD MODE
250   POKE 515,0
260   INPUT "READY OUTPUT";A$
265   REM   SET SAVE MODE
270   POKE 517,255
280   FOR I = 0 TO 239
290   PRINT A$(I); CHR$ (13);
300   NEXT I
305   REM   CLEAR SAVE MODE
310   POKE 517,0
```

Listing 2

```
10   REM   THIS PROGRAM WRITES
20   REM   THE START ADDRESS
30   REM   OF A MACHINE LANGUAGE
40   REM   PROGRAM AT THE END OF
50   REM   A SELF-LOADING/AUTO-START
60   REM   OBJECT TAPE
80   INPUT "ENTER START ADDR";A$
90 A$ = "$" + A$
100   INPUT "READY OUTPUT";A$
110   REM   SET SAVE MODE
120   POKE 517,255
130   PRINT A$
140   REM   CLEAR SAVE MODE
150   POKE 517,0
```

Installation Procedure

Dump is too big to be POKEd with a BASIC program. It is preferable to use an object tape. The OSI Assembler/Editor will generate an object tape, but you need a loader. OSI does not tell you, but they give you a loader; you can use the Assembler/Editor check-sum loader to load your object tape. Listing 1 is a BASIC program that will copy the loader from OSI Assembler/Editor tape (the input tape) to your object tape (the output tape).

Once the loader is on the object tape, load the Assembler/Editor and input the Dump program (listing 3). Note that comment lines in listing 3 do not have line numbers. This is because the source file of the 8K version is too small to hold the Dump program with the comments. So do not input any comments if your machine has only 8K.

Next, assemble the program with "A1" to ensure that there are no errors. Then save the source listing as this can be useful if you wish to customize Dump later. While still in save mode, put the object tape in the cassette recorder, wind it past the end of the loader, and type "A2", ready the recorder for writing and hit RETURN. This will write the object program on the tape.

If you wish a self-starting tape, the BASIC program in listing 2 will write the start address in the format required by the loader at the end of the object file on the tape. For the 8K version, reply 1E00 to "ENTER START ADDRESS". If you do not write a start address on the object tape, use the BREAK key to exit from the loader. Typing M1E00G will run Dump.

Using Dump

To load the program, hit BREAK, type "ML", put the object tape in the recorder, and start the recorder. Once the program is loaded, it will self-start. The screen is first cleared and three prompts are displayed at the bottom of the screen. You can:

1. Enter the 4-digit hexadecimal address of where the dump is to start and 64 bytes will be displayed (see figure 1).

2. Hit RETURN to dump the next higher 64 bytes. If RETURN is used the first time round, the dump will start at $0000.

3. Enter "R", to cause Dump to execute a RTS instruction.

Listing 3

```
0800                ;*******************************
0800                ;*                             *
0800                ;* OSI C1P MEMORY DUMP PROGRAM *
0800                ;*                             *
0800                ;*       BY FRANCOIS FAGUY     *
0800                ;*                             *
0800                ;*******************************
0800                ;*
0800                ;DUMPS 64 BYTES OF MEMORY ON THE SCREEN
0800                ; IN BOTH HEX AND ASCII
0800                ;
0800                ;CAN BE RUN FROM THE MONITOR: M 1E00 G
0800                ; OR AS A USR(X) FUNCTION FROM BASIC
0800                ;
0800                DLOC   EPZ $14              ;CURRENT DISPLAY LOCATION
0800                DADDR  EPZ $16              ;POINTER TO CURRENT ADDRESS
0800                ;
0800                BASIN  EQU $FFEB            ;BASIC KEYBOARD INPUT ROUTINE
0800                DSPLY  EQU 53510            ;FIRST BYTE USED IN VIDEO RAM
0800                ;
1E00                       ORG $1E00
1E00                       OBJ $800
1E00                ;
1E00                ;CLEAR THE SCREEN
1E00                ;
1E00 A200           DUMP   LDX #$00             ;INIT X REG.
1E02 A920                  LDA #$20             ;SPACE
1E04 9D00D3         CLEAR  STA $D300,X          ;FILL VIDEO RAM WITH SPACES
1E07 9D00D2                STA $D200,X
1E0A 9D00D1                STA $D100,X
1E0D 9D00D0                STA $D000,X
1E10 E8                    INX
1E11 D0F1                  BNE CLEAR
1E13                ;
1E13                ;DISPLAY PROMPT MESSAGES
1E13                ;
1E13 A011                  LDY #17
1E15 B9331F         MSG10  LDA MSG1,Y
1E18 9926D3                STA DSPLY+544,Y
1E1B 88                    DEY
1E1C 10F7                  BPL MSG10
1E1E A011                  LDY #17
1E20 B9451F         MSG20  LDA MSG2,Y
1E23 9946D3                STA DSPLY+576,Y
1E26 88                    DEY
1E27 10F7                  BPL MSG20
1E29 A010                  LDY #16
1E2B B9571F         MSG30  LDA MSG3,Y
1E2E 9966D3                STA DSPLY+608,Y
1E31 88                    DEY
1E32 10F7                  BPL MSG30
1E34                ;
1E34                ;GET THE START ADDRESSS FROM THE KEYBOARD
1E34                ;
1E34 A2FC           GET    LDX #252             ;INIT REG. X FOR 4 CHAR.
1E36 20EBFF         GET05  JSR BASIN            ;READ A CHARACTER
1E39 C90D                  CMP #$0D             ;<CR> ?
1E3B F046                  BEQ DUMP05           ;YES, DUMP NEXT 64 BYTES
1E3D C952                  CMP #'R'
1E3F D001                  BNE GET08            ;NO, CARRY ON
1E41 60                    RTS
1E42 9DCACF         GET08  STA DSPLY-316,X      ;DISPLAY THE CHARACTER
1E45 C930                  CMP #'0'
1E47 30ED                  BMI GET05            ;<0 = ERROR
1E49 C93A                  CMP #'9'+1
1E4B 300A                  BMI GET10            ;NO = GOOD CHARACTER
1E4D C941                  CMP #'A'
1E4F 30E5                  BMI GET05            ;<'A' = ERROR
1E51 C947                  CMP #'F'+1
1E53 B0E1                  BCS GET05            ;>'F' = ERROR
1E55 E906                  SBC #$06             ;('A'-'9'-2)--CONVERT HEX DIGITS
1E57 290F           GET10  AND #$0F             ;CONVERT HEX DIGITS 0-F
1E59 9D2B1E                STA ADDRIN-252,X     ;SAVE HEX DIGIT
1E5C E8                    INX                  ;CHECK FOR 4 CHARACTERS
1E5D D0D7                  BNE GET05            ;NEXT CHARACTER
1E5F                ;
1E5F                ;PACK ADDRESS IN TWO BYTES
1E5F                ;
1E5F AD291F                LDA ADDRIN+2         ;THIRD HEX DIGIT
1E62 0A                    ASL
1E63 0A                    ASL
1E64 0A                    ASL
1E65 0A                    ASL
1E66 0D2A1F                ORA ADDRIN+3         ;FOURTH HEX DIGIT
1E69 8D2C1F                STA ADDR+1           ;SAVE LOW BYTE OF ADDRESS
1E6C AD271F                LDA ADDRIN           ;FIRST HEX DIGIT
1E6F 0A                    ASL                  ;SHIFT TO 4 HIGH BITS OF ACC.
1E70 0A                    ASL
1E71 0A                    ASL
1E72 0A                    ASL
1E73 0D281F                ORA ADDRIN+1         ;SECOND HEX DIGIT
1E76 8D2B1F                STA ADDR             ;SAVE HIGH BYTE OF ADDRESS
1E79                ;
1E79                ;ERASE INPUT AREA
1E79                ;
1E79 A203                  LDX #$03
1E7B A920                  LDA #$20             ;SPACE
1E7D 9DC6D0         GET15  STA DSPLY-64,X
1E80 CA                    DEX
1E81 10FA                  BPL GET15
```

(continued

Listing 3

```
1E83                       ;
1E83                       ;NOW THAT WE HAVE THE START ADDRESS,
1E83                       ; START DUMPING
1E83                       ;
1E83 18                    DUMP05 CLC
1E84 AD2C1F                       LDA ADDR+1           ;SAVE START ADDRESS + 64
1E87 6940                         ADC #64
1E89 8D2E1F                       STA SADDR+1
1E8C AD2B1F                       LDA ADDR
1E8F 6900                         ADC #$00             ;ADD CARRY TO HIGH BYTE
1E91 8D2D1F                       STA SADDR
1E94 AD2F1F                       LDA SLOC             ;SET STARTING VIDEO RAM ADDR.
1E97 8514                         STA DLOC
1E99 AD301F                       LDA SLOC+1
1E9C 8515                         STA DLOC+1
1E9E                       ;
1E9E                       ;DISPLAY ADDRESS OF FIRST BYTE OF THIS LINE
1E9E                       ;
1E9E AD311F                DUMP10 LDA ADDRP            ;SETUP ADDR. FOR HEXASC
1EA1 8516                         STA DADDR
1EA3 AD321F                       LDA ADDRP+1
1EA6 8517                         STA DADDR+1
1EA8 A001                         LDY #$01             ;INIT REG. Y FOR 2 BYTES
1EAA 20F21E                       JSR HEXASC           ;DISPLAY ADDRESS
1EAD                       ;
1EAD                       ;DISPLAY NEXT 4 BYTES IN HEX
1EAD                       ;
1EAD 18                           CLC
1EAE AD2C1F                       LDA ADDR+1           ;SETUP ADDR. FOR HEXASC.
1EB1 8516                         STA DADDR
1EB3 6904                         ADC #$04             ;AND ADD 4 TO ADDRESS
1EB5 8D2C1F                       STA ADDR+1
1EB8 AD2B1F                       LDA ADDR
1EBB 8517                         STA DADDR+1
1EBD 6900                         ADC #$00             ;ADD CARRY TO HIGH BYTE
1EBF 8D2B1F                       STA ADDR
1EC2 A905                         LDA #$05             ;ADD 5 TO VIDEO RAM POINTER
1EC4 201B1F                       JSR INCLOC
1EC7 A003                         LDY #$03             ;INIT REG. Y FOR 4 BYTES
1EC9 20F21E                       JSR HEXASC           ;DISPLAY 4 BYTES
1ECC                       ;
1ECC                       ;DISPLAY SAME 4 BYTES IN ASCII
1ECC                       ;
1ECC A909                         LDA #$09             ;ADD 9 TO VIDEO RAM POINTER
1ECE 201B1F                       JSR INCLOC
1ED1 A003                         LDY #$03             ;INIT REG. Y FOR 4 BYTES
1ED3 B116                  DUMP15 LDA (DADDR),Y        ;GET BYTE
1ED5 9114                         STA (DLOC),Y         ;DISPLAY IT
1ED7 88                           DEY                  ;MORE BYTES?
1ED8 10F9                         BPL DUMP15           ;YES, DISPLAY THEM
1EDA A912                         LDA #18              ;ADD 18 TO VIDEO RAM POINTER
1EDC 201B1F                       JSR INCLOC
1EDF                       ;
1EDF                       ;CHECK IF WE ARE FINISHED
1EDF                       ;
1EDF AD2E1F                       LDA SADDR+1          ;LOW BYTE EQUAL?
1EE2 CD2C1F                       CMP ADDR+1
1EE5 D0B7                         BNE DUMP10           ;NO, NEXT LINE
1EE7 AD2D1F                       LDA SADDR
1EEA CD2B1F                       CMP ADDR             ;HIGH BYTE EQUAL?
1EED D0AF                         BNE DUMP10           ;GET NEXT START ADDRESS
1EEF 4C341E                       JMP GET              ;GET NEXT START ADDRESS
1EF2                       ;
1EF2                       ;THIS SUBROUTINE CONVERTS FROM 2 HEX DIGITS
1EF2                       ; PER BYTE TO 2 ASCII CHARACTERS IN 2 BYTES
1EF2                       ;
1EF2                       ;DADDR: POINTS TO THE FIRST INPUT BYTE
1EF2                       ;DLOC : POINTS TO OUTPUT AREA
1EF2                       ;Y REG: NUMBER OF BYTES MINUS 1 TO CONVERT
1EF2                       ;
1EF2 B116                  HEXASC LDA (DADDR),Y        ;GET BYTE
1EF4 AA                           TAX                  ;SAVE IT IN REG. X
1EF5 98                           TYA
1EF6 0A                           ASL                  ;MULTIPLY REG. Y BY 2
1EF7 A8                           TAY
1EF8 8A                           TXA                  ;PUT BYTE BACK IN REG. A
1EF9 4A                           LSR                  ;EXTRACT FIRST DIGIT
1EFA 4A                           LSR
1EFB 4A                           LSR
1EFC 4A                           LSR
1EFD 20121F                       JSR HEXA10           ;MAKE IT A CHARACTER
1F00 9114                         STA (DLOC),Y         ;DISPLAY IT
1F02 8A                           TXA                  ;PUT BYTE BACK IN REG. A
1F03 290F                         AND #$0F             ;EXTRACT SECOND DIGIT
1F05 C8                           INY                  ;NEXT OUTPUT BYTE
1F06 20121F                       JSR HEXA10           ;MAKE IT A CHARACTER
1F09 9114                         STA (DLOC),Y         ;DISPLAY IT
1F0B 98                           TYA
1F0C 4A                           LSR                  ;DIVIDE REG. Y BY 2
1F0D A8                           TAY
1F0E 88                           DEY                  ;MORE BYTES?
1F0F 10E1                         BPL HEXASC           ;YES, CONVERT THEM
1F11 60                           RTS                  ;NO, RETURN
1F12                       ;
1F12                       ;CONVERT UNPACKED HEX DIGIT IN REG. A
1F12                       ; TO ASCII CHARACTER IN REG. A
1F12                       ;
1F12 C90A                  HEXA10 CMP #10              ;LESS THAN 10?
1F14 9002                         BCC HEXA15
```

(continued)

The last option can be useful for debugging: Dump can be called from an assembler program using JSR $1E00 or from BASIC using the USR(X) function. You can dump part of memory and then continue your program execution where it left off.

To use Dump with BASIC, hit BREAK when the program is loaded, then type "C" to cold start and reply 7680 to "MEMORY SIZE".

Program Logic

(All line numbers refer to listing 3)

Lines 10 to 40 are equates for the following symbols:

BASIN: the BASIC input routine, used by Dump for all keyboard input.

DSPLY: the start of the first line of dump in the video RAM. This value can be adjusted if your TV monitor has a different overscan from mine.

DLOC and DADDR: two page-zero words used as pointers with indirect-postindexed addressing. Locations $14-$17 are part of a BASIC input buffer and using them does not seem to have any adverse effect.

Lines 60-150 clear the screen.

Lines 160-330 display the prompts.

Lines 340-780 read the keyboard and execute a RTS if "R" is entered, or branch to DUMP05 if you hit RETURN, or translate the 4 hexadecimal digits to an address.

At lines 790-900 at label DUMP05, the start address plus 64 is saved in SADDR. SADDR will be used later to decide when the display is full. The page-zero pointer (DLOC) to video RAM is set to the DSPLY value.

Lines 910-970 display the address of the first byte of the current line.

Lines 980-1100 display the hexadecimal value of the next four bytes.

Lines 1110-1200 display the same 4 bytes in ASCII.

Lines 1210-1270 check for the end of the 64 bytes.

Lines 1280-1580 are the subroutine HEXASC. It is used to display addresses and the hexadecimal dump. Refer to listing 3 for more details.

Lines 1590-1660 are the subroutine INCLOC. It is used to update the current video RAM position pointer (DLOC).

Francois Faguy has 10 years of programming experience. Starting as an application programmer, he moved to operating system support and data base administration. His hardware experience includes the DEC PDP 11 line and almost all systems marketed by IBM in the last 15 years, from the 1130 to the 3033. After working for large Canadian corporations, he is now a freelance consultant.

MICRO

Listing 3 *(continued)*

```
1F16 6906            ADC #$06              ;NO, ADD OFFSET FOR A-F
1F18 6930     HEXA15 ADC #'0'              ;ADD OFFSET FOR ASCII
1F1A 60              RTS
1F1B          ;
1F1B          ;THIS SUBROUTINE ADDS REG. A TO DLOC
1F1B          ;
1F1B 18       INCLOC CLC
1F1C 6514            ADC DLOC              ;ADD TO LOW BYTE
1F1E 8514            STA DLOC              ;SAVE LOW BYTE
1F20 A515            LDA DLOC+1            ;GET HIGH BYTE
1F22 6900            ADC #$00              ;ADD CARRY
1F24 8515            STA DLOC+1            ;SAVE HIGH BYTE
1F26 60              RTS
1F27          ;
1F27          ;WORK AREAS
1F27          ;
1F27 000000   ADDRIN HEX 00000000          ;SAVE 4 HEX DIGITS OF START ADDI
1F2A 00
1F2B 0000     ADDR   HEX 0000              ;POINTER TO NEXT BYTE TO DUMP
1F2D 0000     SADDR  HEX 0000              ;START ADDRESS + 64
1F2F 06D1     SLOC   ADR DSPLY             ;STARTING VIDEO RAM LOCATION
1F31 2B1F     ADDRP  ADR ADDR              ;POINTER TO ADDR FOR HEXASC
1F33 3C4352   MSG1   ASC '<CR>:NEXT 64 BYTES'
1F36 3E3A4E
1F39 455854
1F3C 203634
1F3F 204259
1F42 544553
1F45 523A52   MSG2   ASC 'R:RETURN TO CALLER'
1F48 455455
1F4B 524E20
1F4E 544F20
1F51 43414C
1F54 4C4552
1F57 342044   MSG3   ASC '4 DIGITS HEX ADDR'
1F5A 494749
1F5D 545320
1F60 484558
1F63 204144
1F66 4452
```

***Figure 1:* The information displayed by the DUMP Utility Program. The first four characters of each line represent the address in hex of the first byte displayed on the line. The next eight characters, are the hex content of four bytes. The last four characters are the ASCII or graphic value of the same four bytes.**

HEX	DEC	+0					+5					+10					+15					+20					DEC	HEX
D083	53379																										53403	D09B
D0A3	53411																										53435	D0BB
D0C3	53443				1	F	2	0																			53467	D0DB
D0E3	53475																										53499	D0FB
D103	53507				1	F	2	0		A	5	1	5	6	9	0	0		[graphic]	[graphic]	i	[graphic]					53531	D11B
D123	53539				1	F	2	4		8	5	1	5	6	0	0	1		[graphic]	[graphic]		[graphic]					53563	D13B
D143	53571				1	F	2	8		0	F	0	2	0	0	1	F		[graphic]	[graphic]	[graphic]	[graphic]					53595	D15B
D163	53603				1	F	2	C		3	0	1	F	6	0	0	6		0	[graphic]		[graphic]					53627	D17B
D183	53635				1	F	3	0		D	1	2	B	1	F	3	C		[graphic]	+	[graphic]	<					53659	D19B
D1A3	53667				1	F	3	4		4	3	5	2	3	E	3	A		C	R		:					53691	D1BB
D1C3	53699				1	F	3	8		4	E	4	5	5	8	5	4		N	E	X	T					53723	D1DB
D1E3	53731				1	F	3	C		2	0	3	6	3	4	2	0			6	4						53755	D1FB
D203	53763				1	F	4	0		4	2	5	9	5	4	4	5		B	Y	T	E					53787	D21B
D223	53795				1	F	4	4		5	3	5	2	3	A	5	2		S	R	:	R					53819	D23B
D243	53827				1	F	4	8		4	5	5	4	5	5	5	2		E	T	U	R					53851	D25B
D263	53859				1	F	4	C		4	E	2	0	5	4	4	F		N		T	O					53883	D27B
D283	53891				1	F	5	0		2	0	4	3	4	1	4	C			C	A	L					53915	D29B
D2A3	53923				1	F	5	4		4	C	4	5	5	2	3	4		L	E	R	4					53947	D2BB
D2C3	53955				1	F	5	8		2	0	4	4	4	9	4	7			D	I	G					53979	D2DB
D2E3	53987				1	F	5	C		4	9	5	4	5	3	2	0		I	T	S						54011	D2FB
D303	54019																										54043	D31B
D323	54051					C	R		:	N	E	X	T		6	4		B	Y	T	E	S					54075	D33B
D343	54083				R	:	R	E	T	U	R	N		T	O		C	A	L	L	E	R					54107	D35B
D363	54115				4		D	I	G	I	T	S		H	E	X		A	D	D	R						54139	D37B
D383	54147																										54171	D39B
		+0					+5					+10					+15					+20						

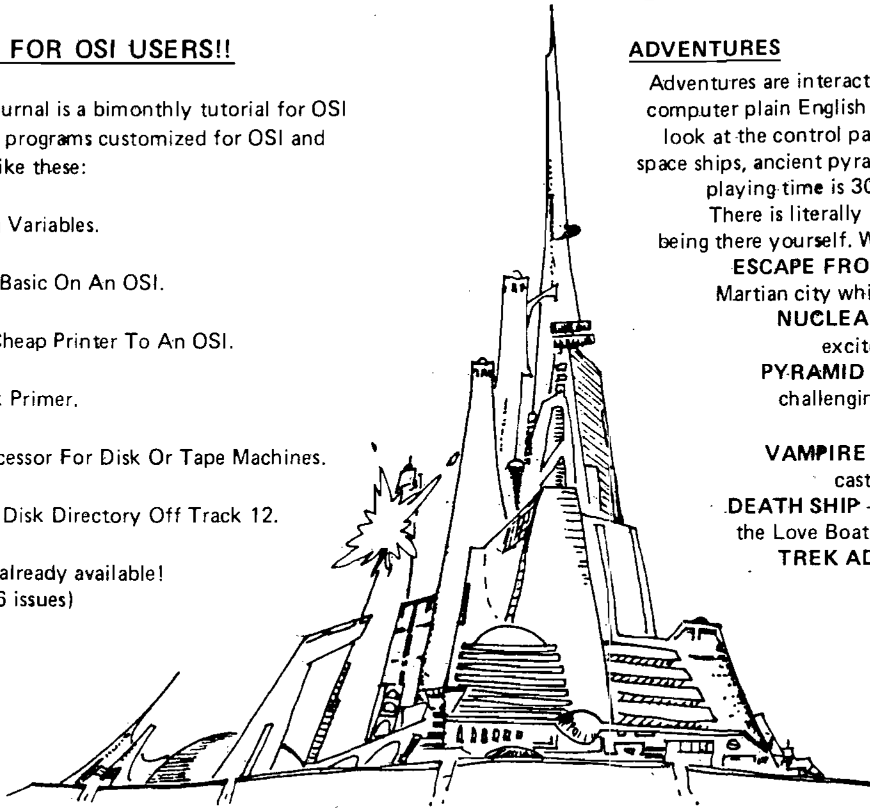
OSI SOFTWARE FOR OSI OSI

A JOURNAL FOR OSI USERS!!

The Aardvark Journal is a bimonthly tutorial for OSI users. It features programs customized for OSI and has run articles like these:

1) Using String Variables.
2) High Speed Basic On An OSI.
3) Hooking a Cheap Printer To An OSI.
4) An OSI Disk Primer.
5) A Word Processor For Disk Or Tape Machines.
6) Moving The Disk Directory Off Track 12.

First year issues already available!
$9.00 per year (6 issues)



ADVENTURES

Adventures are interactive fantasies where you give the computer plain English commands (i.e. take the sword, look at the control panel.) as you explore alien cities, space ships, ancient pyramids and sunken subs. Average playing time is 30 to 40 hours in several sessions. There is literally nothing else like them – except being there yourself. We have six adventures available.

ESCAPE FROM MARS – Explore an ancient Martian city while you prepare for your escape.

NUCLEAR SUBMARINE – Fast moving excitement at the bottom of the sea.

PYRAMID – Our most advanced and most challenging adventure. Takes place in our own special ancient pyramid.

VAMPIRE CASTLE – A day in old Drac's castle. But it's getting dark outside.

DEATH SHIP – It's a cruise ship – but it ain't the Love Boat and survival is far from certain.

TREK ADVENTURE – Takes place on a familiar starship. Almost as good as being there.

NEW SUPPORT ROMS FOR BASIC IN ROM MACHINES

C1S – for the C1P only, this ROM adds full screen edit functions (insert, delete, change characters in a basic line). Softwave selectable scroll windows, two instant screen clears (scroll window only and full screen), software chose of OSI or standard keyboard format, Bell support, 600 Baud cassette support, and a few other features. It plugs in in place of the OSI ROM. NOTE: this ROM also supports video conversions for 24, 32, 48 or 64 characters per line. Replaces video swap tape on C1P model 2. All that and it sells for a measly $39.95.

C1E/C2E for C1/C2/C4/C8 Basic in ROM machines. This ROM adds full screen editing, software selectable scroll windows, keyboard correction (software selectable), and contains an extended machine code monitor. It has breakpoint utilities, machine code load and save, block memory move and hex dump utilities. A must for the machine code programmer replaces OSI support ROM. Requires installation of additional chip when installed in a C2 or C4. C1 installation requires only a jumper move. Specify system $59.95.

DISK UTILITIES

SUPER COPY – Single Disk Copier
This copy program makes multiple copies, copies track zero, and copies all the tracks that your memory can hold at one time – up to 12 tracks at a pass. It's almost as fast as dual disk copying. – $15.95

MAXIPROSS (WORD PROCESSOR) – 65D polled keyboard only - has global and line edit, right and left margin justification, imbedded margin commands, choice of single, double or triple spacing, file access capabilities and all the features of a major word processor – and it's only $39.95.

P.C. BOARDS

MEMORY BOARDS!! – for the C1P. – and they contain parallel ports!
Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available now!

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

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

ARCADE AND VIDEO GAMES

GALAXIA one of the fastest and finest arcade games ever written for the OSI, this one features rows of evasive, hardhitting, dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. – P.S. The price is a giveaway. SPECIFY SYSTEM!
Cassette $9.95 – Disk $12.95

TIME TREK (8K) – real time Startrek action. See your torpedoes move across the screen! Real graphics – no more scrolling displays. $9.95

INTERCEPTOR C1P ONLY! An all machine code program as fast and smooth as the arcades. You use your interceptor to protect your cities from hordes of enemy invaders. A pair of automatic cannons help out, but the action speeds up with each wave of incoming ships. The fastest and most exciting C1P game yet.
C1P Cassette $19.95

MINOS – A game with amazing 3D graphics. You see a maze from the top, the screen blanks, and then you are in the maze at ground level, finding your way through on foot. Realistic enough to cause claustrophobia. – $12.95

SINGLE STEPPER / MONITOR

This is probably the finest debugging tool for machine code ever offered for OSI systems. Its' trace function allows you to single step through a machine code program while it continuously displays the A, X, Y and status registers and the program and stack pointers. You can change any of the registers or pointers or any memory location at any time under program control. It takes well under 1k and can be relocated anywhere in free memory. It is a fine tool for all systems – and the best news of all is the extremely low price we put on it. – Tape $19.95 – Disk $24.95

FOR DISK SYSTEMS – (65D, polled keyboard and standard video only.)

SUPERDISK. Contains a basic text editor with functions similar to the above programs and also contains a renumberer, variable table maker, search and new BEXEC* programs. The BEXEC* provides a directory, create, delete, and change utilities on one track and is worth having by itself. – $24.95 on 5" disk - $26.95 on 8".

AARDVARK IS NOW AN OSI DEALER!

Now you can buy from people who can support your machine.

– THIS MONTH'S SPECIALS –

Superboard II	$279
C1P Model II	429
C4P	749
8K 610 board for C1P	269
Epson MX-80 printer with RS232 installed	595

. . . and we'll include a free Text Editor Tape with each machine!

True 32X32 Video Mod Plans for C1P
(4 Chips $3.00 Crystal Required)
$7.95

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 • 1690 Bolton • Walled Lake, MI 48088**
(313) 669-3110 OSI

MICRO Classified

Each classified ad costs only $10.00 per insertion, pre-paid with typewritten copy. These ads are limited to one per company/individual per month. Please limit these entries to less than 40 words. (Oversized ads will be rejected and returned.) Title line, name and address are not considered in count. Ads received before the 20th of the second month preceding the month of publication will be published in next issue, i.e. May 20th for the July issue. For further information, call (617) 256-5515.

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

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

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 pp documentation.

Jerome B. Blumenthal, M.D.
7500 E. Hellman
Rosemead, California 91770

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

PET Machine Language Guide

Comprehensive manual to aid machine language programmer. More than 30 routines are fully detailed so that the reader can put them to immediate use. OLD or NEW ROMs. $6.95 + .75 postage. VISA & Mastercharge accepted.

Abacus Software
P.O. Box 7211
Grand Rapids, Michigan 49510

New Catalog

Includes hardware and software i
C1P/Superboard and other com
Games, simulations, utilities,
chips, boards and programs. Send $
catalog and sample program (refu
order).

Software Plus +
1818 Ridge Avenue
Florence, Alabama 35630

Ohio Scientific

SPACE WARS: for C1P or Superboa
Put your ship through S-turns, fire
lasers at the enemy. WIZARDS & W
RIORS: fight the creatures in
dungeons, search for treasure. Wher
finish one dungeon there's al
another. W & W for all OSI. SPACE W
$4.95, WIZARDS & WARRIORS $
cassette only.

Danny Havey
14430 Whittier Blvd., Suite 109
Whittier, California 90605

Accounts Receivable by SBCS

For the Apple II. This conversio
Osborne's Accounts Receivable soft
contains the same capabilities, plus m
enhancements that increase your Ap
flexibility, speed, and performance.
alone or integrate with existing Ge
Ledger program. Retail price $180.

Small Business Computer Syster
4140 Greenwood
Lincoln, Nebraska 68504

Applechess Openings

This program contains 1310 half mov
86 final positions that are reached w
less than 7 moves and no more than 1
quires 48K, Applesoft, Integer in ROM
drive, chess board and pieces. Av
only on disk (please specify 3.2 or 3.3
Send $20 in check or money order to

Bill Cowan
24329 Westwood Dr.
Westlake, Ohio 44145

Machine Language to DATA Statement Conversion

Many times machine language routines are implemented in BASIC programs as DATA statements. This article will demonstrate an easy and accurate way to incorporate the routines into your BASIC programs.

Les Cain
1319 N. 16th
Grand Junction, Colorado 81501

Anyone who has written machine code routines and then tried to convert them to DATA statements to include in a BASIC program, knows the problems encountered in converting hex to decimal, and then typing in the DATA statements. This method works but is slow and is subject to numerous errors.

While converting an Othello program from Mr. Earl Morris to work on disk BASIC, I had to change some of the machine code to work with the disk USR(X) functions, and then redo the DATA statements to POKE in the correct code. That was too much trouble, so I wrote the following short program to do the work for me.

Lines 70 through 110 prompt for the beginning and the ending addresses of the machine code. Subroutine 250 enters with a hex number and returns a decimal number. If you are just looking at the data then line numbers are not needed, and the beginning and ending addresses are printed.

To record on tape, line numbers are required. Be sure line numbers are compatible with the BASIC program. Change line 155 (cassette tape output) to suit your particular system. Change line 230 to a REM statement, then turn on recorder and run the program. Output will have line numbers and DATA statements along with the machine code in decimal format. Then all that is required is to input from cassette into your BASIC program, put in the READ and POKE statements and you're on your way.

MICRO

```
1   REM  MACHINE CODE TO DATA STATEMENT ROUTINE
2   REM   BY LES CAIN
3   REM   MICRO #36 JUNE 1981
4   REM
10   DIM D(4)
20   FOR I = 1 TO 30: PRINT : NEXT
30   PRINT  TAB( 20);"PEEKS AT MACHINE CODE "
40   PRINT  TAB( 20);"AND RETURNS DATA"
50   FOR I = 1 TO 10: PRINT : NEXT
70   INPUT "BEGIN ADDRESS";BE$:N$ = BE$
90   GOSUB 250:B = D:C = B
100   INPUT "END ADDRESS";EN$:N$ = EN$
110   GOSUB 250:E = D:F = E
120   GOSUB 330
130   PRINT : PRINT : PRINT
140   PRINT "DECIMAL";B; TAB( 20);"$";BE$
150   PRINT : PRINT : PRINT
155   REM --INSERT ROUTINE TO OUTPUT TO TAPE AT THIS LINE
170   IF F >  = C THEN  PRINT LN;: PRINT "DATA";
180 AA$ = ""
190   FOR J = B TO B + 15
200 A$ =  STR$ ( PEEK (J))
210 AB$ = ""
220   FOR I = 2 TO  LEN (A$):AB$ = AB$ +  MID$ (A$,I,1): NEXT
225 AA$ = AA$ + AB$
226 F = F - 1
227   IF J <  > B + 15 AND F > C THEN AA$ = AA$ + ","
228   IF F <  = C THEN  PRINT AA$: GOTO 230
229   NEXT : PRINT AA$:B = B + 16:LN = LN + IN: GOTO 170
230   PRINT : PRINT : PRINT "DECIMAL";E; TAB( 20);"HEX  $"EN$
231   GOTO 70
250 J = 1
260   FOR I = 1 TO 4:D(I) = 0: NEXT
270   FOR I = 1 TO 4
280 D(I) =  ASC ( MID$ (N$,J)) - 48
290   IF D(I) > 9 THEN D(I) = D(I) - 7
300 J = J + 1: NEXT
310 D = 4096 * D(1) + 256 * D(2) + 16 * D(3) + D(4)
320   RETURN
330   INPUT "BEGIN LINE NUMBER";LN
340   INPUT "INCREMENT";IN
350   RETURN
```

Telephone Directory/Dialer for the AIM

Turn your AIM into a telephone operator with a directory and dialer program.

Rodney A. Kreuter
Route 1, Box 310
Fincastle, Virginia 24090

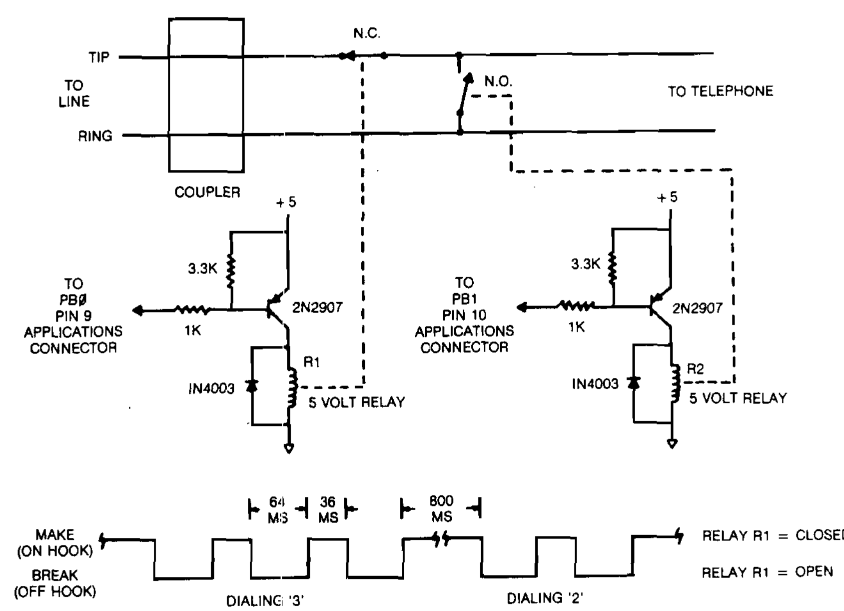
Although using a micro to dial a telephone is certainly not a new idea, I think you'll find this directory/dialer a useful program to add to your AIM 65 library. The directory/dialer can store and dial approximately 100 names and phone numbers in a 4K AIM 65. Since it is written entirely in assembly language, you will not need the BASIC or assembler ROMS. However, you will need at least 2K of RAM to hold the program and the directory.

The directory is simply the list of names and phone numbers that you wish to store. There are a few restrictions: the name can only be 16 characters long (see program modification for longer names). The name can be alpha/numeric but must not contain an '=' sign. The name must be followed by an '=' sign. The number must not contain any character that is not numeric, and each entry must end with a carriage return. For example:

Valid	DAD = 5630211[CR]
Valid	HARDWARE ON 2nd = 3894217[CR]
Invalid	MARY = (703)9458512 [CR] ( ) are not numeric
Invalid	JOE = 814-502-4907 [CR] - - are not numeric

Table 1

Location	Name	Description
$0000,0001	PNTR	This is the pointer used to store the directory in RAM.
$0002,0003	BTMPTR	The bottom or end RAM location of your directory.
$0004,0005	MSGPTR	Message pointer—points to the message string.
$0006,0007	FINPTR	Find pointer—used by string search to find the string.
$0008	LEN	Length of the string entered.
$0020 - 002F	STRING	User entered string.
$0030 - ??	NUMBER	ASCII of number to be dialed.
$0200,0201	-----	Image of PNTR.
$0202,0203	-----	Directory end address.
$0204,0205	-----	Directory start address.



Note: **A direct connection is not legal unless a coupler is used. For more information, please refer to "SYM-Bell" by Randy Sebra (30:17).**

About the Program

The directory/dialer can be divided into three basic programs:

1. *Entry program:* This allows you to assign directory storage space and does the actual storing of your data.

2. *String search program:* This program scans your directory and finds the number you wish to dial.

3. *Interface program:* This program does the actual dialing by using two relays connected to one of the user ports.

Since this program is not heavily commented (I barely had enough RAM to assemble it), some definitions will help in understanding the program. They appear in table 1.

The three pointers from $0200 -$0205 were put there so that they are saved on cassette when the program is dumped. This way the directory can always be updated. Be sure to dump from $0200 to the end of your directory.

After loading the program begin execution at $0210. *Note:* It does *not* begin at $0200.

The following is a sample run:

AIM: Dial (D) or Enter (E)?
USER: E
AIM: New (N) or Add (A)?
USER: N

Note: The first time the program is run you must respond with New in order to assign directory space. Later you will add additional numbers by replying ADD (A).

AIM: From =
USER: 0450 [CR]
AIM: To =
USER: 0600 [CR]
AIM: ∧
USER: ; (Semi-colon gets you out of the entry mode)
AIM: Dial (D) or Enter?
USER: D
AIM: Name ?
USER: Rod
AIM: Rod = 4732128
USER: (Pick up the phone and wait for dial tone. Hit any key and the AIM will begin dialing)
AIM: Redial?
USER: (Any key except 'Y' if you do not wish to redial, 'Y' if you do)

```
0800                ;*****************************************
0800                ;*                                       *
0800                ;* TELEPHONE DIRECTORY/DIALER FOR AIM 65 *
0800                ;*                                       *
0800                ;*        BY RODNEY A. KREUTER           *
0800                ;*****************************************
0800                ;*
0800                ;AIM SUBROUTINES
0800                ;
0800                BLANK2 EQU $E83B
0800                CRLOW  EQU $EA13
0800                FROM   EQU $E7A3
0800                OUTPUT EQU $E97A
0800                READ   EQU $E93C
0800                REDOUT EQU $E973
0800                TO     EQU $E7A7
0800                ;
0800                PNTR   EPZ $00              ;RAM POINTER
0800                BTMPTR EPZ $02              ;END OF RAM
0800                MSGPTR EPZ $04
0800                FINPTR EPZ $06              ;USED TO FIND STRIN
0800                LEN    EPZ $08              ;LENGTH OF STRING
0800                STRING EPZ $20
0800                NUMBER EPZ $30
0800                ;
0210                       ORG $210
0210                       OBJ $800
0210                ;
0210 A200           GO     LDX #$00
0212 207C03                JSR MSGSUB
0215 203CE9         LP0    JSR READ
0218 C945                  CMP #'E'             ;ENTER?
021A F006                  BEQ ENTER
021C C944                  CMP #'D'             ;DIAL?
021E F070                  BEQ DIAL
0220 D0F3                  BNE LP0
0222                ;
0222 A201           ENTER  LDX #$01
0224 207C03                JSR MSGSUB
0227 203CE9         LP1    JSR READ
022A C941                  CMP #'A'             ;ADD?
022C F030                  BEQ ADD
022E C94E                  CMP #'N'             ;NEW?
0230 F002                  BEQ NEW
0232 D0F3                  BNE LP1
0234                ;
0234 2013EA         NEW    JSR CRLOW
0237 20A3E7                JSR FROM
023A AD1CA4                LDA $A41C
023D 8D0002                STA $200
0240 8D0402                STA $204
0243 AD1DA4                LDA $A41D
0246 8D0102                STA $201
0249 8D0502                STA $205
024C 203BE8                JSR BLANK2
024F                ;
024F 20A7E7         MORE   JSR TO
0252 AD1CA4                LDA $A41C
0255 8D0202                STA $202
0258 AD1DA4                LDA $A41D
025B 8D0302                STA $203
025E                ;
025E                ;MOVE POINTER TO ZERO PAGE
025E                ;
025E 2013EA         ADD    JSR CRLOW
0261 A203                  LDX #$03
0263 BD0002         LP2    LDA $200,X
0266 9500                  STA $00,X
0268 CA                    DEX
0269 10F8                  BPL LP2
026B A000                  LDY #$00
026D                ;
026D                ;GET HIS INPUT
026D                ;
026D 2073E9         PUTIN  JSR REDOUT
0270 9100                  STA (PNTR),Y         ;PUT IT IN RAM
0272 C93B                  CMP #';'
0274 F09A                  BEQ GO
0276 C90D                  CMP #$0D
0278 D003                  BNE NCR
```

```
027A 2013EA              JSR CRLOW
027D            ;
027D            ;NO CARRIAGE RETURN
027D            ;
027D 209803     NCR      JSR INCPTR
0280 90EB                BCC PUTIN
0282 A202                LDX #$02
0284 207C03              JSR MSGSUB
0287 203CE9              JSR READ
028A 2013EA              JSR CRLOW
028D 4C4F02              JMP MORE
0290            ;
0290 A203       DIAL     LDX #$03
0292 207C03              JSR MSGSUB
0295 A200                LDX #$00
0297 2073E9     LP3      JSR REDOUT
029A 9520                STA STRING,X
029C C90D                CMP #$0D
029E F003                BEQ LP7
02A0 E8                  INX
02A1 D0F4                BNE LP3
02A3 CA         LP7      DEX
02A4 8608                STX LEN
02A6 AD0402              LDA $204
02A9 8506                STA FINPTR
02AB AD0502              LDA $205
02AE 8507                STA FINPTR+1
02B0            ;
02B0            ;FIND HIS STRING
02B0            ;
02B0 A200       LP5      LDX #$00
02B2 A000                LDY #$00
02B4 B106       LP4      LDA (FINPTR),Y
02B6 D520                CMP STRING,X
02B8 D00A                BNE INCFIN
02BA B520                LDA STRING,X
02BC E408                CPX LEN
02BE F029                BEQ DIALIT
02C0 E8                  INX
02C1 C8                  INY
02C2 D0F0                BNE LP4
02C4 18         INCFIN   CLC
02C5 D8                  CLD
02C6 A506                LDA FINPTR
02C8 6901                ADC #$01
02CA 8506                STA FINPTR
02CC A507                LDA FINPTR+1
02CE 6900                ADC #$00
02D0 8507                STA FINPTR+1
02D2 C503                CMP BTMPTR+1
02D4 90DA                BCC LP5                ;OK
02D6 D006                BNE NOFIND
02D8 A506                LDA FINPTR
02DA C502                CMP BTMPTR
02DC 90D2                BCC LP5                ;OK
02DE            ;
02DE A205       NOFIND   LDX #$05
02E0 207C03              JSR MSGSUB
02E3 203CE9              JSR READ
02E6 4C9002              JMP DIAL
02E9            ;
02E9 2013EA     DIALIT   JSR CRLOW
02EC A000                LDY #$00
02EE B106       LP8      LDA (FINPTR),Y
02F0 C90D                CMP #$0D
02F2 F006                BEQ DODIAL
02F4 207AE9              JSR OUTPUT
02F7 C8                  INY
02F8 D0F4                BNE LP8
02FA            ;
02FA 203CE9     DODIAL   JSR READ
02FD A000                LDY #$00
02FF A200                LDX #$00
0301 B106       LP9      LDA (FINPTR),Y
0303 C93D                CMP #'='
0305 F003                BEQ GOTIT
0307 C8                  INY
0308 D0F7                BNE LP9
030A            ;
030A C8         GOTIT    INY
030B B106       LP10     LDA (FINPTR),Y
```

(continued)

Special Cases

If the AIM cannot find the string you have entered it will respond with:

> AIM: Can't find that name.
> Hit any key to get back to the string enter point.

If your directory is full, AIM will respond with:

> AIM: Out of memory.
> Hit any key and AIM will ask for a new directory ending address.

Hardware

The hardware required to do the actual dialing is shown in figure 1 and is fairly straightforward. Dial pulsing was chosen instead of tones since it is still the only universal method of dialing. Relay R2 is used to short the phone during dialing to suppress annoying clicks and pops. Relay R1 does the actual pulsing.

Program Modifications

The dialer/directory was not written to be relocatable since the AIM 65 is the only machine on which it will run. Modifying it to run on other machines will require a fair amount of work. The only references that make it difficult to relocate in the AIM are the six references to $0200 - $0205.

Longer names may be used by relocating "number" in page zero. This will allow the string to be longer without overrunning the number storage.

The dialing time is set up for standard 10 pulses/second dialing. The make time (set up by subroutine TIM64) is 64 milliseconds. The break time (TIM36) is 36 milliseconds. Inter-digit time is 800 milliseconds caused by jumping to subroutine TIM50 sixteen times. Other dialing methods may call for a change in this timing.

Rod Kreuter is a senior circuit designer for International Telephone and Telegraph in Roanoke, Virginia. At work he uses a Rockwell System 65 to develop 6502 machine controls for ITT, and has an AIM 65 at home. His home system consists of a 4K AIM 65 with a homebrew CRT interface similar to the one described in Rockwell's application note *R6500* N1.2. His hobbies include writing, skiing, and photography.

MICRO

```
030D 9530              STA NUMBER,X
030F C90D              CMP #$0D
0311 F004              BEQ PULSE
0313 C8                INY
0314 E8                INX
0315 D0F4              BNE LP10
0317           ;
0317 A200      PULSE   LDX #$00
0319 8E0BA0            STX $A00B
031C 8E0EA0            STX $A00E
031F A203              LDX #$03
0321 8E02A0            STX $A002
0324 8E00A0            STX $A000
0327 A200              LDX #$00
0329 B530      LP11    LDA NUMBER,X
032B C90D              CMP #$0D
032D F036              BEQ DONE
032F C930              CMP #$30          ;IS IT A ZERO?
0331 D005              BNE NTZERO
0333 A00A              LDY #$0A
0335 4C3D03            JMP RELAY
0338 38        NTZERO  SEC
0339 D8                CLD
033A E930              SBC #$30
033C A8                TAY
033D           ;
033D A901      RELAY   LDA #$01
033F 8D00A0            STA $A000         ;CLOSE RELAY R2
0342 8A                TXA
0343 48                PHA
0344 A20F              LDX #$0F          ;800 MS INTERDIGIT
0346 20C903    LP12    JSR TIM50
0349 CA                DEX
034A D0FA              BNE LP12
034C 68                PLA
034D AA                TAX
034E A900      LP14    LDA #$00
0350 8D00A0            STA $A000         ;OPEN RELAY R1
0353 20D603            JSR TIM64
0356 A901              LDA #$01
0358 8D00A0            STA $A000         ;CLOSE RELAY R1
035B 20BC03            JSR TIM36
035E 88                DEY
035F D0ED              BNE LP14
0361 E8                INX
0362 4C2903            JMP LP11
0365 A9FF      DONE    LDA #$FF
0367 8D00A0            STA $A000         ;OPEN RELAY R2
036A A204              LDX #$04
036C 207C03            JSR MSGSUB
036F 203CE9            JSR READ
0372 C959              CMP #'Y'
0374 D003              BNE REDO
0376 4CE902            JMP DIALIT
0379 4C1002    REDO    JMP GO
037C           ;
037C           ;** SUBS **
037C           ;
037C 2013EA    MSGSUB  JSR CRLOW
037F BD4104            LDA MSGTB0,X
0382 8504              STA MSGPTR
0384 BD4704            LDA MSGTB1,X
0387 8505              STA MSGPTR+1
0389 A000              LDY #$00
038B B104      MSLP    LDA (MSGPTR),Y
038D C93B              CMP #';'
038F F006              BEQ MSOUT
0391 207AE9            JSR OUTPUT
0394 C8                INY
0395 D0F4              BNE MSLP
0397 60        MSOUT   RTS
0398           ;
0398 18        INCPTR  CLC
0399 D8                CLD
039A A500              LDA PNTR
039C 6901              ADC #$01
039E 8500              STA PNTR
03A0 8D0002            STA $200
03A3 A501              LDA PNTR+1
03A5 6900              ADC #$00
03A7 8501              STA PNTR+1
```

```
03A9 8D0102          STA $201
03AC C503            CMP BTMPTR+1
03AE 900A            BCC OK0
03B0 D006            BNE NOTOK
03B2 A500            LDA PNTR
03B4 C502            CMP BTMPTR
03B6 9002            BCC OK0
03B8 38       NOTOK  SEC
03B9 60              RTS
03BA 18       OK0    CLC
03BB 60              RTS
03BC          ;
03BC A9A0     TIM36  LDA #$A0          ;36 MS
03BE 8D08A0          STA $A008
03C1 A98C            LDA #$8C
03C3 8D09A0          STA $A009
03C6 4CE003          JMP TIMOUT
03C9          ;
03C9 A950     TIM50  LDA #$50          ;50 MS
03CB 8D08A0          STA $A008
03CE A9C3            LDA #$C3
03D0 8D09A0          STA $A009
03D3 4CE003          JMP TIMOUT
03D6          ;
03D6 A900     TIM64  LDA #$00          ;64 MS
03D8 8D08A0          STA $A008
03DB A9FF            LDA #$FF
03DD 8D09A0          STA $A009
03E0          ;
03E0 AD0DA0   TIMOUT LDA $A00D
03E3 2920            AND #$20
03E5 F0F9            BEQ TIMOUT
03E7 60              RTS
03E8          ;
03E8          ;** TABLES **
03E8          ;
03E8 444941   M0     ASC 'DIAL(D) OR ENTER(E)?;'
03EB 4C2844
03EE 29204F
03F1 522045
03F4 4E5445
03F7 522845
03FA 293F3B
03FD 4E4557   M1     ASC 'NEW(N) OR ADD(A)?;'
0400 284E29
0403 204F52
0406 204144
0409 442841
040C 293F3B
040F 4F5554   M2     ASC 'OUT OF MEMORY.;'
0412 204F46
0415 204D45
0418 4D4F52
041B 592E3B
041E 4E414D   M3     ASC 'NAME?;'
0421 453F3B
0424 524544   M4     ASC 'REDIAL?;'
0427 49414C
042A 3F3B
042C 43414E   M5     ASC 'CAN''T FIND THAT NAME;'
042F 275420
0432 46494E
0435 442054
0438 484154
043B 204E41
043E 4D453B
0441          ;
0441 E8       MSGTB0 BYT M0
0442 FD              BYT M1
0443 0F              BYT M2
0444 1E              BYT M3
0445 24              BYT M4
0446 2C              BYT M5
0447          ;
0447 03       MSGTB1 HBY M0
0448 03              HBY M1
0449 04              HBY M2
044A 04              HBY M3
044B 04              HBY M4
044C 04              HBY M5
```

New Publications

(Continued from page 25)

Graphics

IEEE Computer Graphics and Applications, a new quarterly which began in January 1981, is published by the IEEE Computer Society (10662 Los Vaqueros Circle, Los Alamitos, California 90720). It is edited for designers and users in all computer graphics application areas such as business graphics; test and measurement; process control and instrumentation; navigation and guidance; consumer electronics; military electronics; patient care; petrochemicals; communication; transportation; CAD/CAM; VLSI design; education. An annual subscription is $8.00 plus society member dues ($14.00) or $23.00 for nonmembers.

Computer Graphics News is a bimonthly tabloid to begin in September in conjunction with the annual meeting of the National Computer Graphics Association in Baltimore. The newspaper will be sponsored by the association and published by Scherago Associates, Inc. (1515 Broadway, New York, New York 10036). The publisher plans an initial controlled circulation of 25,000 to individuals interested in computer graphics.

Biomedical

Computers in Psychiatry/Psychology is a 16-page bimonthly newsletter founded in 1978, devoted to the field of mental health. It covers such subjects as the computerization of the professional office and computer-based diagnosis. An annual subscription is $25.00 from Computers in Psychiatry/Psychology, 26 Trumbull Street, New Haven, Connecticut 06511.

National Report on Computers and Health is an 8-page, biweekly newsletter edited for health professionals and the information processing industry — vendors, users, consultants, associations, and government. It covers scientific developments, market intelligence, new products, government regulatory activities, and new initiatives in university medical centers, in the National Center for Health Services Research, and among consultants. An annual subscription is $192.00 for 25 issues from National Report, P.O. Box 40838, Washington, D.C. 20016.

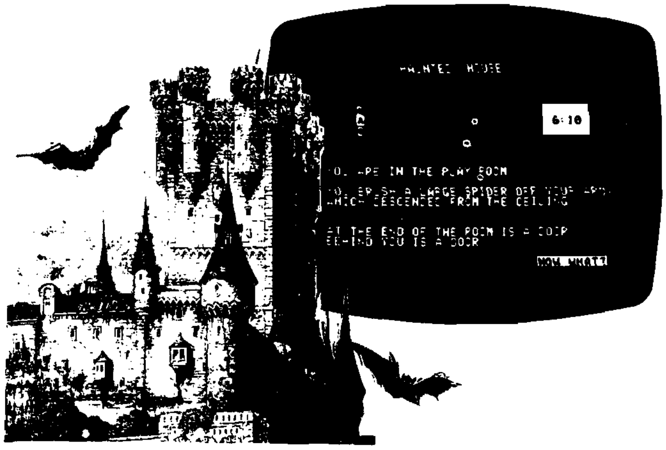
(Continued on page 101)

Apple II sensational software

creative computing software

Haunted House

Cassette CS-4005
$11.95
Requires 16K

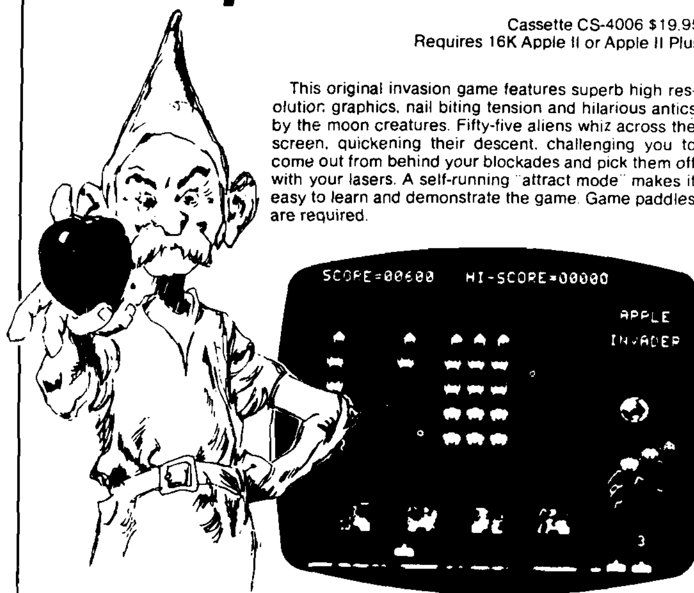


It's 6:00 pm and you have until midnight to find the secret passageway out of a large rambling HAUNTED HOUSE. During your search you'll encounter skeleton keys, charms, friendly ghosts, and evil spirits. Sound effects add to the eerieness. The layout changes in every game.

Super Invasion

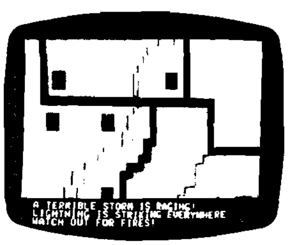
Cassette CS-4006 $19.95
Requires 16K Apple II or Apple II Plus

This original invasion game features superb high resolution graphics, nail biting tension and hilarious antics by the moon creatures. Fifty-five aliens whiz across the screen, quickening their descent, challenging you to come out from behind your blockades and pick them off with your lasers. A self-running "attract mode" makes it easy to learn and demonstrate the game. Game paddles are required.

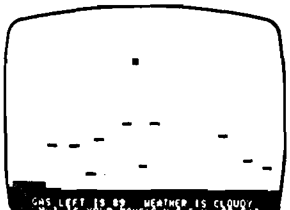


Outdoor Games

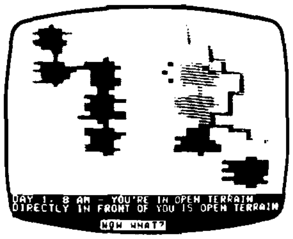
Cassette CS-4010 $14.95 4 Programs Requires 16K Apple II or Apple II Plus



Forest Fire. Use chemical retardants and backfires to control raging forest fires.



Fishing Trip. Try to catch flounder and salmon while avoiding logs, sharks, bad weather and running out of fuel.



Treasure Island I. Your map shows buried treasure but unfortunately you don't know where you are. Try to find the treasure while moving about and observing your surroundings. You have a 3-day supply of food and water. You may find useful objects (compass, weapons, a horse) but watch out for hazards (robot guards, pirates, caves, crocodiles, mountain lions and more).

Treasure Island II. Same game except you have to use a metal detector to find the treasure.

Space War

Cassette CS-4009 $14.95
Requires 16K
Apple II or Apple II Plus

Take command in Space War. Select from five game modes, including reverse gravity, and the battle begins. Challenge your opponent with missle fire, force him to collide with the sun or to explode upon re-entry from hyperspace. Be wary/.. He may circle out of sight and re-appear on the opposite side of the galaxy. (This is the classic MIT game redisgned especially for the Apple.)

Outdoor Games and Haunted House

Disk 4504, $24.95
Requires 32K Apple II or Apple II Plus

This disk contains all five programs from cassettes CS-4005 and CS-4010.

Super Invasion Space War

Disk CS-4508 $29.95
Requires 48K Apple II or Apple II Plus

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. Visa, MasterCard and American Express orders may be called in toll-free

Order today at no risk. If you are not completely satisfied, your money will be promptly and courteously refunded

Attn: Doreen
Creative Computing Software
Morris Plains, NJ 07950
Toll-free **800-631-8112**
In NJ, 201-540-0445

creative computing software

Hot pursuit through space and the vortices of time!

RAMWARE PRESENTS. . .

Time Lord

The fallen Time Lord, who presumptuously calls himself The Master, is at large. The elders of Waldrom have supplied you with the hyperspace-worthy vessel Tardus, and commissioned you to eliminate the evil "Master". Your resources include clones who will fight for you, the formidable CRASER weapons of the Tardus, and magic weapons such as Fusion Grenades and Borelian Matrix Crystals.

Traveling through hyperspace in search of the evil one, you will encounter Time Eaters, Neutron Storms, and other alien creatures and phenomena. Entering real space to search planets, you will encounter still other dangers. You will enter native settlements to buy food and supplies — or to fight for survival.

And once you find The Master can you destroy him?

Based on Dr. Who of PBS fame.
Apple Integer Basic,
Disk, 48K . . . $29.95

TSE-HARDSIDE
6 South St., Milford, NH 03055 (603) 673-5144
TOLL FREE OUT-OF-STATE 1-800-258-1790


TSE-HARDSIDE
An Interactive Approach
BASIC
FOR HOME COMPUTERS
A SELF-TEACHING GUIDE
BOB ALBR
LeROY FIN
ERALD R.
TRS-80
BASIC


ATARI 400
I'm a computer orphan!
Save $2.
If you or a friend would like to receive our TSE/HARDSIDE Complete Computer Catalog, simply send $1.00 and receive a $2.00 certificate good toward your first purchase of software, hardware, books or specialty items from TSE/HARDSIDE. I have a (check one) ☐ TRS-80, ☐ APPLE ☐ ATARI ☐ PET.
A photocopy of this coupon will be accepted.
Name
Address
City State Zip
NIBB
MAIL TO: TSE/HARDSIDE
6 South Street
Milford, NH 03055

Macros for Micros

An introduction to the MACRO assembler.

John Figueras
65 Steele Rd.
Victor, New York 14564

Macro definition is a common feature of the advanced assemblers available on large computers. To my knowledge, the only 6502-based assembler with this capability is the ASSM/TED 6502 Macro Assembler sold by Carl Moser.[1] I will describe practical applications of macros to programming an Apple II computer, and show how to set up a macro library that can be stored on disk, and which may be used as a subroutine generator to supply utilities that will simplify machine language programming.

A macro definition is a predefined block of assembler code that is assembled into the machine language program wherever the macro is called. An example of a macro definition is shown in figure 1. (All examples use the notation of ASSM/TED and were written for the Apple II computer.) The three exclamation marks designate the subsequent name, KEYB, as the name of a macro definition. It is by this name that the macro is called in the program. The pseudo op-codes, .MD and .ME define, respectively, the beginning and end of the macro definition. The statement(s) falling between these comprise the *body* of the definition. The macro in figure 1 doesn't do much—it simply calls the Apple keyboard routine. You might wonder "Why all the fuss for *this*?" But consider that we may now replace a call to the hexadecimal address $FD67 of a keyboard subroutine with the mnemonic, KEYB. Then, in creating a source program, to call the keyboard I simply use KEYB, which is much easier to remember than JSR $FD67. Essentially, macros allow you to create your own convenient programming symbols.

A macro is called in an assembly language program by using the macro name as an opcode. (Examples will be shown later.) When the program is assembled, code contained in the body of the macro definition will be inserted in place of the macro name wherever it occurs. For example, wherever I use the name KEYB, as defined in figure 1, the assembler will substitute the machine code equivalent, a JSR $FD67. If the body of this macro definition contained twenty assembly language instructions, then all twenty statements would be assembled into the program. This can be a problem, since indiscriminate use of macros can lead to undesirable inflation in the amount of memory required for the program.

It may be difficult for beginning machine language programmers to grasp the difference between a macro and a subroutine (at least, I had this difficulty). There is a superficial resemblance between the two, since each is a block of statements that is called in a program. But the resemblance ends there. A subroutine is a block of code that occurs only once in a program and is called by a branch instruction, which diverts the program flow to the subroutine. Provision is made for a return to the calling program by storing a return address when the subroutine is called. A macro, on the other hand, produces in-line code during assembly each time the macro is called. While they use more memory space, macros are more efficient because they do not require subroutine branch and return instructions.

Application to Utilities Storage

One problem facing the machine language programmer is that of handling utility routines, particularly those for input/output operations. The Apple monitor contains a large number of these utilities, which may be called by the user's programs, with a JSR. The task of finding and interpreting these

***Figure 1:* Example of a Macro Definition**

```
>PR
;READ KEYBOARD. # CHAR IN NCHAR
!!!KEYBRD  .MD
           JSR $FD67
           STX NCHAR
           .ME
;
;DISPLAY BUFFER ON CRT
!!!DISPLAY .MD
```

***Figure 2:* Macro Library of I/O Utilities**

```
>PR
;READ KEYBOARD. # CHAR IN NCHAR
!!!KEYBRD  .MD
           JSR $FD67
           STX NCHAR
           .ME
;
;DISPLAY BUFFER ON CRT
!!!DISPLAY .MD
           SEC
           LDX #$00
...LOOP1   LDA BUFFER,X
           ORA #$80
           JSR $FDF0
           INX
           CPX NCHAR
           BCC ...LOOP1
           .ME
;
;ASSIGN FIXED ADDRESSES
!!!INIT    .MD
           .OS
           .ES
NCHAR      .DS 01
BUFFER     .DE $0200
ZPAGE      .DE $4A
           .ME
//
```

***Figure 3:* Example of Keyboard/CRT I/O**

```
;SAMPLE PROGRAM 1
;READ & DISPLAY KEYBOARD ENTRY
           .BA $5000
           INIT
;DEFINE SUBROUTINES
VIDEO      DISPLAY
           RTS
KEYIN      KEYBRD
           RTS
TRIAL      JSR KEYIN
           JSR VIDEO
           RTS
           .EN
```

utilities has been considerably eased by the publication of *The Apple II Monitor Peeled*[2], which describes the functions and locations of a large number of important routines. These include reading the keyboard, sending characters to a CRT, defining the location of the input buffer, cursor manipulation and many others. Until this volume was released, it was difficult to know how to use the monitor routines that Apple kindly listed in their reference manual[3], without any explanation.

Though the information for applying the monitor routines is now available, one still needs to know a number of memory addresses to use them. Casual programmers, like myself, have to look these up repeatedly because we forget the addresses from one programming session to another. Moreover, many of the routines require small drivers to run them, and I find that I can't remember how I wrote the driver last time any better than I can remember the addresses! It would be convenient, therefore, to pre-program the most-needed utilities, store them on disk, and call them from disk for insertion into a program. One would then have a *subroutine library*, like those used to support programming on large computers. Or, (and this is the direction I chose), one could store the same information in a *macro library*.

The tendency of macros to use up memory can be overcome by calling the macro inside a subroutine. The macro library is loaded into the ASSM/TED text buffer; the required subroutine is formed by setting up the desired subroutine name, calling the appropriate macro out of the library into the subroutine, and closing with an RTS. The macro is assembled only once and may now be used repeatedly by means of subroutine calls, without direct use of macro calls. One can use macro calls directly, without subroutine calls. If the macro block appears only once in a program, or if it is very short, this avoids the overhead of subroutine calls. However, if the macro block is long, and is used more than once, then putting the macro call in a subroutine is more efficient.

Sample Application: I/O Utilities

Leaving these abstract considerations, let's look at some implementations. Figure 2 is a listing of a small macro library comprising three modules. The first one, KEYBRD, allows the Apple keyboard to be read by means of a call to a monitor subroutine at $FD67. The monitor routine loads keyboard input into a text buffer located at $0200 and stores the character count in the X register. In the macro definition, this character count is transferred for later use to a memory location NCHAR. This memory location must be assigned before the macro is called. This is taken care of by another macro, INIT, which will be discussed later.

The second macro definition, DISPLAY, sends the contents of the Apple text buffer, character by character, to the CRT, by a call to a monitor subroutine at $FDF0. Note that the text buffer is addressed by a name, BUFFER, which is assigned in the macro, INIT. The character count, NCHAR, is required to control the number of characters sent to the CRT. This is the same count created in KEYBRD. The internal loop, ...LOOP1, is named with three opening dots, in accordance with Moser's requirements in ASSM/TED. This convention permits the macro definition to be used several times within a program. Each use will generate a new label to replace LOOP1, otherwise location conflicts for the label would occur. If the macro definition is used only once, this precaution is not necessary; I invoked it to allow greater freedom of use of the macro.

The third macro definition, INIT, initializes several assembler parameters and assigns storage for variables. The .OS pseudo-op must be included in every source program to enable compilation of machine code. The pseudo-op .ES enables the listing of the machine code derived from expansion of macros. If it is not present, the machine code due to macros will not appear in the output listing. Since I want .OS and .ES to appear in the programs I write, I include them in INIT, and avoid the need to remember them. Also included in INIT is the assignment of storage for NCHAR (.DS 01 reserves one byte of storage), assignment of the address of the input buffer, $0200, to the label BUFFER, and definition of a zero page address, ZPAGE. Note that the three macros taken together have eliminated the need to remember four addresses, and have given me by-name access to two variables, NCHAR and BUFFER. Because of its function, INIT must be the first statement in a program after definition of program origin, since it defines locations of variables needed by other macros.

Figure 3 illustrates the use of macros in subroutine generation. The program, TRIAL, reads the keyboard

***Figure 4:* Example Using Direct Macro Calls**

```
              .BA $5000
              INIT
TRIAL         KEYBRD
              DISPLAY
              RTS
              .EN
```

***Figure 5:* Display a Message from Memory**

```
>
>;SAMPLE PROGRAM 2
;DISPLAY MESSAGE IN MEMORY
           .BA $500
           INIT
;DEFINE SUBROUTINE
VIDEO      DISPLAY
           RTS
MSG        .BY 'MESSAGE 1'
TEMP       .BY 09
TRIAL      LDA TEMP
           STA NCHAR
           LDX #$00
LOOP       LDA MSG,X
           STA BUFFER,X
           INX
           CPX NCHAR
           BNE LOOP
           JSR VIDEO
           RTS
           .EN
//
```

***Figure 6:* Macros for Data Transfer with Address Passing**

```
;MACROS TO TRANSFER CHARS
;FROM MEM TO BUFFER
!!!PASSADR .MD (MSG CNT)
           LDA CNT
           STA NCHAR
           LDA #L,MSG
           STA ZPAGE
           LDA #H,MSG
           STA ZPAGE+01
           .ME
!!!MEMBUFF .MD
           LDY #$00
LOOP2...   LDA (ZPAGE),Y
           STA BUFFER,Y
           INY
           CPY NCHAR
           BNE LOOP2
           .ME
//
```

***Figure 7:* Program to Display Two Messages Using Macros in Figure 6**

```
;
;SAMPLE PROGRAM 3
;DISPLAY TWO MESSAGES FROM MEM
           .BA $5000
           INIT
;DEFINE SUBROUTINES
MESSAGE    MEMBUFF
           RTS
VIDEO      DISPLAY
           RTS
MSG1       .BY 'FIRST MESSAGE'
           $8D
CNT1       .BY =-MSG1
MSG2       .BY 'SECOND MESSAGE'
           $8D
CNT2       .BY =-MSG2
TRIAL      PASSADR (MSG1 CNT1)
           JSR MESSAGE
           JSR VIDEO
           PASSADR (MSG2 CNT2)
           JSR MESSAGE
           JSR VIDEO
           RTS
           .EN
//
```

and displays the entry. (A double display will occur because the monitor routine KEYBRD also provides an echo.) The program is assigned an origin at $5000 by the pseudo-op .BA. INIT is called to initialize variables and pseudo-ops. Two subroutines are defined. The first one, VIDEO, sends characters to the CRT and its body is loaded from the macro DISPLAY (figure 2). The second one, KEYIN, enables keyboard input; it is loaded from the macro KEYBRD (figure 2). The simple structure of these subroutines masks the complexities that may be built into the macro definitions. The program starts at the label TRIAL.

Following invocation of the two subroutines, RTS returns control to the assembler. The closing .EN defines the end of the program to the assembler. This program really does not require the use of subroutines, but is a simple example of how subroutines could be defined. Since the macros in figure 3 are used only once, the very brief program in figure 4, based on direct macro calls, is a more reasonable implementation.

The second program example, figure 5, displays a message stored in memory (that is, one written into the program). The macros defined in figure 2 are used, except for KEYBRD, since there is no keyboard input. In figure 5, the subroutine VIDEO is defined as before. The message to be displayed is stored as a character string in a location labelled MSG (.BY means "define bytes"). The number of characters in the message is stored in a location named TEMP.

Program TRIAL begins by transferring the character count stored in TEMP to NCHAR, where it can be used by DISPLAY. The loop makes a character-by-character transfer from message location MSG to the display BUFFER, which is accessed in the subroutine VIDEO.

We note in the above program that code is used to transfer data from memory into the display buffer. Since this transfer is likely to be used repeatedly as a basic operation in displaying labels and instructions, it would be desirable to turn this code into a macro definition for use in the body of a subroutine. An immediate difficulty arises from the fact that the message and character count (MSG and TEMP) occur at fixed addresses. Other messages and counts which are at different addresses are not accessible to this program. If a subroutine is set up to pass data to the display buffer from memory, we would like to be able to pass the addresses of the message and the message count to the subroutine, so that it can be applied wherever these data fall in memory. It turns out that passing addresses to a subroutine requires a surprising amount of code (see the remarks by R.C. Vile[4]).

However, the macro language in ASSM/TED permits addresses to be passed to macro definitions. We would like to take advantage of this without the high memory overhead that repeated use of large macros might entail. The solution is to partition the macro into a small segment that does the address passing, and a larger segment that operates on the data in the passed addresses. Addresses passed by the small segment are stored in fixed memory locations accessible to the large segment. In programming applications, the small segment could be used without much memory overhead as a *macro*, and the large segment could be used as the body of a subroutine. An example of such a partition appears in figure 6, which contains the data transfer segment of the program in figure 5.

The first macro definition in figure 6, PASSADR, enables the passing of two addresses, MSG and CNT, of the message and message count. In ASSM/TED convention, these addresses appear as arguments in parentheses following the macro name. PASSADR uses the address of CNT, whatever that may be in the program, to transfer the count stored there to the pre-defined location NCHAR. The high and low bytes of address MSG are stored by PASSADR in zero page addresses ZPAGE and ZPAGE+01. The actual moving of data from memory location MSG to the display buffer is done in the second macro, MEMBUFF. This routine uses indirect indexed addressing based on ZPAGE for getting the data in MSG. The ZPAGE location must, of course, be defined, and this is done in the macro INIT (see figure 2). MEMBUFF can be used to form the body of a subroutine.

An application of these two new macros for displaying *two* messages stored in memory appears in figure 7. The messages are stored as bytes (.BY) in addresses MSG1 and MSG2. The required character counts are calculated by using the ASSM/TED pseudo-op "=" to get the current value of the program counter, and then subtracting from it the address of the corresponding message (e.g., = − MSG1). Since the program counter is read after the definition of the message, the difference between that reading and the address of the beginning of the message must give the message length in bytes. The messages themselves are terminated by a carriage return, $8D, to allow each message to appear on a separate line. PASSADR is used twice in the program with two different sets of addresses in parentheses. PASSADR is used as a macro in the program, while MEMBUFF is used to supply the body of a subroutine, MESSAGE.

The emphasis so far has been on the use of macro definitions as a means to create the equivalent of a subroutine library. There are other ways in which a subroutine library may be created, but I consider the use of macro definitions described here as the least troublesome and most flexible way in which to formulate such a library. Given the language resources of the Apple computer, it is also the most memory-conserving way.

Conclusion

You may be persuaded by now that the use of macro definitions offers a very powerful programming tool to the machine language programmer. Its most interesting spin-off is that it allows you to design your own programming language at the machine code level. The examples in this article barely scratch the surface of possible applications. One area in which macros are useful is arithmetic operations. One can design macros for addition, subtraction, multiplication and division of sixteen bit numbers, and define double precision versions of these macros. The addresses of the numbers to be operated on could be passed as arguments in the macro definitions. And then there are high and low resolution graphics... and floating point arithmetic... and array definition... and....

References

1. ASSM/TED 6502 Macro Assembler, Carl W. Moser, Eastern House Software, 3239 Linda Drive, Winston Salem, North Carolina 27106.

2. *The Apple II Monitor Peeled*, William E. Dougherty, 14349 San Jose St., Mission Hills, California 91345. (Widely available through vendors.)

3. *Apple II Reference Manual*, Apple Computer, Inc., January 1978.

4. R.C. Vile, Jr., MICRO, Issue 20, pg. 25 (January 1980).

MICRO

Create a Data Disk for DOS 3.2 and 3.2.1

Save space on your Apple data disks by eliminating the copy of DOS.

Glenn R. Sogge
Fantasy Research & Development
P.O. Box 203
Evanston, Illinois 60204

According to the information in the DOS 3.2 manual, an initialized disk contains 403 sectors (of a total 455) that can be utilized for the storage of user information. (User information also includes some file overhead of track and sector lists.) This amounts to 103,168 bytes of memory space or 88.57% of the maximum storage capacity of the disk. (The maximum storage is 13 * 35 * 256 = 116,480 bytes.) This article explains how to increase the user storage to 112,640 bytes or 96.70% of the maximum—an increase of 8.13%! Given the limited storage capabilities of 5¼ inch disks to begin with, this improvement can be quite important—especially for business and data base software.

The cost of this increase in storage is the loss of the DOS on the disks. This is not too high a price, however, because we usually don't need dozens of copies of DOS floating around. In general, the user will boot the system up and use the DOS that is then residing in the machine, using the disks only for information storage and handling. Even though a program may use many different disks, the DOS that is written on each one is generally useless, but still takes up three tracks of space (9984 bytes).

One advantage of having the DOS on every disk is that any disk is bootable. The procedure outlined here will create data disks that are bootable with an overhead of only 2 sectors (512 bytes) besides the directory track (track $11).

A Note on Notation

In this article, tracks, sectors, and relative bytes (within the sector) will be indicated like this:

11,C,AC

The contents of such locations will be indicated like this:

(11,C,AC) = FF

or

(11,4,00) = 0 1 FD 38
(successive bytes)

All numbers will be in hexadecimal so the '$' should be assumed if not present.

Beginnings

The simplest way to gain more space is to change the bitmap in the VTOC to free up the sectors occupied by the DOS. By changing the contents of the bytes at (11,0,38-43), we can deallocate the sectors normally reserved for DOS. Several of the disk utilities commercially available have just such an "expunge" routine. The problem with this simple method is that the disk will probably hang when booted, because either new information will have been stored in the sectors that contain the secondary boot code, or portions of DOS will keep disappearing as more information is stored.

Since we want to free up this space anyway, we will begin by changing the bitmap and then worry about making the disk boot later. With one of the disk utilities available, read in the (11,0) sector and make the following changes, then rewrite it to the disk:

(11,0,38) to FF E0 00 00
(11,0,3C) to FF F8 00 00
(11,0,40) to FF F8 00 00

These changes free up all of the sectors of the first three tracks except for sectors 0 and 1 of track 0. These will be used to make the disk bootable.

How a Disk Boots

When a disk boots, the first sector (0,0) is read into memory, unscrambled, and placed at $300 - $3FF. This code then begins reading in from sector (0,0) again and places the code into memory. The number of sectors of track zero that are to be read in, and where they are to be stored, can be easily modified. The byte at (0,0,FF) contains the highest sector value to read, times 8, and the byte at (0,0,FE) contains the page address of where to begin storing the code.

After the track 0 sectors are read in, the code jumps to the memory location where sector (0,1) has been stored and continues execution. With a normal disk, this code is the third stage of the boot, and the RWTS routines read in the rest of DOS and start it running. For example, if (0,0,FF) is $48, sectors 0 through 9 will be read into memory ($9 times $8 equals $48). If (0,0,FE) is $36, sector (0,0) goes at $3600, (0,1) at $3700, (0,2) at $3800, and so on. After the requisite number of sectors have been read in, execution will continue at $3700.

By changing the bytes at (0,0,FE) and (0,0,FF) and placing new code in sector (0,1), the boot routines will automatically load and execute it. (For those of you who have tried to figure out the page 3 boot code, the value of (0,0,FE) ends up at $3CC and the value of (0,0,FF) ends up at $3FF.)

The Data Disk Routine

The routine on the data disk should notify the user that there is no DOS present, and then gracefully return to the user. Most expunge routines don't do this and somehow cause the routine to abort, or require the user to press reset to gain control of his machine. If the machine has the Autostart ROM, even resetting may not work because the first part of the boot will have crashed the page 3 PWREDUP vector bytes, thus causing the ROM to think that it is the first time through the procedure. It then begins the boot process all over again by looking for a disk and starting up the boot.

This is clearly inelegant and totally unacceptable in a turnkey system. The system should trap all foreseeable user errors and handle them, without requiring the user to be a computer operator. The user should be able to put any disk in the system (even if by mistake) and not have the roof fall in. In other words, as far as the booting procedure is concerned, one sequence of actions is all the user needs to learn.

The short routine accompanying this article is an example of the kind of routine required. The routine first disconnects the I/O hooks in page zero, resets to keyboard and video mode, and clears the screen. The drive is turned off and a message notifying the user that DOS is not present is displayed. BASIC is then entered at the cold start point. (This could be changed to warm start BASIC if desired.) The user now knows what went wrong and can decide how to proceed.

You will notice that the routine to print out the error message is written in a way that is relocatable. This was done so that the code would run from any page in memory; the value of this capability is discussed in the next section.

```
                        ********************************
                        * DOS DATA DISK CODE
                        ********************************
                        *
                        * BY GLENN R. SOGGE
                        * FANTASY RESEARCH
                        * & DEVELOPMENT
                        *
                        * P.O. BOX 203
                        * EVANSTON, IL
                        * 60204
                        *
                        * LAST REVISION
                        * 5/23/80
                        *
                        ***********
                        *
                        * THIS CODE GOES
                        * ON TRACK ZERO,
                        * SECTOR 1
                        *
                        * IT WILL RUN FROM
                        * ANY PAGE BOUNDARY
                        *
                        ***********
                        *
                        SETVID    EQU  $FE93
                        SETKBD    EQU  $FE89
                        A1        EQU  $3C
                        COUT      EQU  $FDED
                        HOME      EQU  $FC58
                        BASIC     EQU  $E000
                        RTS1      EQU  $F831
                        SLOT      EQU  $2B
                        MOTOFF    EQU  $C088
                        *
                                  ORG  $8500
                        *OBJ $8500
                        *
8500: 20 93 FE          NOBOOT    JSR  SETVID       UNHOOK DOS
8503: 20 89 FE                    JSR  SETKBD       POINTERS
8506: 20 58 FC                    JSR  HOME         CLEAR SCREEN
8509: A6 2B                       LDX  SLOT         WHO CALLED?
850B: 9D 88 C0                    STA  MOTOFF,X     TURN HIM OFF
850E: 20 31 F8                    JSR  RTS1         WHAT PAGE AM I ON?
8511: BA                          TSX
8512: CA                          DEX
8513: 9A                          TXS
8514: 68                          PLA
8515: 85 3D                       STA  A1+1         POINT A1 TO
8517: A9 2A                       LDA  #MSG         THE MESSAGE
8519: 85 3C                       STA  A1
851B: A0 00                       LDY  #$00
851D: B1 3C             PRLOOP    LDA  (A1),Y       PRINT OUT THE
851F: F0 06                       BEQ  DONE         MSG TO USER
8521: 20 ED FD                    JSR  COUT
8524: C8                          INY
8525: D0 F6                       BNE  PRLOOP
8527: 4C 00 E0          DONE      JMP  BASIC        GO TO LANGUAGE
                        *
852A: CE CF A0          MSG       ASC  "NO          DOS ON THIS DISK"
853D: 87                          DW   $87          BELL
853E: 00                          DW   $00
                        *
                                  SYM
```

Putting it Together

Now that we have an understanding of the booting process and a routine to use with it, it's time to put them together. Since the ROM boot routine crashes pages 8 and 9 with its "nibble buffers," a good place to put the new code is right above them, to keep all the damage in one area. To do this, change the byte (0,0,FE) to $0A and the byte at (0,0,FF) to $08. This changes the boot to read in sectors (0,0) and (0,1), to place them in memory starting at $A00, and to jump to $B00. The error routine should be placed on the disk at (0,1) and it will end up in memory at $B00 ready to run.

If, for some reason, pages $A and $B are inappropriate to your system or programs, change the value of (0,0,FE) to a page that is more suitable. (The routine was made relocatable for this reason.) Pages $8 and $9 cannot be used because these buffers are necessary for reading in the code.

The Master Disk

This procedure is not an unreasonable amount of work to do once or twice, but it is not something you would want to turn into a habit. So, master data disk that can then b copied as many times as needed shoul be made. *Note:* some copy program may not copy information from, or tc the normal locations that DOS oc cupies on a disk. If your program is c this kind, you'll have to transfer th (0,0) and (0,1) sectors manually to th new disk. The modified VTOC shoul be copied correctly.

The following is the general procedural outline:

1. Initialize a disk in the normal manner.
2. Delete the 'HELLO' program.
3. Change the VTOC bytes as outlined above.
4. Change the sector (0,0) bytes as outlined above.
5. Put the error routine on sector (0,1).
6. Test the disk by booting it.
7. Make a copy of the disk.
8. Boot the copy disk.

If everything is okay, you now have a master data disk (with no files on it) from which to generate more.

Notice that no change is made in the VTOC to the bits corresponding to track $11 (the directory and the VTOC). This track is kept 'unavailable' so the directory and the VTOC will still be there for the DOS that accesses the disk.

Extensions

The experienced machine language hacker can extend this technique to create disks that automatically load and run machine language programs, as long as they fit completely on track 0 or if they include the RWTS routines and controlling code to read in more of the disk. If you examine the code on a normal disk at sector (0,1), you will see the type of code required.

The designers of operating systems can change or replace all or part of the Apple DOS by changing the contents of the sectors normally occupied by DOS, and letting the various boot routines bring it into memory. This generally requires using the existing RWTS code on track 0 and something similar to the third stage boot code that starts with sector (0,1), but it is not necessary. The programmer can create a whole new system if desired.

By utilizing the Apple RWTS routines that normally reside on track 0, the disks of different operating systems can be physically compatible even though the information structures may not be. There are already enough incompatible DOS's and physical formats around in the micro world; I hope that as more DOS's develop for the Apple, their underlying physical structure will remain the same. Some alternatives are needed to the Apple DOS for various users, but the media shouldn't be incompatible at all levels.

I, for example, am working on an implementation of FIG-Forth (the Forth Interest Group's definition of a minimal standard Forth) for the Apple, and plan to use the standard RWTS routines and linkages—but not the whole DOS—to allow Forth access to the disks created under 3.2 and BASIC, and *vice versa*. Different languages and operating systems allow alternative processing operations on the same information, but only if the information is physically accessible.

I hope this article can contribute to the development of such systems and would like to hear from anyone working along these lines.

MICRO

apple][

Apple Color Filter

This short machine language subroutine will allow you to filter out any selected color from the Apple hi-resolution graphics screen.

Stephen R. Berggren
2347 Duncan Dr. #4
Fairborn, Ohio 45324

One of the most fascinating capabilities of the new Apple Graphics Tablet is its ability to separate the colors on the high resolution graphics screen. It can act like a color filter, removing all colors from the screen except a chosen one. This can be extremely useful in doing computer art work, drawing graphs, and, of course, in game graphics. But now you can have a similar capability without buying the graphics tablet. Just use this Apple color filter program.

The color filter is a short machine language program which can erase any selected color from the high resolution screen while leaving the other colors unaffected. To use it, simply load it into page 3 of memory, starting at decimal 768. Then POKE a number from 1 to 4 into memory location 769 and run it with a call 768. The number POKEd into 769 determines what color is erased: 1 erases green, 2 erases violet, 3 erases blue and 4 erases orange. The program takes only about one fourth of a second to filter the entire page one Hi-Res screen.

If you are using only green, violet, blue and orange, everything works fine. But the Apple also draws in white—in fact two kinds of white. This can affect the results of the filter operation. The Apple makes its two whites by combining either green and violet (HCOLOR=3) or blue and orange (HCOLOR=7). The color filter "sees" the white as a combination of the two colors rather than as a separate color.

```
                  ;************************
                  ;*                      *
                  ;*  APPLE COLOR FILTER  *
                  ;*                      *
                  ;* BY STEPHEN BERGGREN  *
                  ;*                      *
                  ;************************
                  ;
                  ;PUT NUMBER FOR COLOR TO BE REMOVED IN $301
                  ; 1 = GREEN, 2 = VIOLET, 3 = BLUE, AND 4 = ORANGE
                  ; WHITE #3 NOT AFFECTED BY 3 OR 4
                  ; WHITE #7 NOT AFFECTED BY 1 OR 2
                  ;
                  ;TO RUN, 300G FROM MONITOR OR CALL 768 FROM BASIC
                  ;
                  SCRLOC EPZ $06              ;ZERO-PAGE LOC. FOR ADDRESSING SCREEN
                  LOSCRN EPZ $20              ;HI-BYTE OF ADDRESS OF SCREN START
                  HISCRN EPZ $40              ;HI-BYTE OF SCREEN END
                  ;
0300                     ORG $300
0300              ;
0300 A200                LDX #$00             ;PUT COLOR VALUE IN X FOR TABLE INDEXING
0302 A000                LDY #$00             ; PUT 0 IN Y FOR INDIRECT SCREEN INDEXING
0304 A900                LDA #$00             ; SET SCRREN START ADDRESS IN SCRLOC
0306 8506                STA SCRLOC
0308 A920                LDA #LOSCRN
030A 8507                STA SCRLOC+1
030C              ;
030C B106         EVNBYT LDA (SCRLOC),Y       ; GET SCREEN BYTE
030E 3008                BMI DOTAB2           ; IF BIT 7 SET, USE TABLE 2
0310 3D4503              AND TABLE1,X         ;MASSK OFF COLOR BITS USING TABLE 1
0313 9106                STA (SCRLOC),Y       ;PUT BACK THE BYTE
0315 4C1D03              JMP ODDBYT           ;DO THE NEXT BYTE
0318              ;
0318 3D4703       DOTAB2 AND TABLE2,X         ;MASK OFF COLOR BITS USING TABLE 2
031B 9106                STA (SCRLOC),Y       ;PUT BACK THE BYTE
031D              ;
031D E606         ODDBYT INC SCRLOC           ;SET UP FOR NEXT SCREEN BYTE
031F B106                LDA (SCRLOC),Y       ;GET SCREEN BYTE
0321 3008                BMI DOTAB4           ;IF BIT 7 SET, USE TABLE 4
0323 3D4903              AND TABLE3,X         ;MASK OFF COLOR BITS USING TABLE 3
0326 9106                STA (SCRLOC),Y       ;PUT BACK THE BYTE
0328 4C3003              JMP INCLOC           ;GO INCREMENT SCRLOC
032B              ;
032B 3D4B03       DOTAB4 AND TABLE4,X         ;MASK OFF COLOR BITS USING TABLE 4
032E 9106                STA (SCRLOC),Y       ;PUT BACK THE BYTE
0330              ;
0330 A900         INCLOC LDA #$00             ;INCREMENT SCRLOC LO
0332 38                  SEC
0333 6506                ADC SCRLOC
0335 8506                STA SCRLOC
0337 90D3                BCC EVNBYT           ;IF NOT OVERFLOW, DO ANOTHER 2 BYTES
0339 A900                LDA #$00             ;INCREMENT SCRLOC HI
033B 38                  SEC
033C 6507                ADC SCRLOC+1
033E 8507                STA SCRLOC+1
0340 C940                CMP #HISCRN          ;WAS THAT THE LAST PAGE?
0342 D0C8                BNE EVNBYT           ;IF NOT, DO NEXT 2 BYTES
0344 60                  RTS                  ;ALL DONE!
0345              ;
0345 00D5         TABLE1 HEX 00D5
0347 AAFF         TABLE2 HEX AAFF
0349 FFAA         TABLE3 HEX FFAA
034B D5FFFF       TABLE4 HEX D5FFFFD5AA
034E D5AA
```

Thus when told to erase green, it will erase all green, including the green part of any white that is made up of green and violet. This turns the white into violet. Of course, any white made up of blue and orange is left alone. So to erase white, simply erase the two colors that make it up. To avoid changing the white to another color, simply draw it in the colors that you do not plan to filter out later.

How the color filter works delves deeply into the mysteries of Apple color graphics. From what I have been able to deduce, it seems that each byte in the Hi-Res memory holds seven screen dots. Each set bit in the lower seven bits will turn on one dot. The highest bit determines whether the dots will be green and violet, or blue and orange. On even bytes, bits 0, 2, 4 and 6 create violet or blue while bits 1, 3 and 5 create green or orange. On odd bytes, this sequence is reversed. This is a very strange system but it seems to work. What the color filter does is mask out all of the bits in the Hi-Res memory area that would create a particular color. By changing all of these color bits to 0, it eliminates the color. The comments in the source program listing give more detail on how the program operates.

Two bytes of zero page memory are needed for the indirect addressing. The program uses bytes 6 and 7, but any two consecutive bytes can be used. As written, the program works only on Hi-Res page one, but by changing the values of LOSCRN to 40 and HISCRN to 60, you can make it work on Hi-Res page two. Finally, if you don't have an assembler, you can simply load the hexadecimal values listed in the table using the Apple monitor's data entry function.

I would like to offer one last note o[n] the Apple color graphics. The colors [I] have referred to here are the ones I ge[t] from my Apple on my television. Th[e] colors you get may be different. Th[e] best approach is to experiment with th[e] program on your system to see wha[t] number inputs erase what colors. Th[e] Applesoft BASIC demonstration pro[-]gram listed here should give you a goo[d] idea of how the color filter works o[n] your system.

```
5  REM  COLOR FILTER DEMO
10  HGR : HOME : VTAB 22
20  FOR I = 1 TO 7
30  HCOLOR= I
40  HPLOT 0,I * 10 TO 250,I * 10 + 50
50  NEXT I
55  FOR J = 1 TO 5000: NEXT J
60  FOR I = 1 TO 4
70  PRINT : PRINT : PRINT "COLOR FILTER INPUT: "I
80  POKE 769,I
90  CALL 768
100  FOR J = 1 TO 5000: NEXT J
110  NEXT I
120  TEXT
130  END
```

MICRO

Serial Line Editor for the Apple

This routine is an extended line editor for the Apple, which allows inserting, deleting, and several other features.

Wes Huntress
650 Chaparral Rd.
Sierra Madre, California 91024

GETLN is a machine language routine which can be used to replace the standard line input routine which resides in the monitor ROM in your Apple. It is called at one entry point or another by both Applesoft and Integer BASICs for line input. The advantage of the alternate routine given here is the editing features that it contains. The Apple monitor ESC editing features are very useful for editing BASIC program lines, but are not the best for editing text. The editing features in GETLN are illustrative of serial text line editing and could form the basis of any line-oriented text processing program. GETLN also allows the input of normally forbidden characters in Applesoft, such as the comma and colon. All of this is gained at a slight disadvantage in usage. Applesoft programs must be moved up two pages in memory and a few extra program steps are required instead of a simple INPUT statement. GETLN should be used only for string input and string editing. The version given here is for Applesoft. With a few changes it can be made to work for Integer as well.

When called, GETLN prompts for input and places the characters in the keyboard buffer at $200.2FF. All editing is done on the characters placed in the keyboard buffer. On return from GETLN it is necessary to move the characters from the keyboard buffer to the memory space that is to be occupied by the string. For Applesoft, this requires that the location in memory of the string variable's address pointer be

```
0800                 ;**********************
0800                 ;*                    *
0800                 ;* SERIAL LINE EDITOR *
0800                 ;*   FOR APPLESOFT    *
0800                 ;*                    *
0800                 ;*         BY         *
0800                 ;*                    *
0800                 ;*    WES HUNTRESS    *
0800                 ;*  SIERRA MADRE, CA  *
0800                 ;*   (213)-355-8125   *
0800                 ;*                    *
0800                 ;*      MAY 1980      *
0800                 ;*                    *
0800                 ;**********************
0800                 ;
0800                 ;EQUATES: CONSTANTS
0800                 ;
0800                 BS     EPZ $88
0800                 CR     EPZ $8D
0800                 CSM    EPZ $9D
0800                 CTL    EPZ $20
0800                 ESC    EPZ $9B
0800                 FIX    EPZ $7F
0800                 INV    EPZ $80
0800                 NAK    EPZ $95
0800                 BEND   EPZ $FE
0800                 ZERO   EPZ $00
0800                 BLANK  EPZ $A0
0800                 ;
0800                 ;EQUATES: POINTERS
0800                 ;
0800                 CHAR#  EPZ $19
0800                 EOL    EPZ $1A
0800                 STRT   EPZ $1B
0800                 TEMP   EPZ $1C
0800                 SUBSTR EPZ $1D
0800                 SUBEND EPZ $1E
0800                 MODE   EPZ $1F
0800                 ;
0800                 ;EQUATES: MONITOR ADDRESSES
0800                 ;
0800                 BUFFER EQU $0200
0800                 KEYIN  EQU $FD0C
0800                 PRINT  EQU $FDED
0800                 BACKSP EQU $FC10
0800                 ADVANC EQU $FBF4
0800                 RETURN EQU $FC62
0800                 CLREOP EQU $FC42
0800                 BELL   EQU $FF3A
0800                 ;
0800                        ORG $0800
0800                 ;
0800                 ;INITIALIZE KEYBOARD BUFFER
0800                 ;
0800 A0A0            GETLN  LDY #BLANK          ;LOAD BLANK CHARACTER
0802 8C0002          CLRB   STY BUFFER          ;STORE IT IN KEYBOARD BUFFER
0805 EE0308                 INC *-$2            ;FROM $0200
0808 D0F8                   BNE CLRB            ;TO $02FF
080A A200                   LDX #ZERO           ;SET POINTERS TO ZERO:
080C 8619                   STX CHAR#           ;CHARACTER NUMBER IN THE STRING
080E 861A                   STX EOL             ;END OF LINE POINTER
0810 861D                   STX SUBSTRT         ;SUBSTRING START POINTER
0812 861E                   STX SUBEND          ;SUBSTRING END POINTER
0814 861F                   STX MODE            ;MAINLINE/SUBSTRING MODE FLAG
0816                 ;
0816                 ;MAINLINE CHARACTER ENTRY ROUTINE
0816                 ;
0816 200CFD          GETCHR JSR KEYIN           ;GET CHAR USING MONITOR ROUTINE
0819 C988            GETCH1 CMP #BS             ;BACKSPACE?
081B F05B                   BEQ BKSPCE          ;YES, GOTO BACKSPACE ROUTINE
081D C99B                   CMP #ESC            ;ESCAPE KEY?
081F F031                   BEQ ESCAPE          ;YES, GOTO ESCAPE VECTOR ROUTINE
0821 C995                   CMP #NAK            ;FORWARD ARROW?
```

(continued)

known. The method used to accomplish this is the same as given in *CONTACT#6*. A dummy variable is declared as the first variable in the program, i.e. **X$** = " ", which assigns the two-byte variable name to the first two locations in memory at the LOMEM: pointer. The third location is assigned to the string length, and the fourth and fifth locations to the address of the string in memory, low byte first.

The LOMEM: pointer is at $69-70, so that the address of the string **X$** can now be found indirectly from the LOMEM: pointer. A separate machine language program is provided called GI which interfaces the GETLN routine with Applesoft programs by placing the address of the keyboard buffer, and the buffer string length, into the proper location for **X$** using the LOMEM: pointer.

The string **X$** is now assigned to the string in the keyboard buffer. In order to move it into the upper part of memory where Applesoft strings are normally stored, and to prevent the string from being clobbered the next time GETLN is called, the statement **X$** = MID$(X$,1) is used. This statement performs a memory move from the present location of **X$** (the keyboard buffer) to the next available space in high memory, and is the key to the success of the interface of GETLN with Applesoft programs.

How to Use It

To use GETLN with Applesoft programs, both GI and GETLN must be present in memory. To set up your program and call for input, use the following procedure:

```
5 X$ = " ":REM FIRST
  VARIABLE DECLARATION
.
.
.
100 CALL 834:A$ = MID$(X$,1):
    REM KEYBOARD INPUT
```

Line 100 replaces the INPUT A$ statement. CALL 834 is to the keyboard input entry point in the GI interface routine. Three other entry points are provided in the interface routine. The call

```
100 CALL 853:X$ = MID$(X$,1):
    REM DOS INPUT
```

replaces the INPUT A$ statement when READing text files from the disk. A separate routine from the keyboard

```
0823 F061           BEQ FORWRD         ;YES, GOTO FORWARD ARROW ROUTINE
0825 C98D           CMP #CR            ;RETURN?
0827 F063           BEQ LINEND         ;YES, GOTO EXIT ROUTINE
0829 A619           LDX CHAR#          ;NONE OF THESE, GET CURRENT CHAR#
082B 297F           AND #FIX           ;FIX NEG ASCII INPUT FOR APPLESOFT
082D 204508         JSR STRPNT         ;STORE AND PRINT CHAR
0830         ;
0830         ;POINTER UPDATING
0830         ;
0830 E619    FXPTRS INC CHAR#          ;INC POSITION-IN-STRING POINTER
0832 A619           LDX CHAR#          ;GET IT
0834 E41E           CPX SUBEND         ;AT END OF SUBSTRING OR BUFFER?
0836 F076           BEQ WHICH          ;YES, GO FIND OUT WHICH
0838 A41A           LDY EOL            ;GET END OF LINE POINTER
083A C419           CPY CHAR#          ;END OF CURRENT LINE?
083C B004           BCS FXPOUT         ;NO, SKIP EOL POINTER UPDATE
083E E61A           INC EOL            ;INCREMENT END OF LINE POINTER
0840 F05F           BEQ BUFULL         ;256 CHARS! GOTO BUFFER FULL
0842 4C1608  FXPOUT JMP GETCHR         ;DONE. GET ANOTHER CHARACTER
0845         ;
0845         ;STORE AND PRINT ROUTINE
0845         ;
0845 9D0002  STRPNT STA BUFFER,X       ;STORE IN CURRENT BUFFER LOC.
0848 C920           CMP #CTL           ;CONTROL CHARACTER?
084A 9002           BCC PNT            ;NO, SKIP TO PRINT
084C 0980           ORA #INV           ;YES, CONVERT TO INVERSE
084E 20EDFD  PNT    JSR PRINT          ;PRINT TO SCREEN
0851 60             RTS
0852         ;
0852         ;ESCAPE KEY VECTOR ROUTINE
0852         ;
0852 A41F    ESCAPE LDY MODE           ;SUBSTRING MODE?
0854 D048           BNE SBEXV          ;YES, GOTO SUBSTRING EXIT VECTOR
0856 200CFD         JSR KEYIN          ;GET ANOTHER CHARACTER
0859 C995           CMP #NAK           ;FORWARD ARROW?
085B F00F           BEQ INSV           ;YES, GOTO INSERT MODE VECTOR
085D C988           CMP #BS            ;BACKSPACE?
085F F011           BEQ DELV           ;YES, GOTO DELETE MODE VECTOR
0861 C9A0           CMP #BLANK         ;SPACE CHAR?
0863 F00A           BEQ ZMMV           ;YES, GOTO CURSOR ZOOM VECTOR
0865 C99D           CMP #CSM           ;CTRL-SHIFT-M?
0867 F00C           BEQ ZAPV           ;YES, GOTO LINE ZAP VECTOR
0869 4C7409         JMP CHRFND         ;NONE OF THESE, GOTO CHAR FIND
086C 4C0509  INSV   JMP INSERT         ;GOTO INSERT ROUTINE
086F 4C5509  ZMMV   JMP ZOOM           ;GOTO CURSOR ZOOM ROUTINE
0872 4CED08  DELV   JMP DELETE         ;GOTO DELETE ROUTINE
0875 4C9A09  ZAPV   JMP ZAP            ;GOTO DELETE-TO-EOL ROUTINE
0878         ;
0878         ;BACKSPACE ROUTINE
0878         ;
0878 A419    BKSPCE LDY CHAR#          ;GET POSITION IN LINE
087A C41D           CPY SUBSTRT        ;AT BEGINNING OF LINE/SUBSTRING?
087C F005           BEQ BSOUT          ;YES, RETURN
087E C619           DEC CHAR#          ;NO, DECREMENT POSITION IN LINE
0880 2010FC         JSR BACKSP         ;BACKSPACE CURSOR
0883 4C1608  BSOUT  JMP GETCHR         ;RETURN
0886         ;
0886         ;FORWARD ARROW ROUTINE
0886         ;
0886 20F4FB  FORWRD JSR ADVANC         ;ADVANCE CURSOR
0889 4C3008         JMP FXPTRS         ;RETURN TO INCREMENT CHAR#
088C         ;
088C         ;EXIT ROUTINE
088C         ;
088C A41F    LINEND LDY MODE           ;SUBSTRING MODE?
088E D00E           BNE SBEXV          ;YES, GOTO SUBSTRING EXIT
0890 A619           LDX CHAR#          ;STORE CHARACTER COUNT
0892 861A           STX EOL            ;IN EOL POINTER
0894 9D0002         STA BUFFER,X       ;STORE CR AT END OF STRING
0897 2042FC         JSR CLREOP         ;CLEAR SCREEN TO END OF PAGE
089A 2062FC         JSR RETURN         ;PERFORM CARRIAGE RETURN
089D 60             RTS                ;EXIT TO CALLER
089E 4C3D09  SBEXV  JMP SUBEXT         ;GOTO SUBSTRING EXIT
08A1         ;
08A1         ;BUFFER FULL ROUTINE
08A1         ;
08A1 C61A    BUFULL DEC EOL            ;DECREMENT EOL POINTER
08A3 C619    BUFUL1 DEC CHAR#          ;DECREMENT CURSOR POSITION
08A5 2010FC         JSR BACKSP         ;BACKSPACE
08A8 203AFF  BELEX  JSR BELL           ;SOUND BELL
08AB 4C1608         JMP GETCHR         ;RETURN
08AE         ;
08AE         ;DETERMINE MAINLINE OR SUBSTRING MODE
08AE         ;
08AE A41F    WHICH  LDY MODE           ;SUBSTRING MODE?
08B0 F0F1           BEQ BUFUL1         ;NO, GOTO BUFFER END ROUTINE
08B2 4C1709         JMP MOVEFD         ;YES, MOVE RIGHT STRING FORWARD
08B5         ;
08B5         ;MOVE STRING BACK ROUTINE
08B5         ;
08B5 A619    MOVEBK LDX CHAR#          ;GET DESTINATION START
08B7 A41B           LDY STRT           ;GET STRING START
08B9 A51A           LDA EOL            ;GET STRING END
08BB 38             SEC
08BC E51B           SBC STRT           ;SUBTRACT STRING START
08BE 18             CLC
08BF 6519           ADC CHAR#          ;ADD PRESENT CURSOR POSITION
08C1 851C           STA TEMP           ;STORE NEW EOL POINTER
08C3 B90002  MVBLP  LDA BUFFER,Y       ;GET STRING CHARACTER
```

```
08C6 204508          JSR STRPNT      ;STORE AND PRINT CHARACTER
08C9 C8              INY             ;INCREMENT THE
08CA E8              INX             ;POSITION POINTERS
08CB C41A            CPY EOL         ;END OF STRING?
08CD 90F4            BCC MVBLP       ;NO, GET ANOTHER CHARACTER
08CF 2042FC          JSR CLREOP      ;YES, CLEAR TO END OF PAGE
08D2 8A              TXA             ;STORE CURSOR POSITION
08D3 A8              TAY             ;IN Y REGISTER
08D4 A9A0            LDA #BLANK      ;GET SPACE CHARACTER
08D6 9D0002   CLRLP  STA BUFFER,X    ;STORE IN BUFFER BEYOND NEW EOL
08D9 E8              INX             ;INCREMENT POSITION
08DA E41A            CPX EOL         ;AT OLD END OF LINE?
08DC 90F8            BCC CLRLP       ;NO, DO IT AGAIN
08DE A61C            LDX TEMP        ;YES, GET NEW EOL
08E0 861A            STX EOL         ;STORE IT
08E2 98              TYA             ;GET CURSOR POSITION
08E3 AA              TAX             ;BACK INTO X REGISTER
08E4          ;
08E4          ;RESTORE CURSOR ROUTINE
08E4          ;
08E4 2010FC   RESTOR JSR BACKSP      ;BACKSPACE
08E7 CA              DEX             ;DECREMENT CURSOR POSITION
08E8 E419            CPX CHAR#       ;AT PRESENT CHARACTER POSITION?
08EA D0F8            BNE RESTOR      ;NO, DO IT AGAIN
08EC 60              RTS             ;YES, RETURN
08ED          ;
08ED          ;DELETE ROUTINE
08ED          ;
08ED A619     DELETE LDX CHAR#       ;GET PRESENT CHARACTER POSITION
08EF E8              INX             ;INCREMENT TO NEXT CHARACTER
08F0 861B            STX STRT        ;STORE STRING START POSITION
08F2 A41A     DELELP LDY EOL         ;GET END OF LINE POINTER
08F4 C419            CPY CHAR#       ;SAME AS NEXT CHARACTER POSITION?
08F6 F00A            BEQ DELOUT      ;YES, NOTHING TO DELETE!
08F8 20B508          JSR MOVEBK      ;NO, MOVE STRING BACK ONE SPACE
08FB 200CFD          JSR KEYIN       ;GET ANOTHER CHARACTER
08FE C988            CMP #BS         ;ANOTHER BACKSPACE CHARACTER?
0900 F0F0            BEQ DELELP      ;YES, DELETE ANOTHER CHARACTER
0902 4C1908   DELOUT JMP GETCH1      ;NO, BACK TO MAINLINE
0905          ;
0905          ;INSERT ROUTINE INITIALIZE
0905          ;
0905 A61A     INSERT LDX EOL         ;GET END OF LINE POINTER
0907 E0FE            CPX #BEND       ;END OF ALLOWABLE INSERTIONS?
0909 B09D            BCS BELEX       ;YES, STOP INPUT
090B A619            LDX CHAR#       ;NO, GET POSITION IN LINE
090D E41A            CPX EOL         ;AT END OF LINE?
090F F029            BEQ INOUT       ;YES, NO NEED TO INSERT!
0911 861D            STX SUBSTR      ;NO, STORE SUBSTRING START
0913 861E            STX SUBEND      ;STORE PRESENT SUBSTRING END
0915 851F            STA MODE        ;SET SUBSTRING MODE FLAG
0917          ;
0917          ;MOVE STRING FORWARD ROUTINE
0917          ;
0917 20F4FB   MOVEFD JSR ADVANC      ;ADVANCE CURSOR
091A BD0002          LDA BUFFER,X    ;GET FIRST STRING CHARACTER
091D E61A            INC EOL         ;INCREMENT EOL POINTER
091F F02E            BEQ SBOUT       ;BUFFER END! STOP INPUT
0921 E8       MVFLP  INX             ;POINT TO SECOND CHARACTER
0922 BC0002          LDY BUFFER,X    ;GET SECOND CHARACTER
0925 204508          JSR STRPNT      ;STORE AND PRINT FIRST CHAR
0928 98              TYA             ;TRANFER SECOND CHAR TO ACC.
0929 E41A            CPX EOL         ;END OF LINE?
092B D0F4            BNE MVFLP       ;NO, DO IT AGAIN
092D E8              INX             ;YES
092E 20E408          JSR RESTOR      ;RESTORE CURSOR
0931 98              TYA             ;GET SPACE CHAR INTO ACC.
0932 204508          JSR STRPNT      ;STORE & PRINT AT INSERT POSITION
0935 2010FC          JSR BACKSP      ;RETURN CURSOR TO INSERT POSITION
0938 E61E            INC SUBEND      ;INCREMENT SUBSTRING END POINTER
093A 4C1608   INOUT  JMP GETCHR      ;GET ANOTHER CHAR
093D          ;
093D          ;SUBSTRING EXIT ROUTINE
093D          ;
093D A61E     SUBEXT LDX SUBEND      ;GET SUBSTRING END POSITION
093F 861B            STX STRT        ;STORE IN STRING START POINTER
0941 20B508          JSR MOVEBK      ;MOVE RIGHT STRING BACK
0944 A200            LDX #ZERO       ;RESET THE
0946 861D            STX SUBSTR      ;SUBSTRING START,
0948 861E            STX SUBEND      ;SUBSTRING END POINTERS
094A 861F            STX MODE        ;AND MODE FLAG
094C 4C1608          JMP GETCHR      ;BACK TO MAINLINE
094F 2010FC   SBOUT  JSR BACKSP      ;BACKSPACE
0952 4CA108          JMP BUFULL      ;GOTO BUFFER FULL
0955          ;
0955          ;CURSOR ZOOM ROUTINE
0955          ;
0955 A51A     ZOOM   LDA EOL         ;GET EOL POINTER
0957 F00E            BEQ ZMOUT       ;NULL LINE! RETURN
0959 AA              TAX             ;STORE EOL IN X REGISTER
095A E519            SBC CHAR#       ;CURSOR AT END OF LINE?
095C F00C            BEQ ZBEG        ;YES, ZOOM TO LINE START
095E 8619            STX CHAR#       ;STORE CURSOR POSITION (EOL)
0960 AA              TAX             ;GET ADVANCE COUNT IN X REGISTER
0961 20F4FB   ZOOMLP JSR ADVANC      ;ADVANCE CURSOR
0964 CA              DEX             ;DECREMENT ADVANCE COUNT
0965 D0FA            BNE ZOOMLP      ;ADVANCE AGAIN IF NOT AT EOL
```

(continued)

input routine is required for Applesoft programs since the DOS stores and outputs all text files in negative ASCII. The call

```
100 X$=A$:CALL 800:REM
    PRINT
```

can be used in place of the PRINT A$ statement to print all control characters in inverse video. Otherwise use the PRINT A$ statement as usual. To recall a string for further editing, use

```
100 X$=A$:CALL 807:A$=
    MID$(X$,1):REM EDIT
```

The cursor will be placed on tne screen at the beginning of the recalled string. Dimensioned strings can be used as well as simple strings. GETLN can also be used alone from assembly language using 800G. It will place the input string in the keyboard buffer in standard ASCII terminated by $8D (CR).

GETLN occupies nearly two pages of memory from $800 to $9AF. Since Applesoft programs normally reside in this space, it is necessary to move your program up in memory to make room for GETLN. This is readily accomplished by two statements:

```
POKE 104,10:POKE 2560,0
```

This line must be executed either from immediate mode or from an EXEC file before loading the Applesoft program. The short interface routine occupies locations $300 to $355.

Editing Features

The following edit commands are implemented in GETLN. Except for the usual Apple ←, → and RETURN editing keys, all commands are initiated by hitting the ESC key.

→	Move cursor right, copy character
←	Move cursor left
RETURN	Terminate line, clear to end of page
ESC→	Initiate insert mode, ESC or RET to exit
ESC←	Delete character, recursive
ESC sp bar	Move cursor to beginning (end) of line
ESC char	Move cursor to first occurrence of char
ESC ctrl-shift-M	Delete remainder of line

The first three commands operate just as in the Apple monitor line editor. The monitor ESC functions are replaced with the five ESC functions listed above. Use ESC ——→ to insert characters at any place in the line. Use the usual monitor ——→ and ←—— keys to position the cursor over the character where you wish to insert. ESC ——→ will push right by one character the entire string beginning

```
*800.9CF

0800- A0 A0 8C 00 02 EE 03 08
0808- D0 F8 A2 00 86 19 86 1A
0810- 86 1D 86 1E 86 1F 20 0C
0818- FD C9 88 F0 5B C9 9B F0
0820- 31 C9 95 F0 61 C9 8D F0
0828- 63 A6 19 29 7F 20 45 08
0830- E6 19 A6 19 E4 1E F0 76
0838- A4 1A C4 19 B0 04 E6 1A
0840- F0 5F 4C 16 08 9D 00 02
0848- C9 20 90 02 09 80 20 ED
0850- FD 60 A4 1F D0 48 20 0C
0858- FD C9 95 F0 0F C9 88 F0
0860- 11 C9 A0 F0 0A C9 9D F0
0868- 0C 4C 74 09 4C 05 09 4C
0870- 55 09 4C ED 08 4C 9A 09
0878- A4 19 C4 1D F0 05 C6 19
0880- 20 10 FC 4C 16 08 20 F4
0888- FB 4C 30 08 A4 1F D0 0E
0890- A6 19 86 1A 9D 00 02 20
0898- 42 FC 20 62 FC 60 4C 3D
08A0- 09 C6 1A C6 19 20 10 FC
08A8- 20 3A FF 4C 16 08 A4 1F
08B0- F0 F1 4C 17 09 A6 19 A4
08B8- 1B A5 1A 38 E5 1B 18 65
08C0- 19 85 1C B9 00 02 20 45
08C8- 08 C8 E8 C4 1A 90 F4 20
08D0- 42 FC 8A A8 A9 A0 9D 00
08D8- 02 E8 E4 1A 90 F8 A6 1C
08E0- 86 1A 98 AA 20 10 FC CA
08E8- E4 19 D0 F8 60 A6 19 E8
08F0- 86 1B A4 1A C4 19 F0 0A
08F8- 20 B5 08 20 0C FD C9 88
0900- F0 F0 4C 19 08 A6 1A E0
0908- FE B0 9D A6 19 E4 1A F0
0910- 29 86 1D 86 1E 85 1F 20
0918- F4 FB BD 00 02 E6 1A F0
0920- 2E E8 BC 00 02 20 45 08
0928- 98 E4 1A D0 F4 E8 20 E4
0930- 08 98 20 45 08 20 10 FC
0938- E6 1E 4C 16 08 A6 1E 86
0940- 1B 20 B5 08 A2 00 86 1D
0948- 86 1E 86 1F 4C 16 08 20
0950- 10 FC 4C A1 08 A5 1A F0
0958- 0E AA E5 19 F0 0C 86 19
0960- AA 20 F4 FB CA D0 FA 4C
0968- 16 08 20 10 FC CA D0 FA
0970- 86 19 F0 F3 29 7F 85 1B
0978- A6 19 E8 20 F4 FB E4 19
0980- F0 0D E4 1A B0 0C BD 00
0988- 02 C5 1B D0 ED 86 19 4C
0990- 16 08 20 10 FC CA D0 FA
0998- F0 E4 A6 19 A9 A0 20 45
09A0- 08 E8 E4 1A 90 F8 20 E4
09A8- 08 4C 16 08 A2 FF E8 20
09B0- 0C FD 9D 00 02 C9 8D D0
09B8- F5 86 1A E8 BD FF 01 29
09C0- 7F 9D FF 01 CA D0 F5 A6
09C8- 1A 60 00 00 00 00 00 00
```

```
0967 4C1608   ZMOUT  JMP GETCHR        ;BACK TO MAINLINE
096A 2010FC   ZBEG   JSR BACKSP        ;BACKSPACE
096D CA              DEX               ;DECREMENT POSITION IN LINE
096E D0FA            BNE ZBEG          ;DO IT AGAIN IF NOT AT LINE START
0970 8619            STX CHAR#         ;STORE CURSOR POSITION
0972 F0F3            BEQ ZMOUT         ;BACK TO MAINLINE
0974          ;
0974          ;CHARACTER SEARCH ROUTINE
0974          ;
0974          ;
0974 297F     CHRFND AND #FIX          ;CONVERT NEG ASCII INPUT
0976 851B            STA STRT          ;STORE KEY CHARACTER
0978 A619            LDX CHAR#         ;GET PRESENT CURSOR POSITION
097A E8       CHRFLP INX               ;INCREMENT CURSOR POINTER
097B 20F4FB          JSR ADVANC        ;ADVANCE CURSOR
097E E419     CHRF1  CPX CHAR#         ;AT OLD CURSOR POSITION?
0980 F00D            BEQ CHFOUT        ;YES, CHARACTER NOT FOUND
0982 E41A            CPX EOL           ;END OF LINE?
0984 B00C            BCS SBEG          ;YES, START AGAIN AT LINE START
0986 BD0002          LDA BUFFER,X      ;GET CHARACTER AT THIS POSITION
0989 C51B            CMP STRT          ;SAME AS KEY?
098B D0ED            BNE CHRFLP        ;NO, TRY AGAIN
098D 8619            STX CHAR#         ;YES, STORE CURSOR POSITION
098F 4C1608   CHFOUT JMP GETCHR        ;BACK TO MAINLINE
0992 2010FC   SBEG   JSR BACKSP        ;BACKSPACE
0995 CA              DEX               ;BEGINNING OF LINE?
0996 D0FA            BNE SBEG          ;NO, BACKSPACE AGAIN
0998 F0E4            BEQ CHRF1         ;YES, CONTINUE SEARCH
099A          ;
099A          ;ZAP (DELETE TO END OF LINE) ROUTINE
099A          ;
099A A619     ZAP    LDX CHAR#         ;GET CURSOR POSITION
099C A9A0            LDA #BLANK        ;LOAD ACC. WITH SPACE CHAR
099E 204508   ZAPLP  JSR STRPNT        ;STORE AND PRINT IT
09A1 E8              INX               ;NEXT POSITION
09A2 E41A            CPX EOL           ;END OF LINE?
09A4 90F8            BCC ZAPLP         ;NO, DO IT AGAIN
09A6 20E408          JSR RESTOR        ;YES, RESTORE CURSOR
09A9 4C1608          JMP GETCHR        ;BACK TO MAINLINE
09AC          ;
09AC          ;DISK INPUT ROUTINE
09AC          ;
09AC A2FF     DISKIN LDX #ZERO-$1      ;INITIATE THE
09AE E8       DISKL1 INX               ;CHAR# POINTER
09AF 200CFD          JSR KEYIN         ;GET A CHARACTER
09B2 9D0002          STA BUFFER,X      ;STORE IN BUFFER
09B5 C98D            CMP #CR           ;CARRIAGE RETURN?
09B7 D0F5            BNE DISKL1        ;NO, GET ANOTHER CHARACTER
09B9 861A            STX EOL           ;YES, STORE CHARACTER COUNT
09BB E8              INX               ;INIT FOR ASCII CONVERSION
09BC BDFF01   DISKL2 LDA BUFFER-$1,X   ;GET BUFFER CHARACTER
09BF 297F            AND #FIX          ;CONVERT FOR APPLESOFT
09C1 9DFF01          STA BUFFER-$1,X   ;PUT IT BACK
09C4 CA              DEX               ;COUNT BACK TO ZERO
09C5 D0F5            BNE DISKL2        ;LOOP IF NOT FINISHED
09C7 A61A            LDX EOL           ;CHAR COUNT IN X REG.
09C9 60              RTS               ;EXIT TO CALLER
```

```
0800          ;**********************
0800          ;*                    *
0800          ;*   INTERFACE CODE   *
0800          ;*     FP - GETLN     *
0800          ;*                    *
0800          ;*         BY         *
0800          ;*                    *
0800          ;*    WES HUNTRESS    *
0800          ;*  SIERRA MADRE, CA  *
0800          ;*   (213)-355-8125   *
0800          ;*                    *
0800          ;*      MAY 1980      *
0800          ;*                    *
0800          ;**********************
0800          ;
0800          ;EQUATES: CONSTANTS & ZERO PAGE
0800          ;
0800          CURS   EPZ $19
0800          ZERO   EPZ $00
0800          BLANK  EPZ $A0
0800          LENLOC EPZ $02
0800          STADRL EPZ $08
0800          STADRH EPZ $09
0800          STRLEN EPZ $1A
0800          VARPTR EPZ $69
0800          ;
0800          ;EQUATES: BUFFER & ADDRESSES
0800          ;
0800          BUFFER EQU $0200
0800          GETLN  EQU $0800
0800          EENTRY EQU $0810
```

```
0800              STRPNT EQU $0845
0800              DISKIN EQU $09AC
0800              BACKSP EQU $FC10
0800              RETURN EQU $FC62
0800              ;
0300                     ORG $0300
0300              ;
0300              ;PRINT X$ SUBROUTINE
0300              ;
0300 A002         PSCRN  LDY #LENLOC
0302 B169                LDA (VARPTR),Y      ;GET X$ STRING LENGTH
0304 851A                STA STRLEN          ;STORE STRING LENGTH PTR
0306 C8                  INY
0307 B169                LDA (VARPTR),Y      ;GET X$ ADDR LOW BYTE
0309 8508                STA STADRL          ;STORE IN X$ ADDR PTR LOW
030B C8                  INY
030C B169                LDA (VARPTR),Y      ;GET X$ ADDR HI BYTE
030E 8509                STA STADRH          ;STORE IN X$ ADDR PTR HI
0310 A000                LDY #ZERO           ;INITIATE THE
0312 A200                LDX #ZERO           ;  COUNTERS
0314 B108         PNTLP  LDA (STADRL),Y      ;GET MID$(X$,Y,1)
0316 204508              JSR STRPNT          ;STORE & PRINT
0319 E8                  INX                 ;INCREMENT
031A C8                  INY                 ;  COUNTERS
031B C41A                CPY STRLEN          ;END OF STRING?
031D 90F5                BCC PNTLP           ;NO, GET ANOTHER CHAR
031F 60                  RTS                 ;EXIT TO CALLER
0320              ;
0320              ;PRINT X$ TO SCREEN
0320              ;
0320 200003       PRINT  JSR PSCRN           ;PRINT X$
0323 2062FC              JSR RETURN          ;DO A CARRIAGE RETURN
0326 60                  RTS                 ;EXIT TO CALLER
0327              ;
0327              ;EDIT X$
0327              ;
0327 200003       EDIT   JSR PSCRN           ;PRINT X$
032A A9A0                LDA #BLANK          ;PUT SPACE CHAR
032C 9D0002       EDLP1  STA BUFFER,X        ;  INTO REMAINING
032F E8                  INX                 ;  BUFFER SPACE
0330 D0FA                BNE EDLP1
0332 2010FC       EDLP2  JSR BACKSP          ;RESTORE CURSOR
0335 88                  DEY                 ;  TO LINE START
0336 D0FA                BNE EDLP2
0338 A200                LDX #ZERO           ;STORE CURSOR
033A 8619                STX CURS            ;  POSITION
033C 201008              JSR EENTRY          ;GETLN EDIT ENTRY
033F 4C4503              JMP TOX$            ;PUT IN X$
0342              ;
0342              ;X$ KEYBOARD INPUT
0342              ;
0342 200008       KYBIN  JSR GETLN           ;GET A LINE
0345 A002         TOX$   LDY #LENLOC         ;TRANSFER STRING
0347 8A                  TXA                 ;  LENGTH FROM AFC,
0348 9169                STA (VARPTR),Y      ;  TO X$
034A C8                  INY
034B A900                LDA #ZERO           ;STORE
034D 9169                STA (VARPTR),Y      ;  KEYBOARD
034F C8                  INY                 ;  BUFFER
0350 A902                LDA #LENLOC         ;  ANDRESS
0352 9169                STA (VARPTR),Y      ;  INTO X$
0354 60                  RTS                 ;EXIT TO CALLFR
0355              ;
0355              ;X$ DOS INPUT
0355              ;
0355 20AC09       DOSIN  JSR DISKIN          ;GETLN DOS INPUT ENTRY
0358 4C4503              JMP TOX$            ;PUT INPUT IN X$


*300.35F

0300-  A0 02 B1 69 85 1A C8 B1
0308-  69 85 08 C8 B1 69 85 09
0310-  A0 00 A2 00 B1 08 20 45
0318-  08 E8 C8 C4 1A 90 F5 60
0320-  20 00 03 20 62 FC 60 20
0328-  00 03 A9 A0 9D 00 02 E8
0330-  D0 FA 20 10 FC 88 D0 FA
0338-  A2 00 86 19 20 10 08 4C
0340-  45 03 20 00 08 A0 02 8A
0348-  91 69 C8 A9 00 91 69 C8
0350-  A9 02 91 69 60 20 AC 09
0358-  4C 45 03 00 00 00 00 00
```

from the character under the cursor to the end of the line, leaving a blank under the cursor. As you type in new characters, the old right-hand string is continuously shifted right. The ← and → keys work on the inserted substring as before but will not allow editing left of the first inserted character. In the insert mode, → operates just like the space bar if keyed at the right-hand end of the substring. To terminate the insert mode, press ESC or RETURN. The old right-hand string is moved back one space for reconnection.

The ESC ← command deletes the character under the cursor and pulls left the entire string to the right of the cursor. The function is recursive, so that characters can continue to be deleted by repeated keying of the ← key. The first key pressed other than ← terminates the function.

The ESC space bar command moves the cursor to the end of the line. If the cursor is already at the end of the line, then it is moved to the beginning. This function allows rapid transport of the cursor to the beginning or end of the line.

The ESC char command moves the cursor right in the line to the first occurrence of the character key pressed after the escape key. If the character is not found before the end of the line, then the search branches to the beginning of the line. If the character is not found in the line, then the cursor is not moved.

The ESC ctrl-shift-M command deletes the entire line to the right of the cursor including the character under the cursor. This function allows excess garbage to be cleared from the line for editing readability.

Together these functions give you an intriguing and powerful text line editor. It's much more fun than the Apple monitor line input routine. Try it! You'll like it!

MICRO

Improved KIM Communication Capabilities

Add new I/O capabilities to your KIM with this software/hardware combination.

Ralph Tenny
P.O. Box 545
Richardson, Texas 75080

Code and Text Transfer

Unless he has a programmer, the small system owner often wonders how to program EPROMs for his system. Or, if he locates a friend with a programmer on his system, he then must figure out how to develop the program code on the KIM, test it, and then get the code into the system with the programmer. It is extremely likely that any scheme involving re-entry of the code in the second system will introduce errors, so it is desirable that the KIM produce a copy of its own code in a form usable by the second system.

First you need a program which puts out the exact memory image of the developed and debugged program. KIMOUT is such a program, which uses a second RS-232 port added to KIM. The reason that KIM's serial port is not suitable (in many cases) is that the KIM port has a hardware echo built in. Also, in some cases, the I/O lines driving KIM's serial port are disturbed by the operating system. Thus, a second port (described later) allows you to have an unrestricted and undisturbed, echo-free serial I/O port which won't ruffle the feathers of any other computer system it may be talking to.

The chief difference between KIMOUT and any other memory dump program is that KIMOUT does no data formatting, and inserts no characters which are not part of the memory image desired in EPROM. The software shown uses the second serial I/O program which was adapted from KIM's software to drive the second serial port. All the "new" software is part of an additional 2K of EPROM added to KIM and located at $C000_{16}$ through $C7FF_{16}$. However, these routines have been located beginning at 0200 and 0300 by making the appropriate changes in addresses.

Once the program to be ROMmed is ready, KIMOUT is given the starting and ending addresses of the program as follows:

	Start	End
Address Low	0002	0004
Address High	0003	0005

Table 1

Baud Rate	17F2 CNTL 30	17F3 CNTH 30
110		
300	E8	00
1200	35	00

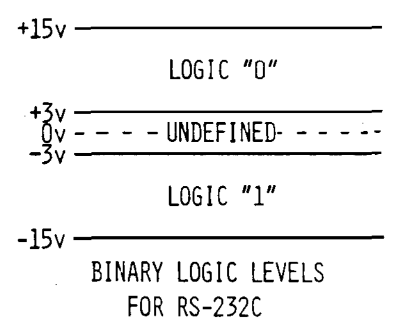


Figure 1: **RS-232 signals have a voltage "deadband" between +3V and −3V to increase the noise immunity of the equipment.**

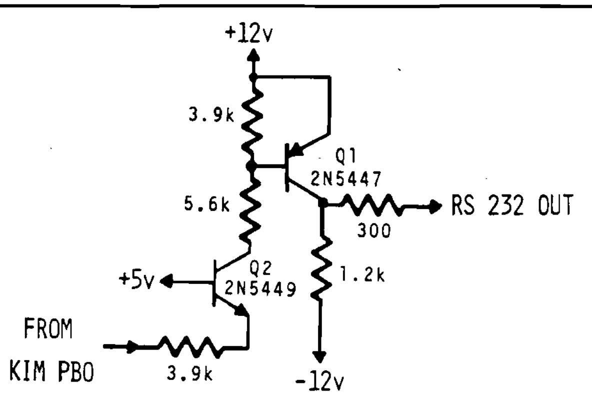


Figure 2: **Two transistors and some resistors make a very simple RS-232 output port to supplement KIM's current loop serial port.**

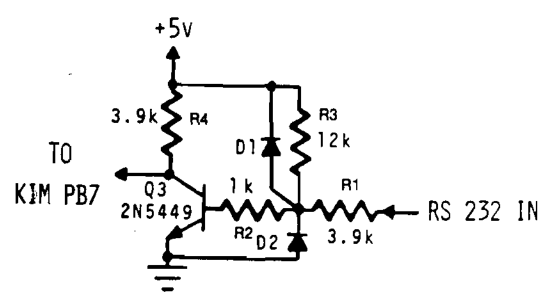


Figure 3: **One transistor buffered by resistors and diodes makes an RS-232 input port with protection against unexpected voltages.**

Set the timing constants in CNTL30 and CNTH30 (17F2, 17F3) for the proper data rate (see table 1), connect the two computers, start the receiving program in the other computer, then start KIMOUT. When KIMOUT has finished, it will re-light the KIM display, and you can terminate the receiving program.

In my case, the receiving program was in a TM990/189 (TI's University Board), which uses only 300 baud. Once the data has been transferred, I check starting and ending bytes, and a few representative other locations in the '189 memory, then dump the data to audio tape. (The TM990/189 will make a digital tape if a Model 733 TI terminal is available.) The '189 at work can read this audio tape and there is a programmer attached to it. About five minutes after dumping the tape, I have another EPROM for KIM!

It should be noted that some EPROM programmers (and some computers) will require that data handled in this manner be formatted into blocks with checksums. The tapes themselves use TI's tag loader format, so the actual transfer between the two University Boards is protected by checksums. So far, I have never encountered an error introduced by the process described, so maybe I've been lucky!

The program called TRANSLATE contains three smaller programs which cooperate in another type of data transfer. The Radio Shack TRS-80C™ computer has a 600 baud printer port, and the software issues only carriage returns instead of the CRLF pair issued by KIM and many other computers at the end of a line. I had no access to any 600 baud printers, and even my CRT terminal needed the line feed to present a picture of the TRS-80C output. So, the first section of TRANSLATE (SETUP) beginning at 0200 will read code or text from memory and add a line feed to any carriage return found.

The second section of TRANSLATE (RCV) beginning at 0238 will receive any continuous string of ASCII characters and place the characters in contiguous memory locations as long as there is memory left. If the string over-writes the end of the buffer (on KIM, the available buffer is 03E0-13FF), it quits listening and bounces back to the KIM monitor. Finally, the third section of TRANSLATE (CLEAR) clears memory beginning at the address specified in 0002 and 0003 (the same buffer is used for all sections of TRANSLATE) and extending through 13FF.

```
0800                ;**************************
0800                ;*                        *
0800                ;* COMMUNICATIONS SUPPORT *
0800                ;*                        *
0800                ;*      BY RALPH TENNY    *
0800                ;*                        *
0800                ;**************************
0800                ;*
0800                PBD     EQU $1702
0800                PBDD    EQU $1703
0800                CNTL30  EQU $17F2
0800                CNTH30  EQU $17F3
0800                TIMH    EQU $17F4
0800                START   EQU $1C4F
0800                CRLF    EQU $1E2F
0800                INITS   EQU $1E88
0800                OUTCH   EQU $1EA0
0800                PACK    EQU $1FAC
0800                TOP     EQU $1FD5
0800                ;
0800                SAL     EPZ $02
0800                SAH     EPZ $03
0800                EAL     EPZ $04
0800                EAH     EPZ $05
0800                YTMP    EPZ $20
0800                TMPY    EPZ $EE
0800                INL     EPZ $F8
0800                TEMP    EPZ $FC
0800                TMPX    EPZ $FD
0800                CHAR    EPZ $FE
0800                ;
0200                        ORG $200
0200                        OBJ $800
0200                ;
0200                ;***  TRANSLATE   ***
0200                ;
0200                ;THIS PROGRAM RECEIVES A HEX ASCII TEXT STRING
0200                ;OVER KIM'S STANDARD SERIAL PORT AND STORES
0200                ;THE STRING IN CONTIGUOUS MEMORY LOCATIONS.
0200                ;THIS SAME TEXT STRING CAN THEN BE OUPUT TO A
0200                ;PRINTER OR OTHER RS 232 DEVICE FOR DISPLAY.
0200                ;THIS ALLOWS KIM TO RECEIVE FROM A CPU WHICH
0200                ;HAS NO BAUD RATE SELECTION, AND TO OUTPUT
0200                ;TO A PRINTER AT ANY BAUD RATE DESIRED.
0200                ;
0200                ;THIS SECTION READS MEMORY, RESETS THE PRINTER
0200                ;(CARRIAGE RETURN-LINE FEED) IF THE CHARACTER
0200                ;IS A CARRIAGE RETURN ($0D), AND OUTPUTS
0200                ;ALL OTHER CHARACTERS.
0200                ;
0200 20881E         SETUP   JSR INITS           ;SET UP KIM STANDARD PORTS
0203 A000           INDX    LDY #$00            ;INITIALIZE Y INDEX
0205 8420                   STY YTMP            ;AND POINTER REGISTER
0207 A420           OUT     LDY YTMP            ;PICK UP POINTER VALUE
0209 B102                   LDA (SAL),Y         ; AND INDEX INTO TEXT BUFFER.
020B C90D                   CMP #$0D            ;IS IT A CARRIAGE RETURN?
020D F011                   BEQ RESET           ;IF SO, RESET THE PRINTER.
020F 20A01E                 JSR OUTCH           ;OTHERWISE, OUTPUT CHARACTER.
0212 E620                   INC YTMP            ;THEN BUMP THE POINTER.
0214 D007                   BNE MORE            ;TEST FOR END-OF-MEMORY PAGE.
0216 18                     CLC                 ;IF SO, PREPARE TO ADD
0217 A503                   LDA SAH             ;GET PAGE POINTER
0219 6901                   ADC #$01            ;AND INCREMENT IT
021B 8503                   STA SAH             ;RESTORE PAGE POINTER
021D 4C0702         MORE    JMP OUT             ;AND KEEP TRUCKIN'
0220 202F1E         RESET   JSR CRLF            ;RESET THE PRINTER
0223 A520                   LDA YTMP            ;GET THE POINTER
0225 38                     SEC                 ;FORCE A CARRY
0226 6502                   ADC SAL             ;TO BUMPT LO BYTE OF ADDRESS
0228 8502                   STA SAL             ;AND RESTORE ADDRESS
022A A503                   LDA SAH             ;GET THE HI BYTE
022C 6900                   ADC #$00            ;ADD IN POSSIBLE CARRY
022E 8503                   STA SAH             ;AND PUT HI BYTE BACK
0230 C914                   CMP #$14            ;END OF MEMORY?
0232 D0CF                   BNE INDX            ;IF NOT, MOVE ON OUT
0234 4C4F1C                 JMP START           ;OTHERWISE, RETURN TO KIM
0237 00                     BRK
0238                ;
0238                ;THIS SECTION RECEIVES INCOMING HEX ASCII
```

```
0238                ;CHARACTERS AND STORES THEM IN MEMORY LOCATIONS
0238                ;DEFINED IN $02 AND $03.
0238                ;
0238 205103   RCV    JSR INIT          ;INITIALIZE SECOND PORT
023B A000            LDY #$00          ;SET Y TO ZERO
023D 8420            STY YTMP          ;ALONG WITH POINTER REGISTER
023F 201F03   IN     JSR GETCHP        ;READ SECOND PORT
0242 C902            CMP #$02          ;VALID CHARACTER?
0244 30F9            BMI IN            ;IF NOT, KEEP TRYING
0246 A420            LDY YTMP          ;PUT POINTER IN Y REGISTER
0248 9102            STA (SAL),Y       ;AND DEPOSIT THE BYTE
024A E620            INC YTMP          ;BUMP THE POINTER,
024C D0F1            BNE IN            ;TEST FOR MEMORY PAGE END
024E A503            LDA SAH           ;IF SO, GET PAGE POINTER
0250 18              CLC               ;PREPARE FOR ADD
0251 6901            ADC #$01          ;INCREMENT PAGE POINTER
0253 8503            STA SAH           ;AND PUT IT BACK
0255 C914            CMP #$14          ;TEST FOR MEMORY END
0257 D0E6            BNE IN            ;IF NOT, GO GET MORE DATA
0259 4C4F1C          JMP START         ;OTHERWISE, RETURN TO KIM
025C                ;
025C                ;THIS SECTION CLEARS A MEMORY BUFFER BY WRITING
025C                ;$00 IN EACH LOCATION
025C                ;
025C A000     CLEAR  LDY #$00          ;CLEAR INDEX POINTER
025E 98              TYA               ;AND THE ACCUMULATOR
025F 9102     WRITE  STA (SAL),Y       ;CLEAR MEMORY BUFFER
0261 E602            INC SAL           ;BUMP THE INDEX
0263 D0FA            BNE WRITE         ;TEST FOR MEMORY PAGE END
0265 A503            LDA SAH           ;IF SO, GET PAGE POINTER
0267 18              CLC               ;PREPARE TO ADD
0268 6901            ADC #$01          ;ONE TO PAGE POINTER
026A 8503            STA SAH           ;AND PUT IT BACK.
026C C914            CMP #$14          ;END OF MEMORY?
026E D0EC            BNE CLEAR         ;IF NOT, CLEAR MORE MEMORY
0270 4C4F1C          JMP START         ;OTHERWISE, RETURN TO KIM
0273                ;
0273                ;KIMOUT
0273                ;
0273                ;THIS PROGRAM UTILIZES A SECOND RS-232 PORT ON
0273                ;KIM TO OUTPUT A CONTINUOUS DATA STREAM
0273                ;(USUALLY TEXT OR PROGRAM DATA) TO AN EPROM
0273                ;PROGRAMMER OR PRINTER.
0273                ;
0280                 ORG SETUP+$80
0280                 OBJ $880
0280                ;
0280 205103   STRT   JSR INIT          ;SET UP POINTER STORAGE
0283 A900     ZERO   LDA #$00          ;SET INITIAL POINTER VALUE
0285 8520            STA YTMP          ;IN A SAFE LOCATION
0287 A420     GET    LDY YTMP          ;LOAD POINTER INTO INDEX
0289 B102            LDA (SAL),Y       ;GET A BYTE OF DATA
028B 200003          JSR PRTBYT        ;AND OUTPUT IT
028E E620            INC YTMP          ;BUMP THE POINTER
0290 18              CLC               ;PREPARE TO ADD
0291 A502            LDA SAL           ;LO BYTE START ADDRESS
0293 6520            ADC YTMP          ;TO THE POINTER
0295 8502            STA SAL           ;FOR NEW START ADDRESS
0297 A503            LDA SAH           ;GET HI BYTE
0299 6900            ADC #$00          ;ADD IN POSSIBLE CARRY
029B 8503            STA SAH           ;AND RESTORE HI BYTE
029D A502            LDA SAL           ;GET LO BYTE
029F C504            CMP EAL           ;AND COMPARE TO END LO BYTE
02A1 9008            BCC NEXT          ;IF NOT, GO MOVE MORE DATA
02A3 A503            LDA SAH           ;OTHERWISE, CHECK HI BYTE
02A5 C505            CMP EAH           ;AGAINST END HI BYTE
02A7 F005            BEQ OUTK          ;IF EQUAL,
02A9 1003            BPL OUTK          ;OR BIGGER, STOP
02AB 4C8002   NEXT   JMP STRT          ;OTHERWISE DO MORE
02AE 4C4F1C   OUTK   JMP START         ;DONE, EXIT TO KIM
02B1 00              BRK
02B2                ;
02B2                ;SERIAL I/O
02B2                ;
02B2                ;THIS PROGRAM IS A SLIGHTLY MODIFIED COPY OF
02B2                ;PORTIONS OF THE KIM-1 MONITOR FUNCTIONS;
02B2                ;WITH THE EXCEPTION OF INIT, THE LABELS HAVE BEEN
02B2                ;PRESERVED.  THE MODIFICATIONS ACCOMODATE THE USE
02B2                ;OF A SEPARATE RS-232 SERIAL PORT, IMPLEMENTED IN
02B2                ;CONJUNCTION WITH THE APPLICATIONS I/O PORT OF KIM.
```

(continued)

TRANSLATE has made it possible for me to "translate" the Radio Shack computer output from 600 baud to 300 baud for a borrowed printer. Both TRANSLATE and KIMOUT will handle any type of computer data, because they deal with exact memory images of the data. I can even generate text such as this on KIM and bring it to this word processor for final editing, formatting and printing on a daisy-wheel printer!

Add A Second RS-232 Port

One problem with the KIM port is that it has a hardware echo built in which is inappropriate in some applications. Also, since the software is all in ROM, it is impossible to modify. These problems may be simply solved by creating a second RS-232 port.

The 20 mA loop port on the KIM-1 can be converted to an RS-232 port by adding some transistors to shift the input/output levels to match RS-232 specifications. Figure 1 details the voltage levels which make up the RS-232 specification. Some RS-232 peripheral devices will work with a smaller voltage swing or other deviations from the spec, but to be sure, build the simple circuits shown in figures 2 and 3.

Figure 2 shows the output circuit. This port will swing to full RS-232 levels and should meet all drive requirements for almost any imaginable peripheral device. Q1 is the output switch, while Q2 is a non-inverting level converter which allows the full ±12v RS-232 swing from Q1, without requiring an open-collector stage on the port line or the UART.

The problem of matching RS-232 input levels to another port pin is solved by the circuit shown in figure 3. A single transistor with input protection can accept ±12v swings and convert them to a level KIM is happy with. R1, D1 and D2 form a protective network for the transistor base. Also R1 with R2 provides adequate input impedance for the incoming signal. R3 is a pull-up to hold the port's input line at a spacing level (logic 0) when there is no input signal.

The KIM provides the basic software UART routines. The routines (PRTBYT, GETCH, OUTSP, OUTCH, and CRLF), use bit PB0 of the KIM Control Port to drive the output, and incoming data is read on PA7. We can do about the same thing, using PB0 of the Application Port for an output and PB7 for input. With those pin

```
02B2                  ;IN ADDITION, THE Y REGISTER OF THE 6502 HAS BEEN
02B2                  ;SAVED WHERE APPROPRIATE.
02B2                  ;
0300                         ORG $300
0300                         OBJ $900
0300                  ;
0300 85FC             PRTBYT STA TEMP          ;SAVE ACCUMULATOR
0302 4A                      LSR               ;SHIFT OFF LOW NIBBLE
0303 4A                      LSR               ;TO ACCESS
0304 4A                      LSR               ;THE HIGH ORDER
0305 4A                      LSR               ;NIBBLE FOR OUTPUT
0306 201103                  JSR HEXTA         ;CONVERT TO HEX AND OUTPUT
0309 A5FC                    LDA TEMP          ;GET OTHER HALF
030B 201103                  JSR HEXTA         ;CONVERT TO HEX AND OUTPUT
030E A5FC                    LDA TEMP          ;RESTORE BYTE IN A
0310 60                      RTS               ;AND RETURN
0311 290F             HEXTA  AND #$0F          ;MASK OFF HI NIBBLE
0313 C90A                    CMP #$0A          ;TEST FOR ALPHA
0315 18                      CLC               ;PREPARE TO ADD
0316 3002                    BMI HEXTA1        ;NOT ALPHA
0318 6907                    ADC #$07          ;ALPHA, ADD MORE
031A 6930             HEXTA1 ADC #$30          ;FIX NON-ALPHA
031C 20A01E                  JSR OUTCH         ;OUTPUT IT
031F 86FD             GETCHP STX TMPX          ;SAVE X REG
0321 84EE                    STY TMPY          ;AND Y REG
0323 A208                    LDX #$08          ;COUNT OF 8 BITS
0325 A901                    LDA #$01          ;MASK IN ACCUMULATOR
0327 2C0217           GET1   BIT PBD           ;TEST FOR START BIT
032A EA                      NOP
032B EA                      NOP
032C 30F9                    BMI GET1          ;KEEP TRYING
032E 209303                  JSR DELAY         ;DELAY ONE BIT
0331 20AA03           GET5   JSR DEHALF        ;DELAY 1/2 BIT
0334 AD0217           GET2   LDA PBD           ;GET 8 BITS
0337 2980                    AND #$80          ;MASK OFF LOW ORDER BITS
0339 46FE                    LSR CHAR          ;SHIFT CHARACTER RIGHT
033B 05FE                    ORA CHAR          ;OR IN RECEIVED BIT
033D 85FE                    STA CHAR          ;AND RESTORE CHAR
033F 209303                  JSR DELAY         ;DELAY ONE BIT TIME
0342 CA                      DEX               ;AND COUNT BIT
0343 D0EF                    BNE GET2          ;REPEAT UNTIL 8 BITS IN
0345 20AA03                  JSR DEHALF        ;THEN, DELAY 1/2 BIT
0348 A4EE                    LDY TMPY          ;RETRIEVE Y
034A A6FD                    LDX TMPX          ;AND X
034C A5FE                    LDA CHAR          ;GET THE CHARACTER
034E 2A                      ROL               ;AND SHIFT OFF THE
034F 4A                      LSR               ;PARITY BIT, THEN
0350 60                      RTS               ;RETURN
0351 A201             INIT   LDX #$01          ;TURN ON ONE BIT
0353 8E0317                  STX PBDD          ;IN THE USER PORT
0356 D8                      CLD               ;SET UP BINARY MODE
0357 78                      SEI               ;INHIBIT INTERRUPTS
0358 60                      RTS               ;AND RETURN
0359 A920             OUTSP  LDA #$20          ;ASCII SPACE
035B 85FE             OUTCHA STA CHAR          ;SAVE THE CHARACTER
035D 84EE                    STY TMPY          ;THE Y REG,
035F 86FD                    STX TMPX          ;AND X REG
0361 209303                  JSR DELAY         ;ONE BIT DELAY
0364 AD0217                  LDA PBD           ;READ THE PORT
0367 29FE                    AND #$FE          ;SET THE START BIT
0369 8D0217                  STA PBD           ;OUTPUT THE BIT
036C 209303                  JSR DELAY         ;WAIT ONE BIT TIME
036F A208                    LDX #$08          ;EIGHT BIT COUNT
0371 AD0217           OUT1   LDA PBD           ;GET THE OUTPUT BIT
0374 29FE                    AND #$FE          ;MASK START BIT
0376 46FE                    LSR CHAR          ;SHIFT BIT OUT OF CHAR
0378 6900                    ADC #$00          ;ADD IN CARRY BIT
037A 8D0217                  STA PBD           ;AND OUTPUT IT
037D 209303                  JSR DELAY         ;WAIT ONE BIT TIME
0380 CA                      DEX               ;COUNT THE BIT
0381 D0EE                    BNE OUT1          ;NOT DONE, GO BACK
0383 AD0217                  LDA PBD           ;LOAD THE OUTPUT BIT
0386 0901                    ORA #$01          ;SET IT HGH
0388 8D0217                  STA PBD           ;TO OUTPUT STOP BIT
038B 209303                  JSR DELAY         ;AND WAIT AGAIN
038E A6FD                    LDX TMPX          ;REMEMBER X
0390 A4EE                    LDY TMPY          ;AND Y
0392 60                      RTS               ;AND RETURN
0393 ADF317           DELAY  LDA CNTH30        ;GET HI BYTE DELAY COUNT
```

```
0396 8DF417          STA TIMH         ;STUFF IT IN THE TIMER
0399 ADF217          LDA CNTL30       ;AND GET THE LO BYTE
039C 38       DE2    SEC              ;SET CARRY FOR SUBTRACT
039D E901     DE4    SBC #$01         ;DECREMENT LO BYTE
039F B003            BCS DE3          ;BRANCH IF NO BORROW
03A1 CEF417          DEC TIMH         ;DECREMENT TIMER VALUE
03A4 ACF417   DE3    LDY TIMH         ;AND STUFF IT IN Y
03A7 10F3            BPL DE2          ;RETURN IF NOT NEGATIVE
03A9 60              RTS              ;OTHERWISE, RETURN
03AA ADF317   DEHALF LDA CNTH30       ;DELAY 1/2 BIT TIME
03AD 8DF417          STA TIMH         ;BY DOING A DOUBLE
03B0 ADF217          LDA CNTL30       ;RIGHT SHIFT OF
03B3 4A              LSR              ;THE COUNT VALUES
03B4 4EF417          LSR TIMH         ;AND THEN
03B7 90E3            BCC DE2          ;COUNTING THEM DOWN
03B9 0980            ORA #$80         ;FORCE A NEGATIVE
03BB B0E0            BCS DE4          ;TO FORCE A BRANCH.
03BD 00              BRK              ;BLOCK SEPARATOR
03BE 201F03   GETBYT JSR GETCHP       ;GO GET A CHARACTER
03C1 20AC1F          JSR PACK         ;MAKE IT A NIBBLE
03C4 201F03          JSR GETCHP       ;GET ANOTHER CHARACTER
03C7 20AC1F          JSR PACK         ;STUFF IT WITH THE OTHER
03CA A5F8            LDA INL          ;GET THE WHOLE THING
03CC 60              RTS              ;AND RETURN
03CD A207     CRLFD  LDX #$07         ;SET INDEX TO SEVEN,
03CF BDD51F   PRTST  LDA TOP,X        ;OUTPUT CR,LF AND
03D2 20A01E          JSR OUTCH        ;NULLS
03D5 CA              DEX              ;COUNT THE CHARACTERS
03D6 10F7            BPL PRTST        ;LOOP UNTIL DONE
03D8 60              RTS              ;AND RETURN
03D9 00              BRK
```

assignments and a program based on the KIM routines, we can minimize the effort needed to build and program a new serial port. The program in listing 1 is basically a copy of the KIM software UART. Note that your choice of input pin will allow you to use these same routines to cause the input from the terminal or a keyboard to generate an interrupt if you so choose. This may be implemented following instructions in the *KIM User Manual* (Appendix H) for using PB7 to cause an interrupt.

Any routine which calls this serial I/O program should first call INIT - (JSR INIT), the normal KIM-1 power-up initialization routine which configures the B Application Port as output on PB0. If you use the remaining five pins of Port B for other purposes, you must override the pin assignments or change the value loaded in X by the statement at 0251_{16} to accommodate the needs of your other hardware. Once the new port has been initialized, you can use any of the routines in this program in exactly the same manner as you have previously used the similar routines from the KIM-1 monitor.

MICRO

Amper Search for the Apple

High speed machine language search routine finds character strings in BASIC arrays.

Alan G. Hill
12092 Deerhorn Dr.
Cincinnati, Ohio 45240

The July, 1979 issue of MICRO included my article entitled "Amper-Sort" which described and utilized the "&" command of Applesoft BASIC to pass parameters to a machine language sort routine. Now comes Amper-Search, a program which, besides being a useful addition to your Amper-library, demonstrates how parameters can be passed bi-directionally.

Amper-Search is a high-speed character search routine that will find and return the subscripts of all occurrences of a specified character string in a target string array. A search of a 2000 element array will take less than 1 second compared to about 90 seconds for an equivalent BASIC routine. Parameters are used to name the target string array, define the character string, define the bounds of the search, and name the variables to receive the subscripts and number of matches. An added bonus in the Amper-Search code is another routine called &DEALLOC. Its function is to give your BASIC program the ability to de-allocate a string array or integer array when it's no longer needed. &DEALLOC can be used with any Applesoft BASIC program.

Let's look at the parameters and how they are passed between the Applesoft program and Amper-Search. The general form is:

&S[EARCH](NA$,L,H,ST$,PL,PH,I%,N%)

Listing 1

```
1  HIMEM: 9 * 4096 + 2 * 256
2 D$ =  CHR$ (4): PRINT D$"NOMONC,I,O"
3  PRINT D$"BLOAD B.AMPER-SEARCH(48K)
4  POKE 1013,76: POKE 1014,0: POKE 1015,146: REM 3F5: JMP  $9200
5  DIM NA$(10),I%(10)
20 NA$(0) = "APPLE CORE"
21 NA$(1) = "CRAB APPLE"
22 NA$(2) = "APPLE&ORANGE"
23 NA$(3) = "APPLE/ORANGE"
24  LIST 5,23
100  REM    FIND ALL OCCURRENCES OF 'APPLE'
101 N% = 0:ST$ = "APPLE"
102  & SEARCH(NA$,0,10,ST$,1,255,I%,N%)
103  LIST 100,102: GOSUB 2000: GOSUB 3000
200  REM  FIND 'APPLE' IN NA$(0) -> NA$(1) COLUMNS 1 -> 5
201 N% = 0:ST$ = "APPLE"
202  & SEARCH(NA$,0,1,ST$,1,5,I%,N%)
203  LIST 200,202: GOSUB 2000: GOSUB 3000
300  REM FIND 'APPLE ORANGE".
301 N% = 0:ST$ = "APPLE" +  CHR$ (14) + "ORANGE"
302  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)
303  LIST 300,302: GOSUB 2000: GOSUB 3000
400  REM  FIND 1ST 'ORANGE'
401 N% =  - 1:ST$ = "ORANGE"
402  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)
403  LIST 400,402: GOSUB 2000: GOSUB 3000
490 ST$ = "CRAB"
492  REM  DYNAMICALLY ALLOCATE/DEALLOCATE M%
495  FOR J = 1 TO 2
500 N% = 0:K% = 0
501  & SEARCH(NA$,0,3,ST$,1,255,K%,N%)
502  DIM M%(N%):N% = 0
503  & SEARCH(NA$,0,3,ST$,1,255,M%,N%)
504  LIST 490,530: GOSUB 2100: GOSUB 3000
510  & DEALLOC(M%)
520 ST$ = "APPLE"
530  NEXT J
600  REM FIND 'E' IN COLUMN 10
601 N% = 0:ST$ = "E"
602  & SEARCH(NA$,0,3,ST$,10,10,I%,N%)
603  LIST 600,602: GOSUB 2000
700  END
2000  IF N% = 0 THEN  PRINT "NONE FOUND": RETURN
2005  FOR I = 0 TO N% - 1
2010  HTAB 4: PRINT NA$(I%(I))
2020  NEXT I
2030  PRINT : RETURN
2100  IF N% = 0 THEN  PRINT "NONE FOUND": RETURN
2105  PRINT
2110  FOR I = 0 TO N% - 1
2120  HTAB 4: PRINT NA$(M%(I))
2130  NEXT I
2140  PRINT : RETURN
3000  FOR I = 1 TO 5000: NEXT I: RETURN
```

(continued)

where:

[ ] bracket optional characters. The "&S" are required characters.

NA$ is the variable name of the single-dimensional string array to be searched.

L is a variable, constant, or expression specifying the value of the subscript of NA$ where the search is to begin, i.e. NA$(L).

H is a variable, constant, or expression specifying the value of the subscript of NA$ where the search is to end, i.e. NA$(H).

ST$ is the variable name of the simple string containing the "search" characters. A special case exists if the string contains a Control N character. See note 4.

PL is a variable, constant, or expression specifying the character position in the NA$(I) string where the search is to begin.

PH is a variable, constant, or expression specifying the character position in the NA$(I) string where the search is to end. PL and PH are equivalent to the MID$ statement of the form: MID$(NA$(I), PL,PH – PL + 1).

I% is the name of the single-dimensional integer array into which the subscripts of NA$ will be placed when a "match" is found. The first occurrence will be placed in I%(0). A special case exists if I% is a simple variable rather than an array variable. See note 5.

N% is the name of the simple integer variable into which the number of "matches" will be placed by Amper-Search. N% should be set to zero each time before Amper-Search is invoked. Setting N% < 0 is a special case. See note 6.

After Amper-Search is invoked, the elements of NA$ which match the ST$ string may be listed with the statement: FOR I=0 TO N% – 1: PRINT NA$(I%(I)): NEXT I.

Notes

1. A match is defined as the consecutive occurrence of all characters in ST$ with those in NA$(L) through NA$(H) and within the PL and PH character positions of NA$(I). A Control N character in the ST$ string is a wild card. It

Run from Listing 1

```
5   DIM NA$(10),I%(10)
20  NA$(0) = "APPLE CORE"
21  NA$(1) = "CRAB APPLE"
22  NA$(2) = "APPLE&ORANGE"
23  NA$(3) = "APPLE/ORANGE"


100  REM   FIND ALL OCCURRENCES OF 'APPLE'
101 N% = 0:ST$ = "APPLE"
102  & SEARCH(NA$,0,10,ST$,1,255,I%,N%)

   APPLE CORE
   CRAB APPLE
   APPLE&ORANGE
   APPLE/ORANGE


200  REM  FIND 'APPLE' IN NA$(0) -> NA$(1) COLUMNS 1 -> 5
201 N% = 0:ST$ = "APPLE"
202  & SEARCH(NA$,0,1,ST$,1,5,I%,N%)

   APPLE CORE


300  REM FIND 'APPLE ORANGE"
301 N% = 0:ST$ = "APPLE" +  CHR$ (14) + "ORANGE"
302  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)

   APPLE&ORANGE
   APPLE/ORANGE


400  REM  FIND 1ST 'ORANGE'
401 N% =  - 1:ST$ = "ORANGE"
402  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)

   APPLE&ORANGE


490 ST$ = "CRAB"
492  REM  DYNAMICALLY ALLOCATE/DEALLOCATE M%
495  FOR J = 1 TO 2
500 N% = 0:K% = 0
501  & SEARCH(NA$,0,3,ST$,1,255,K%,N%)
502  DIM M%(N%):N% = 0
503  & SEARCH(NA$,0,3,ST$,1,255,M%,N%)
504  LIST 490,530: GOSUB 2100: GOSUB 3000
510  & DEALLOC(M%)
520 ST$ = "APPLE"
530  NEXT J


   CRAB APPLE


490 ST$ = "CRAB"
492  REM  DYNAMICALLY ALLOCATE/DEALLOCATE M%
495  FOR J = 1 TO 2
500 N% = 0:K% = 0
501  & SEARCH(NA$,0,3,ST$,1,255,K%,N%)
502  DIM M%(N%):N% = 0
503  & SEARCH(NA$,0,3,ST$,1,255,M%,N%)
504  LIST 490,530: GOSUB 2100: GOSUB 3000
510  & DEALLOC(M%)
520 ST$ = "APPLE"
530  NEXT J


   APPLE CORE
   CRAB APPLE
   APPLE&ORANGE
   APPLE/ORANGE


600  REM FIND 'E' IN COLUMN 10
601 N% = 0:ST$ = "E"
602  & SEARCH(NA$,0,3,ST$,10,10,I%,N%)

   APPLE CORE
   CRAB APPLE
```

Listing 2

```
0  REM  AMPER-SEARCH DEMO
1  REM  BY ALAN G. HILL
1000  GOSUB 10000
1010  POKE 32,20: POKE 33,19: HOME : VTAB 5: PRINT "DO YOU WANT TO": PRINT
    "SPECIFY SEARCH": PRINT "LIMITS(Y/N)? ";: GET A$: PRINT
1020  IF A$ <  > "Y" THEN 1080
1030  VTAB 10: CALL  - 868: INPUT "LOWER SUBSCRIPT:";L: IF L < 0 OR L > 21
     THEN  PRINT B$: GOTO 1030
1040  VTAB 12: CALL  - 868: INPUT "UPPER SUBSCRIPT:";H: IF H < 0 OR H > 21
     OR H < L THEN  PRINT B$: GOTO 1040
1050  VTAB 14: CALL  - 868: INPUT "LOWER COLUMN:";PL: IF PL < 1 OR PL > 25
    5 THEN  PRINT B$: GOTO 1050
1060  VTAB 16: CALL  - 868: INPUT "UPPER COLUMN:";PH: IF PH < 1 OR PH > 25
    5 OR PH < PL THEN  PRINT B$: GOTO 1060
1065  VTAB 18: CALL  - 868: PRINT "FIRST/ALL?";: GET A$: PRINT : IF A$ = "
    F" THEN F% =  - 1
1070  GOTO 1120
1080 L = 0: REM  START AT NA$(0)
1090 H = I: REM   SEARCH ALL
1100 PL = 1: REM  START WITH 1ST COLUMN
1110 PH = 255: REM  MAXIMUM COLUMNS
1115 F% = 0: REM FIND ALL
1120  POKE 32,0: POKE 33,39: VTAB 23: CALL  - 868
1130  INVERSE : PRINT "STRING:";: NORMAL : INPUT " ";ST$
1140  IF  LEN (ST$) = 0 THEN  END
1150 N% = F%: REM INIT COUNTER
1160  REM INVOKE 'AMPER-SEARCH'
1170  & SEARCH(NA$,L,H,ST$,PL,PH,I%,N%)
1180  REM LIST FOUND STRINGS
1190  POKE 32,20: POKE 33,19: HOME
1200  IF N% <  = 0 THEN  PRINT "NONE FOUND": GOTO 1120
1210  FOR I = 0 TO N% - 1
1220  VTAB I%(I) + 1: PRINT NA$(I%(I))
1230  NEXT I
1240  GOTO 1120
10000  REM HOUSEKEEPING
10010  HIMEM: 9 * 4096 + 2 * 256
10015  POKE 235,0
10020 D$ =  CHR$ (4)
10030 B$ =  CHR$ (7)
10040  PRINT D$"NOMONC,I,O"
10050  POKE 1013,76: POKE 1014,0: POKE 1015,146: REM  SETUP '&' VECTOR AT
     $3F5 TO JMP $9200
10060  TEXT : HOME : VTAB 10: HTAB 12: PRINT "AMPER-SEARCH DEMO"
10070  HTAB 19: PRINT "BY": HTAB 14: PRINT "ALAN G. HILL"
10080  PRINT D$"BLOAD B.AMPER-SEARCH(48K)"
10090  FOR I = 1 TO 1000: NEXT I
10100  DIM NA$(22),I%(22)
10110 I = 0
10120  REM INITIALIZE STRING ARRAY
10130  READ NA$(I)
10140  IF NA$(I) = "END" THEN 10160
10150 I = I + 1: GOTO 10130
10160 I = I - 1
10170  HOME
10180  FOR K = 0 TO I
10190  PRINT K; TAB( 4);NA$(K)
10200  NEXT K
10210  RETURN
11000  REM SAMPLE STRINGS
11010  REM NOTE: THIS DEMO IS SCREEN ORIENTED. DON'T PUT MORE THAN 22 ITEM
    S IN THE DATA STATEMENT LIST.

11020  DATA  APPLE II,APPLE SIDER,APPLE CIDER,APPLEVENTION,APPLE PI,APPLES
    AUCE,APPLE TREE,APPLE ORCHARD
11030  DATA APPLE II PLUS,APPLES & ORANGES,APPLE BLOSSOM, CANDIED APPLES,AP
    PLE/ORANGE,APPLESOFT,APPLEODIAN,APPLEVISION
11040  DATA  APPLE STEM,APPLE CORE,APPLE-A-DAY,APPLE PIE, APPLE PEEL,APPLE-
    OF-MY-EYE
```

will match any character in its corresponding NA$(I) position.

2. Any valid variable name may be used as a parameter.

3. 0≤L≤H≤maximum number of elements in NA$. Elements of NA$ can be null strings.

4. 1≤PL≤PH≤255. A PH>LEN (NA$(I)) is allowed and will ensure that the entire NA$(I) string is searched.

5. I% must be dimensioned large enough to hold all matches; i.e. DIM I%(N%). Since you don't know the number of matches before Amper-Search is invoked, you have two alternatives. I% can be dimensioned the same size as NA$, thus assuring enough space to accommodate a complete match. This may waste memory or require more memory than is available. A second alternative is to first define I% as a *simple* variable before invoking Amper-Search. In this special case, Amper-Search will return the number of matches *only*. Your program can then DIM I%(N%), set N%=0, and re-invoke Amper-Search to return the subscripts. Its speed makes this option practical even for large arrays and will conserve memory by not allocating unused I% elements.

6. N% should be ≤ 0 prior to invoking Amper-Search. Set N% = 0 if you want all matches. If N% = 0 upon return, there were no matches. Set N% = −1 if you only want the *first* occurrence of a match. In this special case, N% will be −1 if there were no matches, or +1 if a match were found. The subscript of the matching NA$ element will be found in I%(0).

Note 5 described a method for allocating the minimum size for I% that is large enough to hold the maximum number of matches. You could ask, "What if I use &SEARCH iteratively with a different ST$ string each time that has more matches than I% can hold? Won't that cause a BAD SUBSCRIPT ERROR?" Yes it will. Ideally, one would like to de-allocate I% and re-DIMension it at the new minimum size. The CLEAR command won't do the job because it will clear all variables. Now you should see the utility of yet another Amper-library routine called &DEALLOC which performs the needed function. The general form is:

&D[EALLOC] (A,B,N)

where A,B,N are the named variables of the integer and string arrays to be de-allocated.

[ ] bracket optional characters. "&D" are required.

For example: &D(I%) will de-allocate the I% integer array, &D(XY$,K%) will de-allocate the XY$ string array and the K% integer array.

In order to complete the de-allocation process, your program must follow the &D(XY$) statement with an X=FRE(0) housekeeping statement to regain the memory from character strings referred to only by the de-allocated string array. &DEALLOC cannot be used to increase the size of an array while preserving the current contents of the array.

Now let's look at some simple examples created by running the program in listing 1.

Listing 2 is a general BASIC demo with which you can experiment to learn how Amper-Search can be used.

Some of the routines in Amper-Search can be adapted for use in other Amper-library machine language routines. In addition to the Apple routines described in the July Amper-Sort article, the following routines may also be useful:

GNAME retrieves the string or integer variable name from the "&" parameter list and places it in the NAME buffer in your machine language program. The A-Reg is returned with a "$" or "%" character.

INTE converts the positive ASCII variable name in NAME to Applesoft's 2-character negative ASCII naming convention for integer variable names. If the A-Reg does not contain a "%" upon entry, the carry flag will be set upon return.

STRING performs the same function for string variable names as INTE does for integer variables. The A-Reg must contain a "$" upon entry.

FARRAY will search variable space for the array variable name contained in the NAME buffer. If found, its address will be returned in the X and Y Regs. If not found, the carry flag will be set.

FSIMPL performs the same function for simple variables as FARRAY does for array variables.

&DEALLOC also uses several of the above routines. Similar routines reside somewhere in the Applesoft interpreter, and if they are known, these routines can be adapted.

Amper-Search was assembled using the Microproducts 6 Character Label Editor/Assembler. The Link command makes it very easy to put the above routines in your subroutine library for recall, when needed, by the assembler. Anyone desiring a tape cassette containing the Demo program, the object code assembled at $5200, a copy at $9200 (all for Applesoft ROM), and the source code in Microproducts 6 Character Label Editor/Assembler format may send $6.00 to me at the above address.

My thanks to Bob Kovacs who challenged me to write Amper-Search.

MICRO

Listing 3

```
              ;***************
              ;             *
              ; AMPER-SEARCH *
              ;AND DEALLOCATE*
              ;      BY      *
              ; ALAN G. HILL *
              ;             *
              ;  COMMERCIAL  *
              ;    RIGHTS    *
              ;   RESERVED   *
              ;             *
              ;***************
              ;             *
              ;   FEBRUARY   *
              ;     1980     *
              ;             *
              ;***************
              ; DEFINE ADDRESSES *
              ;
              NAPTR  EQU 00D0
              SAPTR  EQU 00D2
              JAPTR  EQU 00D4
              NPT    EQU 00D6
              L      EQU 00D8
              H      EQU 00DA
              PL     EQU 00DC
              PH     EQU 00DD
              TEM6X  EQU 00DE
              NAPTH  EQU 00E0
              CNAPTR EQU 00E2
              CSAPTR EQU 00E4
              SAVEY  EQU 00E6
              PS     EQU 00E7
              LENNA  EQU 00E8
              LENSA  EQU 00E9
              SWITCH EQU 00EA
              SIZE   EQU 00EB
              OFFSET EQU 00D2
              A1     EQU 00D4
              Z50    EQU 0050
              CHRGOT EQU 00B7
              CHRGET EQU 00B1
              COUT   EQU FDED
              ;          ROM        RAM
              GETBYT EQU E6F8    ; 1EEF
              SYNERR EQU DEC9    ; 16CC
              FRMNUM EQU DD67    ; 156A
              GETADR EQU E752    ; 1F49
              ;
                     ORG 9200
                     OBJ 9200
              ;
              ; PROCESS &
9200  48      BEGIN  PHA
9201  203195         JSR SAVEZP      ; SAVE ZERO PG
9204  68             PLA
9205  A202           LDX #$02
9207  CA      CHRSFN DEX
9208  3053           BMI ERRX
920A  DDA395         CMP CHRTBL,X    ; 'S' OR 'D'
920D  D0F8           BNE CHRSFN      ; TRY AGAIN
920F  8A             TXA
9210  0A             ASL             ; TIMES 2
9211  AA             TAX
9212  20B100  SR02   JSR CHRGET      ; NEXT CHAR
9215  F046           BEQ ERRX
9217  C928           CMP #$28        ; (
9219  D0F7           BNE SR02
921B  BDA095         LDA LOC+01,X    ; JMP TO
921E  48             PHA             ; ROUTINE
921F  BD9F95         LDA LOC,X       ; VIA
9222  48             PHA             ; RTS
9223  60             RTS
              ;
              ; AMPER-SEARCH
              ;
9224  201E94  SEARCH JSR GNAME       ; GET NAME
9227  205D94         JSR STRING      ; CONVERT
922A  207494         JSR FARRAY      ; FIND NAME
922D  B034           BCS ERRV
922F  86D0           STX NAPTR       ; NA$
9231  84D1           STY NAPTR+01
9233  20B100         JSR CHRGET
9236  2067DD         JSR FRMNUM
9239  2052E7         JSR GETADR
923C  A550           LDA Z50
923E  85D8           STA L           ; LOWER SUBSC
9240  A551           LDA Z50+01
9242  85D9           STA L+01
9244  20B100         JSR CHRGET
9247  2067DD         JSR FRMNUM
924A  2052E7         JSR GETADR
924D  A550           LDA Z50
924F  85DA           STA H           ; UPPER SUBSC
9251  A551           LDA Z50+01
9253  85DB           STA H+01
9255  201E94         JSR GNAME
9258  205D94         JSR STRING
925B  901D           BCC SR20
              ;
              ; ERROR *
              ;
925D  205495  ERRX   JSR RSZP
9260  4CC9DE         JMP SYNERR
              ;
              ; VARIABLE NOT FOUND MSG *
              ;
9263  A200    ERRV   LDX #$00
```

```
9265  BDA595      SR18   LDA MSG1,X     ; ERROR MSG
9268  C9C0               CMP #$C0       ; @ DELIMITER
926A  F0F1               BEQ ERRX
926C  0980               ORA #$80
926E  20EDFD             JSR COUT
9271  E00C               CPX #$0C
9273  D002               BNE SR19
9275  A219               LDX #$19
9277  E8          SR19   INX
9278  D0EB               BNE SR18       ; ALWAYS
                  ;
927A  20AE94      SR20   JSR FSIMPL     ; FIND NAME
927D  B0E4               BCS ERRV
927F  86D2               STX SAPTR      ; ST$
9281  84D3               STY SAPTR+01
9283  20B100             JSR CHRGET
9286  20F8E6             JSR GETBYT
9289  86DC               STX PL         ;FIRST POSITION
928B  20B100             JSR CHRGET
928E  20F8E6             JSR GETBYT
9291  86DD               STX PH         ; LAST POSITION
9293  201E94             JSR GNAME
9296  203D94             JSR INTE
9299  B0C2               BCS ERRX
929B  207494             JSR FARRAY
929E  9009               BCC SR21
92A0  20AE94             JSR FSIMPL
92A3  B0BE               BCS ERRV
92A5  A9FF               LDA #$FF
92A7  85EB               STA SIZE       ; # OF HITS ONLY
92A9  86D4        SR21   STX JAPTR      ; I%
92AB  84D5               STY JAPTR+01
92AD  201E94             JSR GNAME
92B0  203D94             JSR INTE
92B3  B0A8               BCS ERRX
92B5  20AE94             JSR FSIMPL
92B8  B0A9               BCS ERRV
92BA  86D6               STX NPT        ; N%
92BC  84D7               STY NPT+01
92BE  20B100             JSR CHRGET
92C1  D09A               BNE ERRX
                  ; FINISHED PARAMETERS *
                  ;
                  ; SET UP POINTERS *
                  ;
92C3  18                 CLC
92C4  A5D4               LDA JAPTR
92C6  6907               ADC #$07
92C8  85D4               STA JAPTR      ; I%
92CA  A5D5               LDA JAPTR+01
92CC  6900               ADC #$00
92CE  85D5               STA JAPTR+01
92D0  A5DA               LDA H
92D2  8550               STA Z50
92D4  A5DB               LDA H+01
92D6  8551               STA Z50+01
92D8  A903               LDA #$03
92DA  8554               STA $54
92DC  A900               LDA #$00
92DE  8555               STA $55
92E0  20E594             JSR MPLY
92E3  86E0               STX NAPTH      ; NA$(H)
92E5  84E1               STY NAPTH+01
92E7  A5D8               LDA L
92E9  8550               STA Z50
92EB  A5D9               LDA L+01
92ED  8551               STA Z50+01
92EF  20E594             JSR MPLY
92F2  86D0               STX NAPTR      ; NA$(L)
92F4  84D1               STY NAPTR+01
                  ;
92F6  18                 CLC
92F7  A5D2               LDA SAPTR
92F9  6902               ADC #$02
92FB  85D2               STA SAPTR      ; ST$
92FD  A5D3               LDA SAPTR+01
92FF  6900               ADC #$00
9301  85D3               STA SAPTR+01
9303  A000               LDY #$00
9305  B1D2               LDA (SAPTR),Y
9307  D003               BNE SR22
9309  4C1A94             JMP RETURN     ; NULL
930C  85E9        SR22   STA LENSA
930E  C8                 INY
930F  B1D2               LDA (SAPTR),Y; SAVE
9311  85E4               STA CSAPTR     ; ADDRESS
9313  C8                 INY
9314  B1D2               LDA (SAPTR),Y
9316  85E5               STA CSAPTR+01
                  ;
                  ; START SEARCH *
                  ;
```

```
9318  A000        NEXT   LDY #$00
931A  B1D0               LDA (NAPTR),Y
931C  F04A               BEQ NEXTNA     ; NULL
931E  85E8               STA LENNA      ; LEN(NA$( ))
9320  C8                 INY
9321  B1D0               LDA (NAPTR),Y
9323  85E2               STA CNAPTR
9325  C8                 INY
9326  B1D0               LDA (NAPTR),Y
9328  85E3               STA CNAPTR+01
932A  A4DC               LDY PL
932C  88                 DEY
932D  C4E8               CPY LENNA
932F  B037               BCS NEXTNA
9331  A900        NXTNAC LDA #$00
9333  85E7               STA PS         ;CURRENT POSITIO
9335  85EA               STA SWITCH
9337  B1E2        CONT   LDA (CNAPTR),Y
9339  C8                 INY
933A  84E6               STY SAVEY
933C  A4E7               LDY PS
933E  D1E4               CMP (CSAPTR),Y
9340  F006               BEQ SR25       ; POSSIBLE MATCH
9342  B1E4               LDA (CSAPTR),Y
9344  C90E               CMP #$0E       ; CNTL N
9346  D011               BNE SR26       ; NOT WILD CARD
                  ;
                  ; POSSIBLE MATCH *
                  ;
9348  A9FF        SR25   LDA #$FF
934A  85EA               STA SWITCH
934C  C8                 INY
934D  C4E9               CPY LENSA      ; AT END?
934F  F038               BEQ MATCH      ; IT'S A MATCH!
9351  E6E7               INC PS
9353  F013               BEQ NEXTNA
9355  A4E6               LDY SAVEY
9357  D0DE               BNE CONT       ; ALWAYS
9359  A4E6        SR26   LDY SAVEY
935B  24EA               BIT SWITCH
935D  1001               BPL SR28
935F  88                 DEY
9360  C4E8        SR28   CPY LENNA      ; AT END?
9362  B004               BCS NEXTNA     ; BR YES
9364  C4DD               CPY PH         ; LAST POSITION
9366  90C9               BCC NXTNAC     ; NEXT CHAR
9368  18          NEXTNA CLC            ; NEXT NA$(I)
9369  A5D0               LDA NAPTR
936B  6903               ADC #$03
936D  85D0               STA NAPTR
936F  A5D1               LDA NAPTR+01
9371  6900               ADC #$00
9373  85D1               STA NAPTR+01
9375  E6D8               INC L
9377  D002               BNE SR33
9379  E6D9               INC L+01
937B  38          SR33   SEC
937C  A5E0               LDA NAPTH
937E  E5D0               SBC NAPTR
9380  A5E1               LDA NAPTH+01
9382  E5D1               SBC NAPTR+01
9384  B092               BCS NEXT
9386  4C1A94             JMP RETURN     ; AT NA$(H)
                  ;
                  ; FOUND A MATCH *
                  ;
9389  24EB        MATCH  BIT SIZE
938B  3018               BMI SZONLY     ; # MATCHES ONLY
938D  A000               LDY #$00
938F  A5D9               LDA L+01       ; SUBSCRIPT
9391  91D4               STA (JAPTR),Y
9393  C8                 INY
9394  A5D8               LDA L
9396  91D4               STA (JAPTR),Y
9398  18                 CLC
9399  A5D4               LDA JAPTR
939B  6902               ADC #$02
939D  85D4               STA JAPTR
939F  A5D5               LDA JAPTR+01
93A1  6900               ADC #$00
93A3  85D5               STA JAPTR+01
93A5  A003        SZONLY LDY #$03
93A7  18                 CLC
93A8  B1D6               LDA (NPT),Y
93AA  6901               ADC #$01       ; N%=N%+1
93AC  91D6               STA (NPT),Y
93AE  88                 DEY
93AF  B1D6               LDA (NPT),Y
93B1  3007               BMI ONLY1      ;1ST OCCURRENCE
93B3  6900               ADC #$00
93B5  91D6               STA (NPT),Y
93B7  4C6893             JMP NEXTNA
```

(continued)

```
93BA  A900          ONLY1  LDA #$00
93BC  91D6                 STA (NPT),Y
93BE  C8                   INY
93BF  A901                 LDA #$01     ; N%=1
93C1  91D6                 STA (NPT),Y
                    ;
                    ; FINISHED AMPER-SEARCH *
                    ;
93C3  4C1A94               JMP RETURN
                    ;
93C6  4C5D92        ERRXX  JMP ERRX
93C9  4C6392        ERRVX  JMP ERRV
                    ;
                    ; DEALLOCATE *
                    ;
93CC  201E94        DEALLO JSR GNAME    ; GET NAME
93CF  C924                 CMP #$24     ; $
93D1  F005                 BEQ RE50
93D3  203D94               JSR INTE     ; %
93D6  D003                 BNE RE55     ; ALWAYS
93D8  205D94        RE50   JSR STRING
93DB  B0E9          RE55   BCS ERRXX
93DD  207494               JSR FARRAY
93E0  B0E7                 BCS ERRVX
93E2  86D0                 STX NAPTR    ; NA$
93E4  84D1                 STY NAPTR+01
93E6  A002                 LDY #$02
93E8  B1D0                 LDA (NAPTR),Y
93EA  85D2                 STA OFFSET
93EC  C8                   INY
93ED  B1D0                 LDA (NAPTR),Y
93EF  85D3                 STA OFFSET+01
93F1  18                   CLC
93F2  A5D2                 LDA OFFSET
93F4  65D0                 ADC NAPTR
93F6  85D4                 STA A1
93F8  A5D3                 LDA OFFSET+01
93FA  65D1                 ADC NAPTR+01
93FC  85D5                 STA A1+01
93FE  201495               JSR MOVE     ; MOVE VARIABLES
9401  38                   SEC
9402  A56D                 LDA $6D
9404  E5D2                 SBC OFFSET
9406  856D                 STA $6D
9408  A56E                 LDA $6E
940A  E5D3                 SBC OFFSET+01
940C  856E                 STA $6E
940E  20B700               JSR CHRGOT
9411  C929                 CMP #$29     ; )
9413  D0B7                 BNE DEALLO   ; NEXT VAR
9415  20B100               JSR CHRGET
9418  D0AC                 BNE ERRXX
                    ;
                    ; FINISHED *
                    ;
941A  205495        RETURN JSR RSZP     ; RESTORE PAGE0
941D  60                   RTS
                    ;
                    ;**************
                    ; SUBROUTINES *
                    ;**************
                    ;
                    ; GET VARIABLE NAME *
941E  A200          GNAME  LDX #$00
9420  20B100        GR01   JSR CHRGET
9423  C92C                 CMP #$2C     ; ,
9425  F011                 BEQ GR03
9427  C929                 CMP #$29     ; )
9429  F00D                 BEQ GR03
942B  9DAF95               STA NAME,X   ; SAVE NAME
942E  E8                   INX
942F  E010                 CPX #$10     ; 16 IS ENOUGH
9431  D0ED                 BNE GR01
9433  68                   PLA
9434  68                   PLA          ; POP STACK
9435  4C5D92               JMP ERRX
9438  CA            GR03   DEX
9439  BDAF95               LDA NAME,X   ; $ OR %
943C  60                   RTS
                    ;
                    ; INTEGER NAME *
                    ;
943D  C925          INTE   CMP #$25     ; %
943F  D01A                 BNE ERRI     ; NOT %
9441  8DB195               STA NAME+02  ; SAVE
9444  E001                 CPX #$01     ; NAME
9446  D004                 BNE GR10     ; IN
9448  A980                 LDA #$80     ; APPLESOFT
944A  D007                 BNE GR14     ; FORMAT
944C  A201          GR10   LDX #$01
944E  A980          GR12   LDA #$80
9450  1DAF95               ORA NAME,X
```

```
9453  9DAF95        GR14   STA NAME,X
9456  CA                   DEX
9457  10F5                 BPL GR12
9459  18                   CLC          ; CLEAR ERR
945A  60                   RTS
945B  38            ERRI   SEC          ; SET ERR
945C  60                   RTS
                    ;
                    ; STRING NAME *
                    ;
945D  C924          STRING CMP #$24     ; $
945F  D011                 BNE ERRS
9461  8DB195               STA NAME+02
9464  A980                 LDA #$80
9466  E001                 CPX #$01     ; SAVE
9468  F003                 BEQ GR18     ; NAME
946A  0DB095               ORA NAME+01
946D  8DB095        GR18   STA NAME+01
9470  18                   CLC
9471  60                   RTS
9472  38            ERRS   SEC          ; SET ERR
9473  60                   RTS
                    ;
                    ; FIND ARRAY NAME *
                    ; IN VARIABLE SPACE *
                    ;
9474  A56B          FARRAY LDA $6B
9476  85DE                 STA TEM6X
9478  A56C                 LDA $6C
947A  85DF                 STA TEM6X+01
947C  A000          F02    LDY #$00
947E  B1DE                 LDA (TEM6X),Y
9480  CDAF95               CMP NAME     ; 1ST CHAR
9483  D008                 BNE F04
9485  C8                   INY
9486  B1DE                 LDA (TEM6X),Y
9488  CDB095               CMP NAME+01  ; 2ND CHAR
948B  F01B                 BEQ FOUND
948D  18            F04    CLC          ; LOOK AT
948E  A002                 LDY #$02     ; NEXT NAME
9490  B1DE                 LDA (TEM6X),Y
9492  65DE                 ADC TEM6X
9494  48                   PHA
9495  C8                   INY
9496  B1DE                 LDA (TEM6X),Y
9498  65DF                 ADC TEM6X+01
949A  85DF                 STA TEM6X+01
949C  68                   PLA
949D  85DE                 STA TEM6X
949F  C56D                 CMP $6D
94A1  A5DF                 LDA TEM6X+01
94A3  E56E                 SBC $6E
94A5  90D5                 BCC F02      ; TRY NEXT ONE
94A7  60                   RTS          ; NOT FOUND
                    ;
94A8  A6DE          FOUND  LDX TEM6X    ; RTN WITH
94AA  A4DF                 LDY TEM6X+01 ; ADDRESS
94AC  18                   CLC
94AD  60                   RTS
                    ;
                    ; FIND SIMPLE NAME *
                    ; IN VARIABLE SPACE *
                    ;
94AE  A569          FSIMPL LDA $69
94B0  85DE                 STA TEM6X
94B2  A56A                 LDA $6A
94B4  85DF                 STA TEM6X+01
94B6  A000          FS2    LDY #$00
94B8  B1DE                 LDA (TEM6X),Y
94BA  CDAF95               CMP NAME     ; 1ST CHAR
94BD  D008                 BNE FS4
94BF  C8                   INY
94C0  B1DE                 LDA (TEM6X),Y
94C2  CDB095               CMP NAME+01  ; 2ND CHAR
94C5  F018                 BEQ FOUNDS
94C7  18            FS4    CLC          ; TRY NEXT ONE
94C8  A5DE                 LDA TEM6X
94CA  6907                 ADC #$07     ; DISPLACEMENT
94CC  85DE                 STA TEM6X
94CE  A5DF                 LDA TEM6X+01
94D0  6900                 ADC #$00
94D2  85DF                 STA TEM6X+01
94D4  A5DE                 LDA TEM6X
94D6  C56D                 CMP $6D      ; AT END?
94D8  A5DF                 LDA TEM6X+01
94DA  E56E                 SBC $6E
94DC  90D8                 BCC FS2      ; NEXT ONE
94DE  60                   RTS          ; NOT FOUND
                    ;
94DF  A6DE          FOUNDS LDX TEM6X    ; RTN WITH
94E1  A4DF                 LDY TEM6X+01 ; ADDRESS
94E3  18                   CLC
94E4  60                   RTS
```

```
                  ;
                  ; MULTIPLY ROUTINE *
                  ;
94E5  18          MPLY   CLC
94E6  A5D0               LDA NAPTR
94E8  6907               ADC #$07
94EA  8552               STA $52
94EC  A5D1               LDA NAPTR+01
94EE  6900               ADC #$00
94F0  8553               STA $53
                  ;
                  ; FROM 'RED' MANUAL *
                  ;
94F2  A010               LDY #$10
94F4  A550        MUL2   LDA $50
94F6  4A                 LSR
94F7  900C               BCC MUL4
94F9  18                 CLC
94FA  A2FE               LDX #$FE
94FC  B554        MUL3   LDA $54,X
94FE  7556               ADC $56,X
9500  9554               STA $54,X
9502  E8                 INX
9503  D0F7               BNE MUL3
9505  A203        MUL4   LDX #$03
9507  7650        MUL5   ROR $50,X
9509  CA                 DEX
950A  10FB               BPL MUL5
950C  88                 DEY
950D  D0E5               BNE MUL2
950F  A650               LDX Z50
9511  A451               LDY Z50+01
9513  60                 RTS
                  ;
                  ; MOVE VARIABLES *
                  ;
9514  A000        MOVE   LDY #$00
9516  B1D4        MV01   LDA (A1),Y
9518  91D0               STA (NAPTR),Y
951A  E6D0               INC NAPTR
951C  D002               BNE NXTA1
951E  E6D1               INC NAPTR+01
9520  A5D4        NXTA1  LDA A1
9522  C56D               CMP $6D
9524  A5D5               LDA A1+01
9526  E56E               SBC $6E
9528  E6D4               INC A1
952A  D002               BNE MV02
952C  E6D5               INC A1+01
952E  90E6        MV02   BCC MV01      ;NEXT ONE
9530  60                 RTS
                  ;
                  ; SAVE ZERO *
                  ; PAGE SPACE *
                  ;
9531  A200        SAVEZP LDX #$00
9533  B5D0        SV02   LDA NAPTR,X
9535  9DD095             STA ZPSV,X
9538  E8                 INX
9539  E020               CPX #$20      ; SAVE
953B  D0F6               BNE SV02      ; 32 SPOTS
953D  A200               LDX #$00
953F  B550        SV04   LDA $50,X     ; ALSO $50.$55
9541  9DCA95             STA SV50,X
9544  E8                 INX
9545  E006               CPX #$06
9547  D0F6               BNE SV04
9549  A20F               LDX #$0F
954B  A920               LDA #$20      ; CLEAR
954D  9DAF95      CLEAR  STA NAME,X    ; NAME AREA
9550  CA                 DEX
9551  10FA               BPL CLEAR
9553  60                 RTS
                  ;
                  ; RESTORE ZERO *
                  ; PAGE SPACE *
                  ;
9554  A200        RSZP   LDX #$00
9556  BDD095      RS02   LDA ZPSV,X
9559  95D0               STA NAPTR,X
955B  E8                 INX
955C  E020               CPX #$20
955E  D0F6               BNE RS02
9560  A200               LDX #$00
9562  BDCA95      RS04   LDA SV50,X
9565  9550               STA $50,X
9567  E8                 INX
9568  E006               CPX #$06
956A  D0F6               BNE RS04
956C  60                 RTS
                  ;
                  ; DATA STORAGE *
                  ;
956D  C1CDD0C5D2ADD3C5C1D2C3C8
9579  C1CCC1CEA0C7AEA0C8C9CCCC
9585  C3CFCDCDC5D2C3C9C1CCA0D2C9C7C8D4D3A0
9597  D2C5D3C5D2D6C5C4
959F  CB93        LOC    DFD CB93      ; DEALLOC-1
95A1  2392               DFD 2392      ; SEARCH-1
95A3  44          CHRTBL DFD 44        ; D
95A4  53                 DFD 53        ; S
95A5  8D          MSG1   DFD 8D
95A6  D6C1D2C9C1C2CCC5A0
95AF  A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0
95BF  8D                 DFD 8D
95C0  CECFD4A0C6CFD5CEC4
95C9  C0                 DFD "@"
95CA  A0A0A0A0A0A0
95D0  A0          -ZPSV  DFD " "       ; $20 SPACES
```

Memory Expansion for the Superboard

A less expensive way to add memory to the Superboard using the OSI 527 memory expansion board.

Fred Boness
11703 60th St.
Kenosha, Wisconsin 53142

The greatest disadvantage of owning a single board computer is its limited memory. The Superboard has space for only 8K of memory, although Ohio Scientific offers the 610 expansion board, which can add 24K to the Superboard. However, a 610 with only 8K of memory costs more than the Superboard itself. There is more on the 610 than memory, like a floppy disk controller, but all I want is a little more memory.

OSI offers a variety of memory boards for their 48-line bus. Adapting one of these to the Superboard means finding the necessary address, data, and control signals on the Superboard's 40-pin expansion socket, and matching them to the 48-line bus. Fortunately, OSI has designed a simple and straightforward system. Figure 1 shows the expansion socket and corresponding bus lines. Only 27 lines are used. Note that +5 volts is not available at the expansion socket. The user's manual for the Superboard includes a complete description of the 48-line bus.

Building the 527

I decided to use the OSI 527 memory board because it is the most like the 610. It is a 24K board which uses 2114 chips. One of the nice things OSI does for experimenters is to sell bare printed circuit boards for many of its products. (OSI sells a fully populated 527 as a CM-9.)

Most of the control and memory decoding logic functions are shown in figure 2. The six high address lines are decoded by four 74LS138 three-to-eight-line decoders. Jumpers W1, W2, and W3 at F9 determine the starting addresses of three independent 8K blocks of memory on 8K boundaries. No changes are made here or at W4, which selects the memory management option. Parts C10, C11, and SW11 are also for memory management and will not be needed.

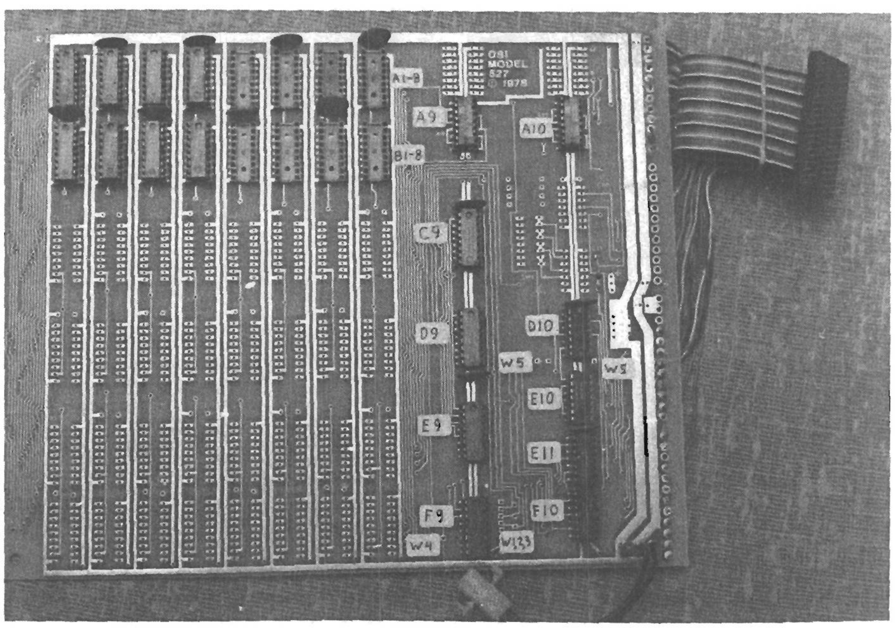
C9, D9, and E9 select pairs of 2114's beginning at A1 and B1 with the active low chip enable lines CEO to CE23.

F10 and E11 are 74LS04 hex inverters used as address line buffers. There are jumpers across each inverter that must be cut before the sockets for the 74LS04's are soldered in place. These jumpers are not shown on the schematics provided by OSI. Jumper W5 at D10 must be changed in two places to buffer address line A6.

While the Superboard documentation uses the name 02 throughout, the 48-line bus has both 02, B39, and

	1 IRQ		40	GND
	2 NMI		39	GND
B4	3 Data direction		38	GND
B5	4 DO		37	GND
B6	5 D1	B9	36	D4
B7	6 D2	B10	35	D5
B8	7 D3	B11	34	D6
	8 GND	B12	33	D7
	9 GND	B40	32	R/W
	10 GND	B42	31	02
	11 —		30	GND
B35	12 A2		29	GND
B34	13 A1		28	GND
B38	14 A0	B48	27	A15
B36	15 A3	B47	26	A14
B37	16 A4	B46	25	A13
B31	17 A5	B45	24	A12
B29	18 A6	B44	23	A11
B30	19 A7	B43	22	A10
B32	20 A8	B33	21	A9

***Figure 1:* Pinouts for the 40-pin socket and corresponding bus (Bxx) lines.**



02VMA, B42. Use 02VMA for this board. VMA is actually a 6800 signal, Valid Memory Address.

The data direction signal, DD, is generated by the memory board and controls the direction of the two 8T26 bus driver/receivers on the board and two 8T28 bus driver/receivers on the Superboard. The 8T28's are the only extra parts needed by the Superboard. They are placed in the sockets between the expansion connecter and the 6502.

I considered several ways of positioning the memory board. I wanted it to be accessible for servicing and convenient in use. It now sits behind the keyboard on nylon standoffs, component side up, with the bus on the left and a 40-conducter ribbon cable running under the board to the expansion socket.

There is a provision in the corner of the 527 board to bring in power and ground. This makes it easy to power the memory board with a short jumper from the fuse on the Superboard. Ground is to a wide trace near the fuse.

The ribbon cable can be soldered into the plated-through holes intended for Molex connecters. Bending hairpins in the tinned wire ends will help since these holes are large. All the wires were first threaded through the holes and checked for correct connection. Then the assembly was checked for fit on the Superboard before the wires were cut to length and soldered.

Testing

The Superboard does its own memory test and I used that for the first sign of success. What I got was the first indication of failure. Further testing using POKE and PEEK showed that no part of the 4K on the board was working.

It was several days later that I found the last of seven trace bridges on the board. One such bridge had been repaired by OSI. Perseverance was rewarded with the simple line "11519 BYTES FREE".

Conclusion

I never liked the idea that the Superboard was a "weak sister" of limited capability. Now it looks as though any board offered for OSI's main line of computers can be adapted to the Superboard. How would you like 16 lines of analog I/O or a Votrax? With a little extra work you could add a backplane. Take your choice.

MICRO

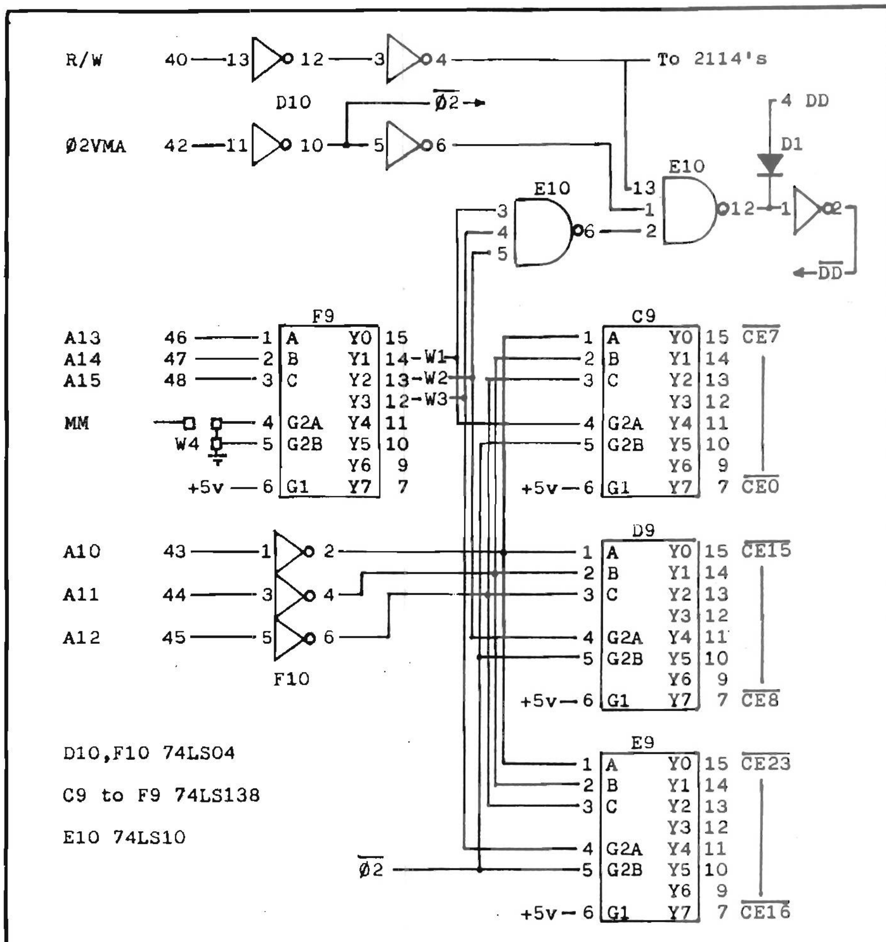
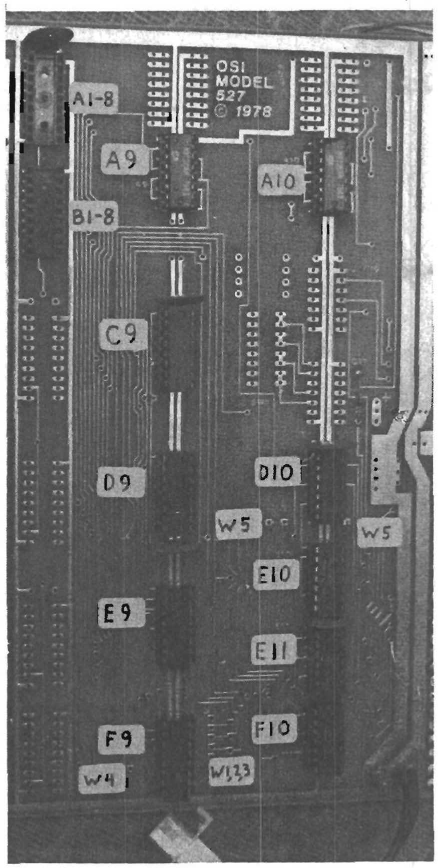


Figure 2: **Decoding of central and six high address lines.**



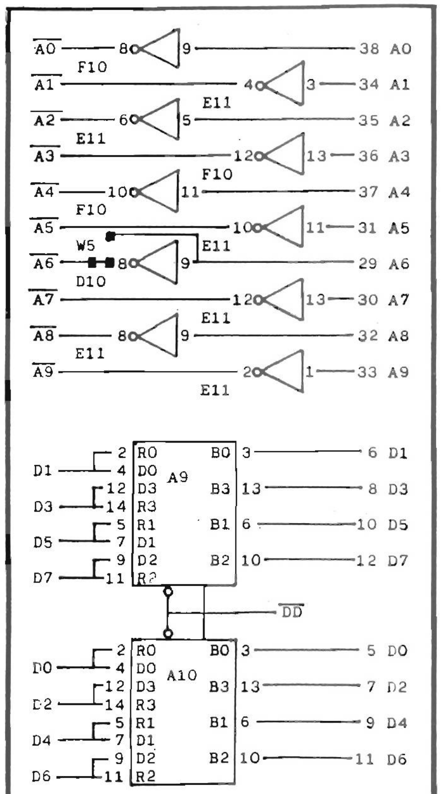


Figure 3: **Buffering of data and low address line.**

Horizontal Screen Scrolling On the CBM/PET

Horizontal scrolling is a convenient method of displaying graphic functions that are too wide to fit on a PET screen. Using only the standard character set, a dramatic increase in resolution is possible.

John E. Girard
676 Alma St. #202
Oakland, California 94610

Long ago I stopped complaining about PET graphic resolution. In most cases it is adequate, and when it isn't adequate, there are always the lines (8 per cell), quarter-boxes and scroll plotting. That's right... *scroll plotting*. If I have left you in the dark, then consider this: If a graph, for example, is cramped and unreadable, then scale it much larger and let it roll past you, like a program listing. The only problem is one of orientation. We expect events to occur from side to side; the built-in scroll feature causes them to occur from down to up at a 90 degree rotation! I chose to solve this problem.

The result was a simple machine language program which moves the contents of the screen, 1 column to the left, whenever called by SYS 826. The program owes its brevity to the use of these "extended ASCII" cursor movement characters.

ASCII Value	Function
$13	cursor home
$1D	cursor right
$14	cursor delete
$0D	carriage return/line feed

The PET routine, called through $FFD2, prints the ASCII character of the accumulator value at current cursor position.

```
100 REM HORIZONTAL SCROLLER/PLOTTER
110 REM WRITTEN BY JOHN GIRARD
120 FORI= 826 TO 856 :READDC:POKEI,DC:NEXT
130 FORI=1TO4:READPD:P(I)=PD:NEXTI
140 FORI=1TO9:READL:PH(I)=L:NEXT
150 DATA169,19,32,210,255,170,169,29
160 DATA32,210,255,169,20,32,210,255
170 DATA169,13,32,210,255,202,224,0,208
180 DATA236,96,0,76,58,3
190 REM PLOTTING CHARACTER DATA
200 DATA 123,126,108,124
210 DATA100,100,82,70,64,67,68,69,99
220 PRINT"▒"
230 PRINT"▒▒DO YOU WISH ▒Q▒UARTER BOX OR"
240 PRINT"▒▒H▒ORIZONTAL LINE PLOTTING CHARACT
    ERS?"
250 GETQ$:IFQ$="Q"THENQ=1:GOTO280
260 IFQ$<>"H"THEN230
270 Q=2
280 PRINT"▒":SYS826:PRINT
290 FORI=1TO39:PRINT"-";:NEXT:PRINT:PRINTTAB(
    15)"╱ ╲            ╱ ╲"
300 PRINT"FUNCTION = SIN( ⊥ /2) * COS( ⊥ /18)
310 Y=9*(1-((SIN(M/2)*COS(M/18)))):Y2=Y
320 Y2=-1*SIN(M/2)*COS(M/18)
330 M=M+1:IFM>55THENM=0
340 ONQGOSUB390,460:SYS826
350 M$=STR$(M):IFM=0THENM$="   0"
360 PRINT"▒▒▒▒╱ ╲"
370 IFSGN(Y2)=-1THENPRINT" ⊥ ="M$,"AMP= ▒"STR
    $(INT((Y2*100)+.5)/100);:GOTO310
380 PRINT" ⊥ ="M$,"AMP= "INT((Y2*100)+.5)/100
    ;:GOTO310
390 REM Q BOX PLOT SUBROUTINE
400 IFY-INT(Y)>.5THENC=2
410 IFY-INT(Y)<=.5THENC=1
420 IFSGN(Y-OY)=1THENC=C+2
430 POKE33526-INT(Y)*40,P(C)
440 OY=Y
450 RETURN
460 REM HORIZ LINE PLOT SUBROUTINE
470 LL=1+INT(9*(Y-INT(Y)))
480 POKE33526-INT(Y)*40,PH(LL)
490 OY=Y
500 RETURN
```

```
          HORIZONTAL SCROLLER
033A A913      LDA #13
033C 20D2FF    JSR FFD2  :CURSOR HOME
033F AA        TAX       :PUT 19 IN X REG
0340 A91D      LDA #1D
0342 20D2FF    JSR FFD2  :CURSOR RIGHT
0345 A914      LDA #14
0347 20D2FF    JSR FFD2  :CURSOR DELETE
034A A90D      LDA #0D
034C 20D2FF    JSR FFD2  :CRLF
034F CA        DEX
0350 E000      CPX #00   :DONE 19 TIMES?
0352 D0EC      BNE 0340  :NO...DO AGAIN
0354 60        RTS       :RETURN TO BASIC
0355 00        BRK
0356 4C3A03    JMP 033A
```

The program starts by sending the cursor *home*. Next, the cursor is moved to the second column, top line. A *delete* is performed; this shifts the top line display to the left by one column. The cursor moves down to the next line, and the process is repeated 18 more times. The bottom 6 lines are untouched and may be used as a text window. The demonstration program, as written, will run on old and upgraded ROM CBM/PETs. I have included the option to plot either horizontal lines or the quarter-boxes. All plotting is done in the 37th column, thus the plotting subroutines are short, simple, and extremely *fast*.

As research associates in Lecture Demonstrations, John Girard and Loren Wright (MICRO's PET Vet) developed more than two dozen college-level physics programs at Berkeley. Mr. Girard is now training for systems analysis on the Burroughs 7800 system at Pacific Telephone Headquarters, San Francisco.

MICRO

Integer Flash for the Apple

It is possible to produce flashing characters in Integer BASIC, but you will need to understand some underlying mechanisms.

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

Have you ever been irked by the lack of an Apple II Integer BASIC FLASH statement? Have you ever wondered why the Integer BASIC manual tells you how to produce inverse video (POKE 50,63), but balks at similar instructions for flashing video? Have you ever experimented, trying to find a POKE 50,V which would "work", but been forced to give up in frustration? Well, despair no more! Read on for the solution to the Integer BASIC FLASH problem.

Apple II Character Representation

The Apple II allows for 64 different characters to be displayed in TEXT mode. The representation of 64 distinct characters only requires 6 bits, but obviously 8 bits are used to store each character in memory. Thus, one could imagine up to four different "flavors" of characters, depending on what value (0-3) the 2 high order bits of the character byte happen to take on. The Apple II Reference Manual, #A2L0001A, contains a table on page 15 which shows the assignment of 8-bit "codes" to actual displaying characters. It turns out that there are only three visually distinguishable modes: NORMAL, FLASHING, and INVERSE.

The codes $80 through $9F are reserved for the control characters (and display as blanks), thus preventing a fourth mode, such as LOW INTENSITY. The distribution of values is shown in table 1.

Table 1

$00 - $1f	INVERSE MODE	@ through _ (underscore)
$20 - $3F	INVERSE MODE	space through ?
$40 - $5F	FLASHING MODE	@ through _
$60 - $7F	FLASHING MODE	space through ?
$80 - $9F	BASIC Control Characters (No Display)	
$A0 - $BF	NORMAL MODE	space through ?
$C0 - $DF	NORMAL MODE	@ through _
$E0 - $FF	Extra codes: Normally will not occur in BASIC. If they are fed to COUT, they display as NORMAL MODE characters space through ?.	

Listing 1

```
   5 GOSUB 1000
  10 TEXT : CALL -936
  15 VTAB 8: TAB 1
  20 FOR I=0 TO 255
  25 POKE 0,I
  30 CALL 1
  35 NEXT I
  99 END
1000 REM  POKE IN THE COUT
1001 REM  INTERFACE SUBROUTINE
1002 REM
1005 POKE 1,165
1006 POKE 2,0
1007 POKE 3,32
1008 POKE 4,237
1009 POKE 5,253
1010 POKE 6,96
1019 RETURN
```

Listing 2

```
  5 KBD=-16384:CLR=-16368:WAIT=500:SHOWIT=100
 10 REM  TEST POKE 50,VALUE
 11 REM  FOR DIFFERENT VALUES
 12 REM  OF "VALUE"!!?
 13 REM
 14 REM  !"#$%&'()*
 15 REM  @ABCDEFGHI
 16 REM  0123456789
 17 REM
 20 FOR I=0 TO 255 STEP 8
 25 POKE 50,I: GOSUB SHOWIT
 30 GOSUB WAIT
 35 NEXT I
 90 POKE 50,255: LIST
 99 END
100 LIST : RETURN
500 KEY= PEEK (KBD)
505 IF KEY<128 THEN RETURN
510 POKE CLR,0
515 KEY= PEEK (KBD): IF KEY<128 THEN 515
520 POKE CLR,0: RETURN
```

The curious individual who wishes to "verify" this table may seek a way to display all the codes from 0 to 255 on the screen. The Apple II Monitor contains the routine COUT, which will place the value of the *code* in the 6502 accumulator onto the next available screen location. The trick is to use a machine language interface routine, which guarantees that a given value will be in the accumulator. This may be accomplished as follows: First POKE the following routine into memory (I have used PAGE 0):

```
LDA $00
JSR COUT ($FDED)
RTS
```

Then use the Integer BASIC statements:

```
POKE 0,I
CALL 1 (assuming you POKEd
starting at location 1)
```

to display the value I. Listing 1 illustrates the application of this approach to produce the desired display of all possible character codes in the order 0 to 255. Run the program to verify the Apple Reference Manual's description.

Quirks in the Character Assignments

In the "normal" ASCII code, the character codes for space through ? *precede* the character codes for @ through __. This relationship is maintained in the NORMAL mode of the Apple II display. However, for both the INVERSE mode and the FLASHING mode, this relationship is *reversed*: the codes for INVERSE space through INVERSE ? *follow* rather than precede the codes for INVERSE @ through INVERSE __. The same relationship holds for the FLASHING mode. Let's see what we may discover about the implications this may hold for the use of location 50 in Integer BASIC.

Page 32 of the Apple II Reference Manual tells us how location 50, the so-called Normal/Inverse Mask location, is used by COUT. Except for control characters, a logical AND is performed between the outgoing character and the value in location 50. If the outgoing character "came from" BASIC, it will be a character with code between $A0 and $DF. Using the value 255 as a mask will preserve all bits of the original code, whereas using the value 63 as a mask will "strip off" the 2 high order bits of the original code. Codes between $A0 and $DF will be transformed to codes between $00 and $3F. But, let's look at that a little more carefully! The values between $A0 and $BF are taken into the values between $20 and $3F, *not* the values between $00 and $1F. Thus @ through __ become INVERSE @ through INVERSE __, and " " (space) through ? become INVERSE " " through INVERSE ?. Figure 1 illustrates this transformation.

Now suppose location 50 contains the number 127. Performing a logical AND of this value with a character code will remove only the *most significant* bit. This will produce exactly the same result as before for the codes $A0 through $BF; consequently, space through ? will be displayed in INVERSE mode. However, for the codes $C0 through $DF the resulting values will now be $40 through $5F. That means that @ through __ will be displayed in FLASHING mode.

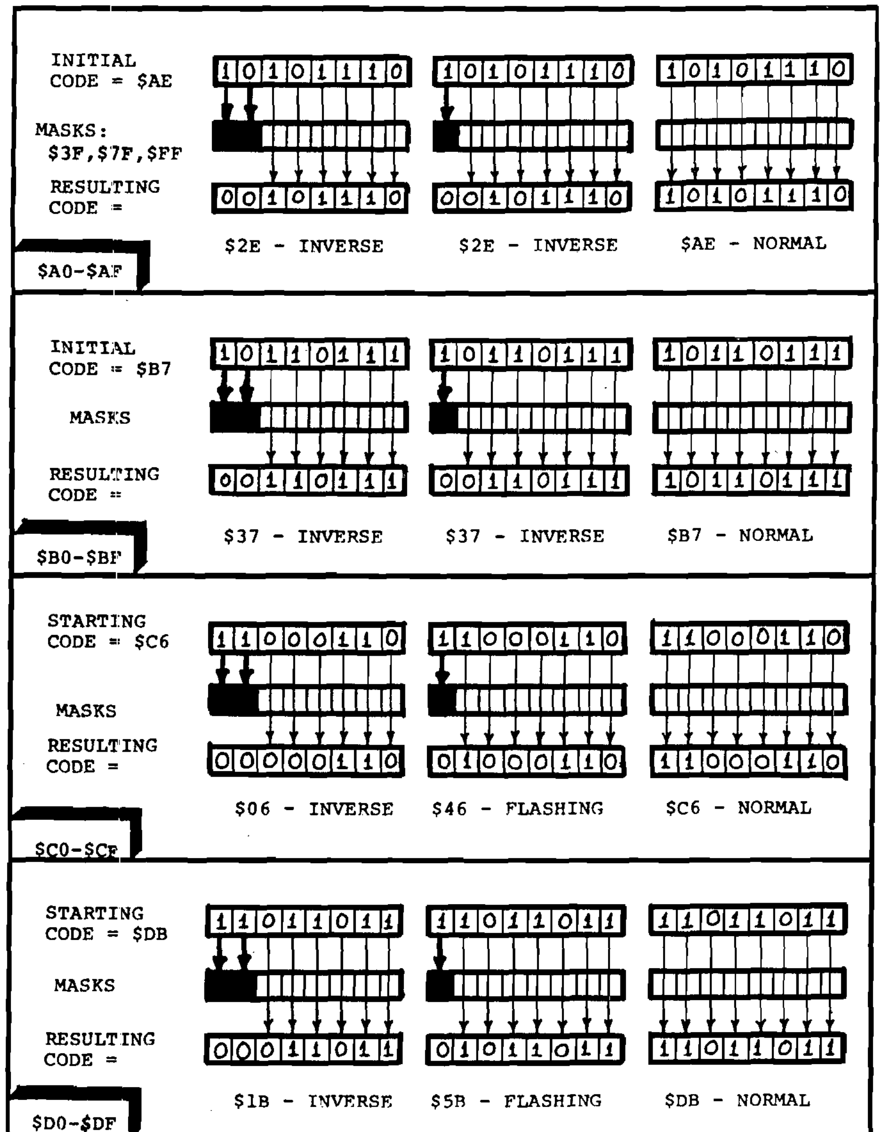


Figure 1 - Integer BASIC Location 50 Manipulations

Listing 3

```
2000 REM   PRINT A FLASHING CHARACTER
2001 REM
2005 IF ASC(CH$)<= ASC("?") THEN POKE 0, ASC(CH$)-64
2010 IF ASC(CH$)> ASC("?") THEN POKE 0, ASC(CH$)-128
2015 CALL 1
2019 RETURN
```

Listing 4

```
0800                ;********************
0800                ;*                  *
0800                ;* FLASH SUBROUTINE *
0800                ;*                  *
0800                ;*  BY RICARD VILE  *
0800                ;*                  *
0800                ;********************
0800                ;*
0800                COUT1  EQU $FDF0
0800                ;
0001                       ORG $1
0001                       OBJ $800
0001                ;
0001 C9A0           FLASH  CMP #$A0          ;CHECK FOR CONTROL CHARACTERS
0003 B003                  BCS $08           ;GO ON IF NOT
0005 4CF0FD                JMP COUT1         ;OTHERWISE GOTO COUT1 RIGHT AWAY
0008 C9C0                  CMP #$C0          ;IS IT BIGGER THAN @?
000A B006                  BCS $12           ;YES
000C 38                    SEC               ;NO
000D E940                  SBC #$40          ;CONVERT BY SUBTRACTING 64
000F 4CF0FD                JMP COUT1
0012 E980                  SBC #$80          ;CONVERT BY SUBTRACTING 128
0014 4CF0FD                JMP COUT1
                           END
```

Listing 5

```
  10 GOSUB 1000
  15 GOSUB FLASH: PRINT "HI"
  20 GOSUB REGULAR: PRINT "HI"
  99 END
1000 REM  POKE IN THE FLASHIT
1001 REM  SUBROUTINE
1002 REM
1005 POKE 1,201
1006 POKE 2,141
1007 POKE 3,208
1008 POKE 4,3
1009 POKE 5,76
1010 POKE 6,240
1011 POKE 7,253
1012 POKE 8,201
1013 POKE 9,192
1014 POKE 10,176
1015 POKE 11,6
1016 POKE 12,56
1017 POKE 13,233
1018 POKE 14,64
1019 POKE 15,76
1020 POKE 16,240
1021 POKE 17,253
1022 POKE 18,233
1023 POKE 19,128
1024 POKE 20,76
1025 POKE 21,240
1026 POKE 22,253
1030 FLASH=1050:REGULAR=1075
1049 RETURN
1050 POKE 54,1: POKE 55,0: RETURN
1075 POKE 54,189: POKE 55,158: RETURN
1099 RETURN
```

Listing 6

```
   5 DIM MSG$(40)
   6 DOSCMD=500
  10 D$="": REM  CONTROL-D
  15 TEXT : CALL -936
  20 INPUT "STARTING LINE NUMBER FOR POKES ",SPOKE
  25 PRINT "STARTING LINE NUMBER FOR FLASH"
  26 INPUT "SUBROUTINE ",SFLASH
  30 MSG$="OPEN INTEGER FLASH": GOSUB DOSCMD
  35 MSG$="WRITE INTEGER FLASH": GOSUB DOSCMD
  40 PRINT SPOKE;" REM POKE IN THE FLASHIT"
  41 PRINT SPOKE+1;" REM SUBROUTINE"
  42 PRINT SPOKE+2;" REM"
  45 PRINT SPOKE+5;" POKE 1,165"
  46 PRINT SPOKE+6;" POKE 2,0"
  47 PRINT SPOKE+7;" POKE 3,32"
  48 PRINT SPOKE+8;" POKE 4,237"
```

(continued)

Placing values other than 63, 127, or 255 into location 50 will cause some of the significant bits of the character code itself to be dropped by COUT before display. The results can be amusing. Try the program in listing 2, for example, or do a POKE 50,254 on an unsuspecting friend's Apple (be sure to stay around to undo the chaos, or you may lose a friend!).

Conversion Factors — Normal to Flashing

Now that we see that location 50 cannot be used to solve the problem, we shall have to find another way. We already have a machine language interface to the COUT routine, as suggested above. What we need now is an Integer BASIC routine to POKE the correct values into location 0 for each character we might wish to print. An inefficient way to do this would be to create a translation table, i.e., an array with one entry for each normal mode character (codes $A0 to $DF). The value stored in each array location would be the code for the corresponding flashing character. Thus, if we name the array FLASH, FLASH(1) would contain 32, FLASH(2) would contain 33, ... ,FLASH(33) would contain 64, FLASH(34) would contain 65, and so on. There is a much easier way, however.

It is based on the observation that the set of 64 characters comes in two 32 character "chunks"—space through ? and @ through _. There is a fixed relationship between normal characters and their corresponding flashing equivalents in each chunk. We can deduce this relationship by comparing the codes for the first character in each chunk:

FLASHING space = 32
NORMAL space = 160
160 − 32 = 128
FLASHING @ = 64
NORMAL @ = 128
128 − 64 = 64

This tells us that the common conversion factor for space through ? is 128 and for @ through _ it is 64. The code for the conversion routine then almost writes itself. Just pick off one character at a time from any string we wish to convert and feed it to the conversion factors! This is exemplified in listing 3.

To use the techniques presented so far in an Integer BASIC program, you should include the two subroutines to POKE the machine language interface (starting at line 1000 of listing 1) and to decimate character strings (listing 3). GOSUB 1000 should be used to initialize the interface and code such as the following:

MSG$ = "THIS IS A MESSAGE!!"
GOSUB 2000

should be used to produce inverse messages.

A Faster Technique — Using CSW

The Apple II Monitor kindly provides a way to augment or to totally replace the COUT (Character **OUT**) subroutine. The COUT subroutine begins with the instruction:

JMP (CSWL)

This indicates an indirect jump to the address stored in the Page Zero locations CSWL and CSWH ($36,$37). When the Apple II is in normal screen mode, these locations contain the address of the instruction immediately following the JMP instruction itself. This means that COUT normally continues by jumping to its own code. However, since CSWL and CSWH are locations in RAM instead of ROM, any running program may replace their values at its convenience (we hope not at its peril!). This occurs, for example, when a PR#1 statement is used to select a printer for output. It also occurs each time the Apple II DOS transfers a character to the disk.

The Integer BASIC PRINT statement causes a character at a time to arrive at the portals of the COUT subroutine carried by the 6502 AC. Thus, we may assume that the accumulator is already "set up" when the JMP (CSWL) instruction is executed. How can we make use of this? We simply write a routine which checks the value of the incoming character to see if it is smaller than or larger than the @ character (code = $C0) and convert it accordingly (as did the Integer BASIC subroutine presented earlier). One small detail—we shall have to check first for control characters, since those should not be translated. The machine language code is shown in the assembly language program of listing 4.

By POKEing this routine instead of our original one, the need is removed for the second Integer BASIC subroutine. To turn on the FLASH mode, use the statements:

POKE 54,1 : POKE 55,0

Listing 6 *(continued)*

```
 49 PRINT SPOKE+9;" POKE 5,253"
 50 PRINT SPOKE+10;" POKE 6,96"
 59 PRINT SPOKE+19;" RETURN"
100 PRINT SFLASH;" REM PRINT A FLASHING CHARACTER"
101 PRINT SFLASH+1;" REM"
105 PRINT SFLASH+5;" IF ASC(CH$)<=191 THEN POKE 0,ASC(CH$)-64"
110 PRINT SFLASH+10;" IF ASC(CH$)>191 THEN POKE 0,ASC(CH$)-128"
115 PRINT SFLASH+15;" CALL 1"
119 PRINT SFLASH+19;" RETURN"
120 MSG$="CLOSE INTEGER FLASH": GOSUB DOSCMD
125 END
500 PRINT D$;MSG$: RETURN
```

Listing 7

```
  5 DIM MSG$(40)
  6 DOSCMD=500
 10 D$="": REM  CONTROL-D
 15 TEXT : CALL -936
 20 INPUT "STARTING LINE NUMBER FOR POKES ",SPOKE
 30 MSG$="OPEN INTEGER FLASH2": GOSUB DOSCMD
 35 MSG$="WRITE INTEGER FLASH2": GOSUB DOSCMD
 40 PRINT SPOKE;" REM POKE IN THE FLASHIT"
 41 PRINT SPOKE+1;" REM SUBROUTINE"
 42 PRINT SPOKE+2;" REM"
 45 PRINT SPOKE+5;" POKE 1,201"
 46 PRINT SPOKE+6;" POKE 2,160"
 47 PRINT SPOKE+7;" POKE 3,176"
 48 PRINT SPOKE+8;" POKE 4,3"
 49 PRINT SPOKE+9;" POKE 5,76"
 50 PRINT SPOKE+10;" POKE 6,240"
 51 PRINT SPOKE+11;" POKE 7,253"
 52 PRINT SPOKE+12;" POKE 8,201"
 53 PRINT SPOKE+13;" POKE 9,192"
 54 PRINT SPOKE+14;" POKE 10,176"
 55 PRINT SPOKE+15;" POKE 11,6"
 56 PRINT SPOKE+16;" POKE 12,56"
 57 PRINT SPOKE+17;" POKE 13,233"
 58 PRINT SPOKE+18;" POKE 14,64"
 59 PRINT SPOKE+19;" POKE 15,76"
 60 PRINT SPOKE+20;" POKE 16,240"
 61 PRINT SPOKE+21;" POKE 17,253"
 62 PRINT SPOKE+22;" POKE 18,233"
 63 PRINT SPOKE+23;" POKE 19,128"
 64 PRINT SPOKE+24;" POKE 20,76"
 65 PRINT SPOKE+25;" POKE 21,240"
 66 PRINT SPOKE+26;" POKE 22,253"
 67 PRINT SPOKE+30;" FLASH=";SPOKE+50;":REGULAR=";SPOKE+75
 68 PRINT SPOKE+49;" RETURN"
 69 PRINT SPOKE+50;" POKE 54,1:POKE 55,0:RETURN"
 70 PRINT SPOKE+75;" POKE 54,189:POKE 55,158: RETURN"
120 MSG$="CLOSE INTEGER FLASH2": GOSUB DOSCMD
125 END
500 PRINT D$;MSG$: RETURN
```

Listing 8

```
 10 TEXT : CALL -936
 15 GOSUB 1000: GOSUB FLASH
 20 VTAB 8
 25 TAB 14: GOSUB 100
 26 TAB 14: GOSUB 110
 27 TAB 14: GOSUB 110
 28 TAB 14: GOSUB 120
 29 TAB 14: GOSUB 110
 30 TAB 14: GOSUB 110
 31 TAB 14: GOSUB 100
 90 GOSUB REGULAR
 99 END
100 GOSUB FLASH: PRINT " ";
101 GOSUB REGULAR: PRINT "    ";
102 GOSUB FLASH: PRINT " ";
103 GOSUB REGULAR: PRINT "  ";
104 GOSUB FLASH: PRINT "      "
109 RETURN
110 GOSUB FLASH: PRINT " ";
111 GOSUB REGULAR: PRINT "    ";
112 GOSUB FLASH: PRINT " ";
113 GOSUB REGULAR: PRINT "     ";
```

```
 114 GOSUB FLASH: PRINT " "
 119 RETURN
 120 GOSUB FLASH: PRINT "     ";
 121 GOSUB REGULAR: PRINT "     ";
 122 GOSUB FLASH: PRINT " "
 129 RETURN
1000 REM POKE IN THE FLASHIT
1001 REM SUBROUTINE
1002 REM
1005 POKE 1,201
1006 POKE 2,160
1007 POKE 3,176
1008 POKE 4,3
1009 POKE 5,76
1010 POKE 6,240
1011 POKE 7,253
1012 POKE 8,201
1013 POKE 9,192
1014 POKE 10,176
1015 POKE 11,6
1016 POKE 12,56
1017 POKE 13,233
1018 POKE 14,64
1019 POKE 15,76
1020 POKE 16,240
1021 POKE 17,253
1022 POKE 18,233
1023 POKE 19,128
1024 POKE 20,76
1025 POKE 21,240
1026 POKE 22,253
1030 FLASH=1050:REGULAR=1075
1049 RETURN
1050 POKE 54,1: POKE 55,0: RETURN
1075 POKE 54,189: POKE 55,158: RETURN
```

Listing 8 B

```
  10 GOSUB 1000: REM ESTABLISH FLASH COMMAND
  15 GOSUB FLASH: REM TURN IT ON
  18 CALL -936
  19 N=1
  20 FOR I=1 TO N
  25 FOR I=0 TO N
  30 R= RND (23)+1:C= RND (39)+1: VTAB R: TAB C: PRINT " ";
  35 NEXT I
  40 CALL -936
  45 N=N+1: IF N=1000 THEN END
  50 GOTO 20
1000 REM POKE IN THE FLASHIT
1001 REM SUBROUTINE
1002 REM
1005 POKE 1,201
1006 POKE 2,160
1007 POKE 3,176
1008 POKE 4,3
1009 POKE 5,76
1010 POKE 6,240
1011 POKE 7,253
1012 POKE 8,201
1013 POKE 9,192
1014 POKE 10,176
1015 POKE 11,6
1016 POKE 12,56
1017 POKE 13,233
1018 POKE 14,64
1019 POKE 15,76
1020 POKE 16,240
1021 POKE 17,253
1022 POKE 18,233
1023 POKE 19,128
1024 POKE 20,76
1025 POKE 21,240
1026 POKE 22,253
1030 FLASH=1050:REGULAR=1075
1049 RETURN
1050 POKE 54,1: POKE 55,0: RETURN
1075 POKE 54,189: POKE 55,158: RETURN
```

To turn it off (return to NORMAL mode), use the statements:

```
POKE 54,189 : POKE 55,158
```

Listing 5 shows the new POKE routine, together with two subroutines implementing the above switching processes. Now to turn on FLASH mode, simply say:

```
GOSUB FLASH
```

and to turn it back off, say:

```
GOSUB REGULAR
```

(Integer BASIC will not allow us to say NORMAL = 1075, since the identifier NORMAL contains the reserved word OR!).

Putting FLASH to Work

Now that you know *how* to FLASH, you certainly will want to use it. One slightly annoying feature of this is that you must key in the subroutines before using them. The line numbers I have chosen to use, may clash with those in your program. If you have a DISK system, you can use the EXEC facility to ease the load.

Listings 6 and 7 show programs that will create textfiles containing the subroutines presented. These programs will prompt you for the desired STARTING LINE NUMBERS of the subroutines. When they finish, you should have a file called either INTEGER FLASH or INTEGER FLASH2, depending on which technique you choose to employ. To include the subroutine(s) in your program, you simply use the EXEC command. For example,

```
> LOAD MYPROGRAM
> EXEC INTEGER FLASH2
```

The EXEC command will not overwrite the program you loaded with the LOAD MYPROGRAM command, but rather add in the lines it contains, just as if you had typed them from the keyboard yourself. It's a great time saver! By this approach you are not always limited to using the same line numbers for the FLASH subroutines. Simply rerun the textfile-creating program and specify new line numbers.

Using the FLASH Feature in Your Programs

No doubt you already have many useful applications of the FLASH mode in titles and prompts. For your extra enlightenment, try the program of listing 8 and enjoy!

MICRO

Polled Keyboard for C1P/Superboard

By continuously interrogating the keyboard it is possible to generate both upper and lower case characters on OSI's C1P/Superboard microcomputer.

Michael J. Alport
5 Woodland Mounds Rd.
Iowa City, Iowa 52240

I was pleased to find, in a recent issue of MICRO (22:17), an article by Edward H. Carlson describing a program which would enable the OSI keyboard to operate as an ordinary typewriter. I had been thinking of writing such a program, to be used in conjunction with a word processor, for some time, and the prospect of having a debugged program which only had to be keyed in looked attractive. My joy was short-lived, however, when I realized that Edward Carlson's program had been written for the 542 board and would not work with the 600 board found in the C1P/Superboard microcomputer. The difference between the two boards is quite simple. Instead of polling the rows/columns with a byte consisting of a combination of seven 0's and a 1, the 600 board uses a combination of seven 1's and a 0. I suspect that a simple fix would be to replace all Mr. Carlson's

STA $DF00

and

LDA $DF00

instructions with

JSR $FCBE

and

JSR $FCCF

respectively. These are monitor routines which use an EOR #$FF to invert the bit pattern, replacing 1's with 0's and vice versa. However, it is

```
 10 DF00=          KYPORT =$DF00
 20 7E00                  *=$7E00
 30 7E01           XREG   *=*+1
 40 7E02           CTRL   *=*+1
 50 7E03           LOC    *=*+1
 60 7E03 20187E ENTER  JSR KEYBRD    MAIN ROUTINE
 70 7E06 8D027E        STA LOC       SAVE FOR RPT KEY
 80 7E09 202DBF        JSR $BF2D     PRINT CHARACTER
 90 7E0C 20027F        JSR DELAY
100 7E0F 20F07E        JSR KYDONE   KEY DEPRESSED?
110 7E12 20027F        JSR DELAY
120 7E15 4C037E LOOP   JMP ENTER
130 7E18 D8     KEYBRD CLD
140 7E19 A2FE          LDX #254     CHECK CTRL ROW
150 7E1B 8E00DF        STX KYPORT
160 7E1E AE00DF        LDX KYPORT
170 7E21 8E017E        STX CTRL     SAVE UNTIL LATER
180 7E24 E0FE          CPX #254     SHIFT LOCK?
190 7E26 D004          BNE CONT     UP, CONTINUE
200 7E28 20EDFE        JSR $FEED    DOWN
210 7E2B 60            RTS
220 7E2C E07F   CONT   CPX #127     REPEAT?
230 7E2E D004          BNE NREP     NO
240 7E30 AD027E        LDA LOC      RETURN WITH LAST CHARACTER
250 7E33 60            RTS
260 7E34 E0DF   NREP   CPX #223     ESC?
270 7E36 D003          BNE CHAR     YES, RETURN WITH $1B
280 7E38 A91B          LDA #$1B
290 7E3A 60            RTS
300 7E3B A007   CHAR   LDY #7       SET UP ROW COUNT
310 7E3D 88     ROW    DEY          BEGIN ROW SEARCH
320 7E3E 30D8          BMI KEYBRD   NO CHARACTER, TRY AGAIN
330 7E40 A207          LDX #7       SET UP COL. COUNT
340 7E42 CA     COL    DEX          BEGIN COLUMN SEARCH
350 7E43 30F8          BMI ROW
360 7E45 B9E97E        LDA MASK,Y   LOAD MASK BYTE
370 7E48 8D00DF        STA KYPORT
380 7E4B AD00DF        LDA KYPORT
390 7E4E DDE97E        CMP MASK,X   COMPARE WITH MASK BYTE
400 7E51 F003          BEQ CALC     MATCH FOUND
410 7E53 4C427E        JMP COL
420 7E56 8E007E CALC   STX XREG     SAVE COL. COUNT
430 7E59 A900          LDA #0       CALC. CHAR. POSITION
440 7E5B 18            CLC
450 7E5C 88     AGAIN  DEY
460 7E5D 3005          BMI ADDX
470 7E5F 6907          ADC #7
480 7E61 4C5C7E        JMP AGAIN
490 7E64 6D007E ADDX   ADC XREG
500 7E67 AA            TAX
510 7E68 AD017E        LDA CTRL     CHECK FOR SHIFT
520 7E6B 2906          AND #6
530 7E6D C906          CMP #6
540 7E6F F005          BEQ NSHIFT   NOT SHIFT
550 7E71 18            CLC          SHIFT-ADD 49 TO CHAR. POINTER
560 7E72 8A            TXA
570 7E73 6931          ADC #49
580 7E75 AA            TAX
590 7E76 BD877E NSHIFT LDA CHARTB,X LOOK UP CHAR. TABLE
600 7E79 AA            TAX
610 7E7A AD017E        LDA CTRL     CTRL?
620 7E7D 2940          AND #$40
630 7E7F D004          BNE NCTRL    NO
640 7E81 8A            TXA
650 7E82 0980          ORA #$80     YES, SET BIT 7
```

sometimes easier to rewrite a complete program than to attempt to modify someone else's. So while I was rewriting the program, I took the opportunity to add a number of features which were not included in the original program.

The program itself should be self-explanatory, especially when read in conjuntion with Mr. Carlson's article. I will, however, make a few comments about the additional features included in my program.

The shift-lock key is continually polled to determine whether it is in the up or down position. If it is in the down position, control is transferred to the normal monitor keyboard routine beginning at $FEED. If the shift-lock is up, the new keyboard routine is executed. This makes it posible to use the new keyboard routine in conjunction with BASIC by placing the address of this keyboard routine in BASIC's input vector location.

I found it necessary to add a delay routine (in addition to the original KYDONE routine) to eliminate excessive contact bounce found on my keyboard. It may be possible to omit this routine on other keyboards.

Michael J. Alport's interest in microcomputing began about two years ago and since then he has been spending half his spare time designing a super I/O board, writing graphics software, and discovering the tremendous potential of FORTH, and the other half trying to decide why he finds microcomputing so exciting. His professional interest lies in plasma physics.

MICRO

```
660 7E84 60                RTS
670 7E85 8A        NCTRL   TXA
680 7E86 60                RTS
690 7E87 31        CHARTB .BYTE '1234567890:-',$7F,' .lo',$0A,$0D,'  '
690 7E88 32                  690 7E92 2D
690 7E89 33                  690 7E93 7F
690 7E8A 34                  690 7E94 20
690 7E8B 35                  690 7E95 2E
690 7E8C 36                  690 7E96 6C
690 7E8D 37                  690 7E97 6F
690 7E8E 38                  690 7E98 0A
690 7E8F 39                  690 7E99 0D
690 7E90 30                  690 7E9A 20
690 7E91 3A                  690 7E9B 20

700 7E9C 77               .BYTE 'wertyuisdfghjkxcvbnm,'
700 7E9D 65                  700 7EA7 68
700 7E9E 72                  700 7EA8 6A
700 7E9F 74                  700 7EA9 6B
700 7EA0 79                  700 7EAA 78
700 7EA1 75                  700 7EAB 63
700 7EA2 69                  700 7EAC 76
700 7EA3 73                  700 7EAD 62
700 7EA4 64                  700 7EAE 6E
700 7EA5 66                  700 7EAF 6D
700 7EA6 67                  700 7EB0 2C

710 7EB1 71               .BYTE 'gaz',$20,'/;p'
710 7EB2 61
710 7EB3 7A
710 7EB4 20
710 7EB5 2F
710 7EB6 3B
710 7EB7 70
720 7EB8 21               .BYTE '!"#$%&',$27,'()0*=',$7F,' >LO',$0A,$0D
720 7EB9 22                  720 7EC2 2A
720 7EBA 23                  720 7EC3 3D
720 7EBB 24                  720 7EC4 7F
720 7EBC 25                  720 7EC5 20
720 7EBD 26                  720 7EC6 3E
720 7EBE 27                  720 7EC7 4C
720 7EBF 28                  720 7EC8 4F
720 7EC0 29                  720 7EC9 0A
720 7EC1 30                  720 7ECA 0D

730 7ECB 20               .BYTE '  WERTYUISDFGHJKXCVBNM<QAZ',$20,'?+P'
730 7ECC 20                  730 7EDB 58
730 7ECD 57                  730 7EDC 43
730 7ECE 45                  730 7EDD 56
730 7ECF 52                  730 7EDE 42
730 7ED0 54                  730 7EDF 4E
730 7ED1 59                  730 7EE0 4D
730 7ED2 55                  730 7EE1 3C
730 7ED3 49                  730 7EE2 51
730 7ED4 53                  730 7EE3 41
730 7ED5 44                  730 7EE4 5A
730 7ED6 46                  730 7EE5 20
730 7ED7 47                  730 7EE6 3F
730 7ED8 48                  730 7EE7 2B
730 7ED9 4A                  730 7EE8 50
730 7EDA 4B

740 7EE9 7F        MASK   .BYTE 127,191,223,239,247,251,253
740 7EEA BF
740 7EEB DF
740 7EEC EF
740 7EED F7
740 7EEE FB
740 7EEF FD
750 7EF0 A900      KYDONE LDA #00
760 7EF2 8D00DF           STA KYPORT
770 7EF5 AD00DF           LDA KYPORT
780 7EF8 C9FF             CMP #$FF
790 7EFA D001             BNE NEXT
800 7EFC 60               RTS
810 7EFD C9FE      NEXT   CMP #$FE
820 7EFF D0EF             BNE KYDONE
830 7F01 60               RTS
840 7F02 A2FF      DELAY  LDX #$FF      DEBOUNCE ROUTINE
850 7F04 A020      LP1    LDY #$20
860 7F06 88        LP2    DEY
870 7F07 D0FD             BNE LP2
880 7F09 CA               DEX
890 7F0A D0F8             BNE LP1
900 7F0C 60               RTS
```

This issue of the Ohio Scientific Small System's Journal is devoted entirely to part two of last month's UCSD Pascal article.

User-Defined Routines in UCSD Pascal

By D.R. Turnidge

Part one of this note introduced the use of the UCSD Pascal utility routine LIBRARY.CODE to install a unit of related procedures and functions in the system library. The unit presented in part one was extremely short and composed entirely of routines written in Pascal. This part presents a more extensive unit of routines which allow the utilization of the audio and color graphics capabilities of the C4P and C8P series of Ohio Scientific computers. This unit is based upon three 6502 assembler routines. The first two of these routines, POKEXT and PEEKEXT, are minor modifications of similar routines which appear in Appendix F of *Pascal Primer* by David Fox and Mitch Waite. We thank the SAMS publishing company for permission to include these two routines here. These routines function like POKE and PEEK in BASIC and provide access to the memory-mapped features of the C4P and C8P. The third routine named SCREXT fills the screen with a specified graphics character or color.

Part Two—Assembler Subroutines

A. Creating the assembler text file PEEKPOKE

The use of the UCSD Adaptable Assembler is discussed in detail in Section 1.7 of [3]. Use the EDITOR to enter the following text and save it in a file named PEEKPOKE.TEXT. (Note: Labels must begin in column one of a source line.)

```
;..................................................
        .MACRO POP  ; a macro to pull the return
         PLA        ; address off the stack
         STA %1
         PLA
         STA %1+1
        .ENDM
;..................................................
        .MACRO PUSH ; a macro to push the return
         LDA %1+1   ; address back on the stack
         PHA
         LDA %1
         PHA
        .ENDM
;..................................................
        .FUNC PEEKEXT,1 ; this function determines the
                        ; contents of a specified memory
                        ; location
RETURN  .EQU 70 ; assigns the value 70 to the label RETURN
         POP RETURN  ; saves return address in locations 70
                     ; and 71
         PLA ; throw away four extraneous bytes of
         PLA ; data on the stack in order to get
         PLA ; at function parameter
         PLA
         PLA        ; pull the parameter (an address) off the
         STA 72     ; stack and place in locations 72 and 73
         PLA
         STA 73
         LDY #0     ; retrieve the value currently stored
         LDA @72,Y; at the specified memory address
         TAY
         LDA #0     ; place the function value (a two byte
         PHA        ; integer) on the stack before returning
         TYA        ; from function call
         PHA
         PUSH RETURN ; restore the return address to stack
         RTS
;..................................................
        .PROC POKEXT,2 ; this procedure deposits a value in
                       ; a specified memory location;
RETURN  .EQU 70
         POP RETURN
         PLA      ; pull the second parameter off the stack
         STA 76  ; (ignore high byte)-store at location 76
         PLA
         PLA      ; pull first parameter (an address) off the
         STA 74  ; stack and store at locations 74 and 75
         PLA
         STA 75
         LDY #0  ; deposit the value stored at location 76 in
         LDA 76  ; the address stored in locations 74 and 75
         STA @74,Y
         PUSH RETURN
         RTS
;..................................................
        .PROC SCREXT,2 ; this procedure fills screen with
                       ; specified character or color
RETURN  .EQU 70
SCRMEM  .EQU 208.
COLMEM  .EQU 224.
         POP RETURN
         LDA #0  ; store address of top of graphics
         STA 77  ; memory in locations 77 and 78
         LDA #SCRMEM
         STA 78
         PLA
         BEQ SCREEN
COLOR    LDA #COLMEM ; if second parameter not zero change
         STA 78      ; to address of top of color memory
SCREEN   PLA
         PLA ; first parameter contains character or
         TAX ; color number for screen fill
         PLA ; store this value in accumulator
         TXA
         LDX #0    ; enter loop to deposit value stored
         LDY #0    ; in accumulator in 2048 consecutive
NEXTPT   STA @77,Y; memory locations beginning at
         INY       ; address stored in locations 77 and 78
         CPY #0
         BNE NEXTPT
         INC 78 ; advance to next page of memory
         INX
         CPX #8      ; check to see if entire screen filled
         BNE NEXTPT ; if not, continue
         PUSH RETURN
         RTS
        .END
;..................................................
```

The next section shows how to assemble this source file. Before proceeding there are several observations which should be made.

1. The directives .PROC and .FUNC identify the beginning of assembly language procedures and functions, respectively. This file contains three routines. The stack is used to pass parameters and return function values. For a procedure call, the parameters are pushed on the stack (last in -first out) under the return address. For a function call, four extra bytes are placed on the stack above the parameters. These four bytes (which are of no value in this context) must be removed to gain access to the function parameters. The function value is returned to the host by placing it on the stack under the return address. The number 2 in the statement .PROC POKEXT,2 specifies that the procedure POKEXT has 2 parameters.

2. The UCSD Adaptable Assembler supports macro definitions. This file contains two macros, POP and PUSH.

3. Page zero memory locations 50-7F (hex) are not reserved by the system and can be used in user-written assembler routines.

B. Assembling the source file

The assembler is invoked by typing "A" in response to the system prompt line. In order for this selection to be valid, one of the disk drives must contain a disk that includes the files SYSTEM.ASSMBLER and 6500.OPCODES. These files are located on the PASCAL2: disk for mini disk systems and on the standard PASCAL: disk for 8" systems. (Note: Section 4.2 of the *UCSD Supplemental User's Document* for Ohio Scientific users describes some alternate disk configurations for mini floppy disk users. The disk labeled #5 Disk 1 should include the file 6500.OPCODES.)

The following steps will assemble PEEKPOKE. TEXT to the code file PEEKPOKE.CODE.

1. Use option N(ew in the filer to make sure the workfile is clear. Like the compiler, the assembler uses the workfile (if one is present) as its input file.

2. Type "A" in response to the system prompt line and answer both of the queries "Assemble what text?" and "To what codefile?" by entering "PEEKPOKE".

3. If you wish the console to display an assembled listing of the program during assembly enter "CONSOLE:" in response to the prompt "Output file for assembled listing:". Otherwise just enter a carriage return.

C. Using POKEXT, PEEKEXT and SCREXT in a Host Pascal program

The procedure and function declaration part of a Pascal program must include declarations for any assembly language routines which it uses. These declarations have the form of a procedure or function heading, followed by the keyword "EXTERNAL". The assembly routines in PEEKPOKE could be declared as follows:

```
PROCEDURE POKEXT(MEMLOC,DATA:INTEGER);
  EXTERNAL;

FUNCTION PEEKEXT(MEMLOC:INTEGER):INTEGER;
  EXTERNAL;

PROCEDURE SCREXT(DATA,OPTION:INTEGER);
  EXTERNAL;
```

These declarations identify these routines as assembly language routines and specify the parameters. In these procedures MEMLOC specifies a memory location for a POKE or a PEEK. This address must be expressed as a signed two's complement number between −32768 and 32767. For example, the address of the control register on the C4P and C8P at 56832 must be converted to −8704 = −(65536 - 56832). The parameter DATA in POKEXT denotes the value (in the range 0 - 255) which is to be stored at MEMLOC. SCREXT fills the entire screen with the graphics character corresponding to the value of DATA if OPTION = 0, otherwise it colors the entire screen with the color corresponding to the value of DATA. The C4P and C8P user's manuals include the appropriate character and color codes.

Before a Pascal program which uses EXTERNAL procedures and functions can be run, it must first be compiled. Then the EXTERNAL procedures and functions must be added to the code file with the LINKER (see section 1.6 of [3]).

The following section describes UNIT SPECIALFEATURES which adds these and other routines to the system library. As pointed out in part one, linking is automatic for routines placed in the system library.

D. UNIT SPECIALFEATURES

This section includes the text for a large unit containing procedures which control the color graphics and audio features of the C4P and C8P. Use the EDITOR to enter this unit and store it in a file named PLOTUNIT.TEXT.

```
(*$L CONSOLE:*)
UNIT SPECIALFEATURES;
 INTERFACE
  TYPE
   COLORS = ( YELLOW,INVYELLOW,RED,INVRED,GREEN,INVGREEN,
              OLIVE,INVOLIVE,BLUE,INVBLUE,PURPLE,INVPURPLE,
              SKYBLUE,INVSKYBLUE,BLACK,INVBLACK );

  VAR OPTIONSET: SET OF (SOUND,KOLOR,VID32 x 32) ;

  PROCEDURE POKE ( MEMLOC,DATA: INTEGER ) ;
  FUNCTION PEEK ( MEMLOC: INTEGER ) : INTEGER;
  PROCEDURE INITOPTIONS;
  PROCEDURE SOUNDON;
  PROCEDURE SOUNDOFF;
  PROCEDURE COLORON;
  PROCEDURE COLOROFF;
  PROCEDURE SCR32 x 32;
  PROCEDURE SCR32 x 64;
  PROCEDURE PLOTCHARACTER ( CHARNUM,XCOOR,YCOOR: INTEGER ) ;
  PROCEDURE ERASECHARACTER ( XCOOR,YCOOR: INTEGER ) ;
  PROCEDURE PLOTCOLOR ( COLOR:COLORS; XCOOR,YCOOR: INTEGER ) ;
  PROCEDURE ERASECOLOR ( XCOOR,YCOOR: INTEGER ) ;
  PROCEDURE FILLGRAPHICS ( CHARNUM: INTEGER ) ;
  PROCEDURE CLEARGRAPHICS;
  PROCEDURE FILLCOLOR ( COLOR:COLORS ) ;
  PROCEDURE CLEARCOLOR;
  PROCEDURE TONE ( FREQUENCY: INTEGER ) ;

 IMPLEMENTATION

  CONST (*   THESE ARE SPECIAL MEMORY ADDRESSES—
             INTEGER VALUES MUST BE EXPRESSED AS
             SIGNED TWO'S COMPLEMENT NUMBERS BETWEEN
             - 32768 and 32767 *)

   SCRTOP = - 12288;
   COLORTOP = - 8192;
   CONTROLREGISTER = - 8704;
   AUDIOPORT = - 8447;

  VAR   (* PRIVATE VARIABLES *)
   SCRLOC,COLORLOC,OPTIONCODE,XCOOR,YCOOR,
    AUDIOVALUE: INTEGER;

  (* EXTERNALLY ASSEMBLED PROCEDURE *)
  PROCEDURE POKEXT (MEMLOC1,DATA1: INTEGER) ;
   EXTERNAL;

  (* EXTERNALLY ASSEMBLED FUNCTION *)
  FUNCTION PEEKEXT (MEMLOC2: INTEGER) : INTEGER;
   EXTERNAL;

  (* EXTERNALLY ASSEMBLED PROCEDURE*)
  PROCEDURE SCREXT (OPTION,DATA1: INTEGER) ;
   EXTERNAL;

  PROCEDURE POKE; (* PUBLIC VERSION OF POKE *)
   BEGIN
    POKEXT(MEMLOC,DATA) ;
   END;

  FUNCTION PEEK; (* PUBLIC VERSION OF PEEK *)
   BEGIN
    PEEK: = PEEKEXT(MEMLOC) ;
   END;

  PROCEDURE SETOPTIONS;      (* PRIVATE PROCEDURE TO SET
                                OPTIONS BASED UPON CURRENT
                                VALUE OF OPTIONSET *)

   BEGIN
    OPTIONCODE: = 1;
    IF VID32 x 32 IN OPTIONSET THEN
     OPTIONCODE: = OPTIONCODE - 1;
    IF SOUND IN OPTIONSET THEN
     BEGIN
      OPTIONCODE: = OPTIONCODE + 2;
      POKEXT(AUDIOPORT,1) ;
     END;
    IF KOLOR IN OPTIONSET THEN
     OPTIONCODE: = OPTIONCODE + 4;
    POKEXT(CONTROLREGISTER,OPTIONCODE) ;
   END;

  PROCEDURE INITOPTIONS; (* PUBLIC PROCEDURE, TURNS COLOR
                            OFF, SOUND OFF, AND SELECTS
                            32 x 64 DISPLAY MODE *)
   BEGIN
    OPTIONSET : = [  ];
    SETOPTIONS;
   END;

  PROCEDURE SOUNDON;
   BEGIN
    OPTIONSET : = OPTIONSET + [SOUND];
    SETOPTIONS;
   END;

  PROCEDURE SOUNDOFF;
   BEGIN
    OPTIONSET : = OPTIONSET - [SOUND];
    SETOPTIONS;
   END;

  PROCEDURE COLORON;
   BEGIN
    OPTIONSET : = OPTIONSET + [KOLOR];
    SETOPTIONS;
   END;

  PROCEDURE COLOROFF;
   BEGIN
    OPTIONSET : = OPTIONSET - [KOLOR];
    SETOPTIONS;
   END;

  PROCEDURE SCR32 x 32;
   BEGIN
    OPTIONSET : = OPTIONSET + [VID32 x 32];
    SETOPTIONS;
   END;

  PROCEDURE SCR32 x 64;
   BEGIN
    OPTIONSET : = OPTIONSET - [VID32 x 64];
    SETOPTIONS;
   END;

  PROCEDURE PLOTCHARACTER; (* PUBLIC PROCEDURE, PLOTS
                              SPECIFIED GRAPHICS CHAR-
                              ACTER AT GIVEN SCREEN
                              LOCATION *)
   BEGIN
    SCRLOC: = SCRTOP + (31 - YCOOR)*64 + XCOOR;
    POKEXT(SCRLOC,CHARNUM) ;
   END;

  PROCEDURE ERASECHARACTER;
   BEGIN
    PLOTCHARACTER(32,XCOOR,YCOOR) ;
   END;

  PROCEDURE PLOTCOLOR; (*  PUBLIC PROCEDURE, PLOTS
                           SPECIFIED COLOR AT GIVEN
                           SCREEN LOCATION *)
   BEGIN
    COLORLOC: = COLORTOP + (31 - YCOOR)*64 + XCOOR;
    POKEXT(COLORLOC,ORD(COLOR)) ;
   END;

  PROCEDURE ERASECOLOR;
   BEGIN
    PLOTCOLOR(BLACK,XCOOR,YCOOR) ;
   END;

  PROCEDURE FILLGRAPHICS; (* PUBLIC PROCEDURE, FILLS
                             ENTIRE GRAPHICS DISPLAY WITH
                             SPECIFIED GRAPHICS CHARACTER *)
   BEGIN
    SCREXT(CHARNUM,0) ;
   END;

  PROCEDURE CLEARGRAPHICS; (* PUBLIC PROCEDURE, CLEARS
                              ENTIRE GRAPHICS DISPLAY
                              AREA *)
   BEGIN
    SCREXT(32,0) ;
   END;

  PROCEDURE FILLCOLOR; (* PUBLIC PROCEDURE, FILLS ENTIRE
                          COLOR DISPLAY WITH SPECIFIED
                          COLOR *)
   BEGIN
    SCREXT(ORD(COLOR),1) ;
   END;

  PROCEDURE CLEARCOLOR; (* PUBLIC PROCEDURE, CLEARS ENTIRE
                           COLOR DISPLAY AREA *)
   BEGIN
    SCREXT(ORD(BLACK),1) ;
   END;

  PROCEDURE TONE; (* PUBLIC PROCEDURE, GENERATES SPECIFIED
                     FREQUENCY USING TONE GENERATOR *)
   BEGIN
    AUDIOVALUE: =
     (24576 + FREQUENCY DIV 4) DIV (FREQUENCY DIV 2) ;
    IF AUDIOVALUE > 255 THEN AUDIOVALUE : = 255;
    POKE(AUDIOPORT,AUDIOVALUE) ;
   END;
```

The following is a brief description of each of the public procedures in this unit:

1. PROCEDURE POKE (MEMLOC,DATA: INTEGER); This procedure is essentially just the assembly procedure POKEXT described above, except that POKE is a "Pascal" program while POKEXT is an assembly routine.

2. FUNCTION PEEK(MEMLOC:INTEGER): INTEGER; Same as above for PEEKEXT.

3. PROCEDURE INITOPTIONS: Initializes the options on the C4P and C8P, turns the color and sound off, and selects the 32 x 64 display mode.

4. PROCEDURE SOUNDON; PROCEDURE SOUNDOFF; Turn the sound option on and off.

5. PROCEDURE COLORON; PROCEDURE COLOROFF; Turn the color option on and off.

6. PROCEDURE SCR32 x 32; PROCEDURE SCR32 x 64; Alternate between the 32 x 32 and 32 x 64 display mode.

7. PROCEDURE PLOTCHARACTER (CHARNUM,XCOOR,YCOOR:INTEGER); Plots the graphics character corresponding to the value of CHARNUM at the screen location with coordinates (XCOOR,YCOOR) relative to the lower left hand corner of the screen.

8. PROCEDURE ERASECHARACTER (XCOOR,YCOOR); Erases the graphics character currently stored at screen location (XCOOR,YCOOR).

9. PROCEDURE PLOTCOLOR(COLOR: COLORS;XCOOR,YCOOR:INTEGER); Plots the specified COLOR at screen location (XCOOR,YCOOR). (Note: Type COLORS is an enumerated type containing the names of all the colors available on the C4P and C8P. COLOR can have values such as YELLOW, INVYELLOW, RED, etc.)

10. PROCEDURE ERASECOLOR (XCOOR, YCOOR); Erases the color currently stored at screen location (XCOOR, YCOOR).

11. PROCEDURE FILLGRAPHICS (CHARNUM:INTEGER); PROCEDURE CLEARGRAPHICS; Allow the graphics display to be filled with the graphics character corresponding to CHARNUM or to be cleared.

12. PROCEDURE FILLCOLOR(COLOR: COLORS); PROCEDURE CLEARCOLOR; Allow the entire screen to be colored the specified COLOR or changed to BLACK.

13. PROCEDURE TONE (FREQUENCY: INTEGER); Uses the tone generator to generate a tone of the specified FREQUENCY.

E. Adding UNIT SPECIALFEATURES to the system library.

Before this unit can be added to the system library it must be compiled. This unit is fairly long and will not compile in the 48K of memory available on the C4P and C8P computers with the standard memory configuration. Section 5 of [4] describes techniques which can be used to free up more memory space. The SPECIALFEATURES unit can be compiled if the soft buffer handlers and the screen handlers are changed from memory resident to disk resident. To do this type "S" for S(ystem State in response to the command prompt line. Then enter the sequence "B","D","C","D","Q". Keyboard response following these changes is extremely sluggish, but larger programs can be compiled. (The original system state can be restored by selecting "S" and then entering the sequence "B","M","C", "M","Q".) Make these changes and then compile the contents of PLOTUNIT.TEXT to the codefile PLOTUNIT.CODE.

The utility program LIBRARY.CODE should now be used as described in part one to create a NEW.LIBRARY. This will include the contents of the current SYSTEM.LIBRARY, PLOTUNIT.CODE and PEEKPOKE.CODE. Once the NEW.LIBRARY has been created, the old SYSTEM.LIBRARY should be renamed OLD.LIBRARY, and the NEW.LIBRARY should be designated as SYSTEM.LIBRARY.

(To be continued)

MICRO Challenges

By Paul Geffen

OS-65D V3.2 Disassembly Manual

Software Consultants of Memphis, Tennessee, has produced one of the most useful pieces of documentation available for OSI floppy disk systems. Two perennial problems with Ohio Scientific small systems have been a poor disk operating system and poor documentation. This makes most assembly language programming very difficult. The user's manuals provide some information on how to use the DOS, but this material is scattered and sketchy, and does not give the assembler programmer what he really needs, which is a listing of the programs.

Software Consultants produces system software for OSI computers and so had to solve these problems. The result is a sixty-page book which contains a complete source for the kernel of OS-65D (not disk BASIC or the Assembler-Editor-Debugger). They claim to have spent 500 hours disassembling and studying this program, and the results were worth the effort. This disassembly is well commented and includes a cross-referenced symbol table.

Now a programmer can interface his own software directly to the DOS without having to spend weeks searching and deciphering the often mysterious techniques used in OS-65D. I feel that the availability of this information enhances the value of OSI small systems by allowing more powerful and efficient software to be written for these machines. This book is not, and does not claim to be, a guide to the DOS or an overview of it. It is only a listing of the source code for the program.

Software Consultants markets the following software for OSI disk systems: a cross-reference utility for BASIC programs, a Spooler/Despooler utility, a FIG Forth and a video routine. All run under OS-65D and/or other operating systems, and source code is available for all products.

Extended Monitor ROM for Superboard and C1P

The system monitor which OSI provides with the 600 board is a "glass teletype" program which doesn't even backspace. This seems out of place on a video-based machine where it would be nice to be able to move the cursor around and edit lines. And the machine level support is limited to five commands (Address mode, Data mode, Increment address, Load from tape and Go). This is only a little more useful than a programmer's panel consisting of lights and switches. Of the various alternatives available from independent sources, the only one I have tried is the BUSTEK Extended Monitor.

This is a 2K ROM which provides enhanced machine level support as well as a screen editor. The eleven machine level commands include Save to tape, Load from tape, Output (sets the save flag), Input (sets the load flag), Go, Register display, a block move, commands to display a block of memory on the screen, and load memory from the screen, and a hexadecimal calculator.

The screen editor provides a window, which allows portions of the screen to be protected from being overwritten or scrolled. The shift keys work normally as does the RUBOUT key. The REPEAT key allows data to be read from the screen into the BASIC input buffer. ESCAPE codes provide cursor up, down, left, right and home as well as clear to end of line and clear to end of screen.

Control characters move the cursor to the beginning or end of a line, insert or delete characters, cancel line and provide a graphics mode, a find character function and a pause during output.

The program does have a few problems. The most serious is the fact that there is no disk bootstrap. It was left out to make room for the extended monitor functions. This ROM can be used only on cassette-based systems. Also, the delete character function assumes a 72-character line and is meant to be used only on the last line of the display. And the insert character key can overflow the input buffer and cause the system to crash. These problems are all due to lack of space — the ROM is entirely filled with code.

The documentation for this product is very good. The 19-page user's manual contains complete operating instructions with numerous examples. In addition, it includes the addresses of 22 subroutines within the monitor and a map of the memory it uses. A complete source listing is available at extra charge. This listing has few comments and no cross-reference table.

Other monitor ROMs with improved features include the C1E and C1S ROMs from Aardvark, as well as a monitor ROM by David Anear which is available from OMEGA, an OSI user's group in Australia.

OMEGA publishes a newsletter with much hardware and software advice as well as short programs. The 81/1 issue contained OS65D notes, a single drive copier in BASIC, a batch mode program which puts a series of commands in memory and then executes them, and a program to allow named cassette files. Subscriptions are $6/year surface and $12 air mail. For more information, contact:

Geoff Cohen
72 Spofforth St.
Holt, ACT, 2615
Australia

The following user's groups have recently sent me newsletters and other information.

The Boston Computer Society has an *OSI User's Group* which meets on the third Thursday of each month at the Polaroid cafeteria in Cambridge, near MIT. Their newsletter is now five issues old and appears monthly. Write to Len Magerman, Dept. 761, 565 Tech Square - 5A, Cambridge, MA 02139 for more information.

About a year old, the *OSI North Coast User's Group*, OSINC, based in the greater Cleveland area, has formal ties with Ohio Scientific. The second issue of their newsletter contains a short "dumb" terminal program for the C4P by Aurora Software Associates. Contact President Lel Somogyi, OSINC, Three King James South, Suite 140, 24600 Center Ridge Road, Westlake, Ohio 44145. Membership is $20 for one year.

Ohio Scientific Users of New York (OSUNY) publishes OSI-tems, now in its fourth year, and one of the largest OSI newsletters. Their recent special hardware issue ran thirty pages. Write to Tom Cheng, 26 Madison St., Apt. 4I, New York, New York 10038 for more information.

AIM 65 RS-232 Interface

An optoisolated full duplex 20mA to RS-232 interface board is available, which can easily be installed with the addition of a ± 12 VDC source. Electrical connection to/from a standard RS-232 connector is shown, and several hardware and software possible problem areas are discussed.

James Guilbeau
6644 Louis XIV Street
New Orleans, Louisiana 70124

The AIM 65 computer can easily be adapted to add an RS-232 data interface at the 20 mA teletype connections. This will allow two-way data communication (without handshaking signals) for a total cost of about $25. A ± 12 VDC supply is required as well as four wires to the application connector J1. If the AIM already has ± 12 VDC, and if a 20 mA teletype would never be used, the data interface board (1½ inches square) can be mounted internally with seven wires soldered directly to the computer board.

A duplex RS-232 interface (data in/out only) can be added to the J1 application TTY connections without modification of the computer. The baud rate is selectable from as low as 110 to as high as 2400 baud. The computer can determine and save the baud rate automatically, on initialization of TTY port, with a series of delete or rubout characters.

The baud rate can also be manually set by loading hex locations $A417 (baud rate) and $A418 (delay) as described in the AIM 65 computer manual. However, the baud rate can be reset under program control if incoming data on the serial TTL port was also initiated by the program. At any one time both the serial TTL/RS-232 and the 20 mA TTY/RS-232 are at the same baud rate.

Table 1: **Connection Table**

	AIM 65 -J1	7901A	RS-232
− 12V	22	1	-
+ 12V	N	2	-
Printer	U	4	-
Keyboard	T	6	-
Printer +5V	S	7	-
Keyboard +24V	R	8	-
Ground	1	10	7 return
Data in	-	9	3 receive
Data out	-	3	2 transmit

EIA standard RS-232-C provides the electronics industry with the ground rules necessary for independent manufacturers to design and produce both data terminal and data communication equipment that conforms to a common interface requirement. As a result, a data communications system can be formed by connecting an RS-232-C data terminal to an RS-232-C data communication peripheral (such as a TTY, MODEM, computer, etc.)

The RS-232-C is a hardware standard which guarantees the following:

1. Each device on RS-232-C will use a standard 25-pin connector which will mate to another standard 25-pin of opposite sex.

2. No matter how the cables are connected, no smoke or damage will occur.

3. The data and handshake lines will each be given a specific name.

4. The RS-232-C standard calls out the interface on one end of the cable to be designated as a "Terminal" and the interface on the other end is "Data Communication Equipment." The standard defines the data handshake signals on each pin of the connector for the "Data Communication Equipment" and the "Terminal."

RS-232-C terminals and data communications equipment are not always hardware compatible. For example, the two instruments must share one of the features from each of the following characteristics:

1. Timing Format—asynchronous.

2. Transmission Mode-Simplex, (serial input) or full duplex (TTY I/O).

3. Baud Rate (bits per second)—110, 150, 300, 600, 1200.

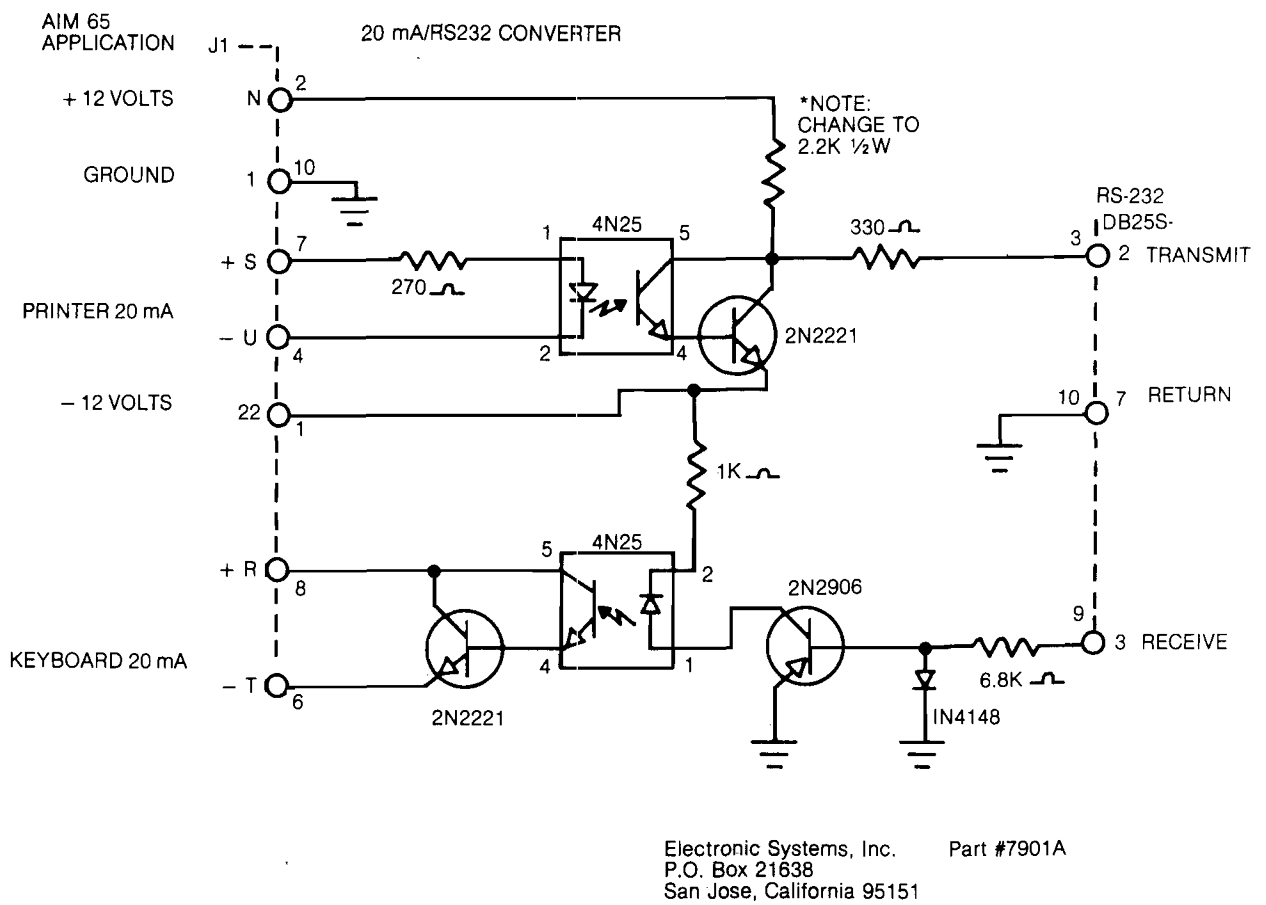
4. Bits per character (7), bits per word (11).

5. Parity Bit—low (not used).

EIA voltage levels are: 1, mark, or OFF = − 25 to − 3 VDC; 0, space, or ON = + 3 to + 25 VDC.

In serial communications, data signals usually come from one pair of lines: additional lines sometimes provide controller handshake or busy signal—used to delay data transmission until the device can handle that data. The data and handshake lines in RS-232-C send information uni-directionally (simplex); that is, one end of a cable transmits data or handshake and the other end receives data or handshake. Care must be taken to insure that each wire in RS-232-C has the appropriate transmitter and receiver combination. Transmitters connected to transmitters, and receivers connected to receivers, provide no data communication.

To alleviate this problem, care must be taken to ensure that the RS-232-C cable is correct for the application. One of the ambiguous areas in an RS-232-C connection is the use of pin 2 for transmitted data (TD) and pin 3 for received data (RD). The confusion



arises in a simplex or half-duplex connection, where pin 2 at one end of the line must go to pin 3 at the other end, and vice versa; this pin transposition can be handled in the cable itself or at either connector.

RS-232-C Cable Application Compatability Test: Measure voltage at pins 2 and 3 with ground lead connected to pin 7.

Perform Test With No Cables Connected:

"TERMINAL " (AIM 65),

pin 2   <−3V   Pin 3   0 to +2V
pin 7 GROUND.

"DATA COMMUNICATIONS DEVICE" (MODEM),

pin 2   0 to +2V   pin 3   <−3V
pin 7 GROUND.

If the computer is going to be used with various kinds of equipment, such as a printer, a modem or another computer, a double-pole, double-throw (DPDT) switch can be installed from pins 2 and 3 to reverse the data connections for the specific application.

This RS-232 installation has no provision for the "handshake" lines such as Clear to Send, Data Set Ready, Busy, etc. If these lines cannot be ignored or by-passed, an additional TTL/RS-232 interface can be used with a Peripheral Interface Adapter (PIA) and an assembly language routine to recognize the signals.

This works fine on paper. However, in practice, the user must be aware of the subtleties of serial binary data interchange to ensure that any two pieces of RS-232-C equipment will be compatible.

There are no software standards associated with RS-232-C. Many types of communication protocols serve RS-232-C systems. One protocol uses USASCII code STX (start of text) to precede data and ETX (end of text) to follow data transmission. Another uses USASCII ACK to acknowledge message receipt, and NAK to indicate no acknowledgement. This ACK/NAK combination is usually found in polling computer configurations. (STX, ETX, ACK and NAK are nonprinting characters, for "handshaking" or control only.)

20 mA/RS-232 optoisolated adapter with parts costs $15.00 (7901A) from Electronic Systems, P.O. Box 21638, San Jose, CA 95151. Not included:

10 contact PC connector: Cinch 50-10A-20 $3.00 (#10P)

25 contact RS-232 female: Cinch DB25S $5.50

Locking screws (2 each): Cinch D20418-2 60¢

For receiving RS-232 data only, a TTL/RS-232 adapter can be connected to the serial TTL input. TTL/RS-232 adapter with parts costs $10.00 (#232 A).

Note: Portions of the above discussion were extracted from John Fluke Mfg. Co. application bulletin #B0101. Used with permission.

MICRO

Real Time Clock for Superboard

By providing a brief pulse once each second to the Superboard and implementing this short program, the computer will maintain and display real time in a background mode.

James L. Mason
34 Farmington Drive
Jacobus, Pennsylvania 17407

After receiving a fuel oil bill for heating my home, I decided to monitor how long my furnace ran, the outside temperature, and the inside temperature. By taking the average temperature difference between inside and outside, and knowing how long the furnace ran over a 24 hour period (therefore the quantity of oil consumed), I could determine the heat loss of my house. I could then compute the cost effectiveness of different means to reduce heat loss.

I wanted the computer to monitor all these parameters and, therefore, I needed two temperature sensors with A/D converters and a real time clock by which the computer could keep track of elapsed time. My main program would run in BASIC for ease of number crunching, while the real time clock would run in the background. In order to accomplish this, the Real Time Clock (RTC) software would be interrupt driven.

My first task was to figure out how to interrupt the Superboard. OSI's documentation did not tell me how to do this, so I turned to MOS Technology's 6500 programming and hardware manuals. These books are extremely well written and I consider them essential for truly understanding how the computer works.

Applying a low true interrupt pulse to the Superboard's IRQ input is done at pin 2 of the expansion connector, J1.

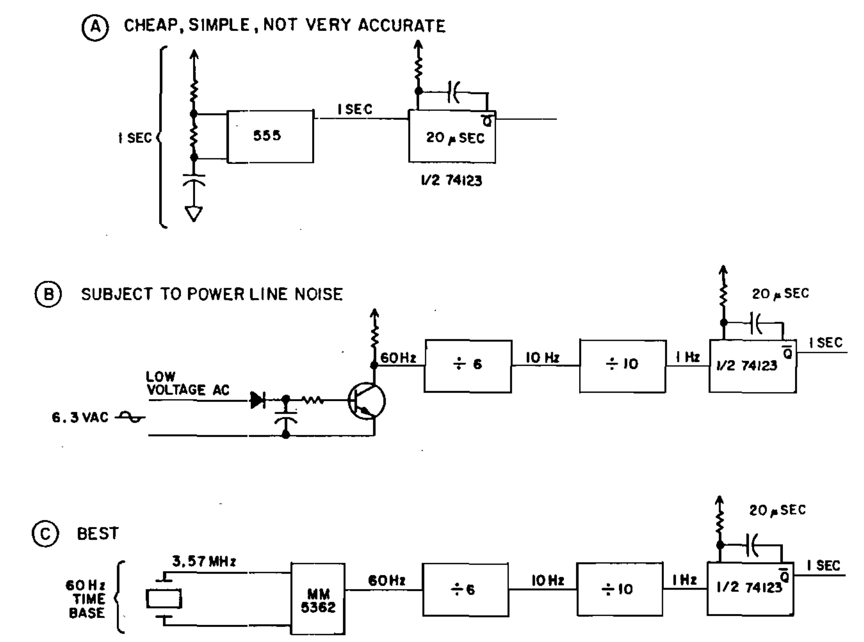


***Figure 1:* Sources of One Second Interrupt Pulses**

The pulse must be long enough so that the processor will detect the interrupt, yet shorter than the interrupt routine so that the routine won't be executed twice for the same pulse. I chose a pulse width of 20 microseconds, which was generated by one-half of a 74123 one-shot. Ballpark values for the resistor and capacitor are 20K and .002 uf respectively. I triggered the one-shot at one second intervals. See figure 1 for possible sources.

At this point if you attempt to interrupt the processor through the IRQ input, nothing will happen. This is because after a restart (whenever the "BREAK" key is pressed), initialization of 6502 automatically masks out the IRQ pin by setting the interrupt disable bit. We must clear this bit to use the IRQ input. This is done by executing the machine language instruction $58 (clear interrupt disable). I did this from BASIC by means of a USR function to call the short machine language subroutine:

LOCATION	HEX CODE	MNEMONIC
0900	58	CLI
0901	60	RTS

The USR vector is defined by the contents of locations 11 and 12 (decimal), therefore location 11 was POKEd with 0 and location 12 was POKEd with 9. Now upon execution of the BASIC instruction, X = USR(X), a low pulse applied to the IRQ pin will cause an interrupt. But to where? The IRQ vector is stored in ROM and therefore could not be changed to point directly to my RTC subroutine. However, the vector does point to a location in RAM in page one of memory that was unused according to the IP memory map. The IRQ vector points to location $01C0, so in $01C0, $01C1, and $01C2 I POKEd a machine code instruction which causes an unconditional jump to my program:

LOCATION	HEX CODE	MNEMONIC
01C0	4C	JMP
01C1	02	(lo byte)
01C2	09	(hi byte)

To use BASIC to install this:

```
POKE 448,76
POKE 449, 2
POKE 450, 9
```

Next, I wrote the machine language program which acted like a "software" counter (see figure 2). Every time the subroutine is called, a memory location representing the number of least significant seconds is incremented. If the least significant seconds' amount becomes greater than an ASCII 9 ($39), the most significant will be incremented and tested for an ASCII 6 ($36) and on down the line, thus forming a 24-hour software clock.

I thought it would be nice to have the time constantly displayed on the screen, but what about scrolling? If you put anything in video memory, it gets scrolled up the screen whenever a carriage return is performed. Luckily, the last line of the screen does *not* get scrolled. So I put the clock (6 digits plus 2 colons) in the last 8 locations of video memory.

Whenever entering an interrupt routine, it is good practice to save the working registers, execute the interrupt routine, restore the registers and finally return from the interrupt. I chose to push the registers (A, X, Y) on the stack. The return address and processor status are automatically saved by the 6502.

To put it all together, I used BASIC to load the machine code by reading a data file and POKEing. To set my USR and interrupt vectors POKEing was used again. A BASIC INPUT command was used to obtain the correct time and the hours, minutes and seconds were then POKEd into the video locations. Finally, the USR function would be executed to enable the interrupts to take effect. See figure 3.

After running the BASIC real time clock program and the time is satisfactorily ticking away, you can do a "NEW" command. The RTC will remain in the background while you write or execute new BASIC programs.

***Figure 2:* Machine Language Routine.**

```
0800                ;************************
0800                ;*                      *
0800                ;*    REAL TIME CLOCK   *
0800                ;* INTERRUPT SUBROUTINE *
0800                ;*   FOR OSI SUPERBOARD *
0800                ;*                      *
0800                ;*      BY JIM MASON    *
0800                ;*                      *
0800                ;************************
0800                ;*
0800                ZERO   EPZ $30
0800                SIX    EPZ $36
0800                COLON  EPZ $3A
0800                FOUR   EPZ $34
0800                TWO    EPZ $32
0800                ;
0800                LSS    EQU $D39B
0800                MSS    EQU $D39A
0800                LSM    EQU $D398
0800                MSM    EQU $D397
0800                LSH    EQU $D395
0800                MSH    EQU $D394
0800                ;
0900                       ORG $900
0900                       OBJ $800
0900                ;
0900 58             IRQEN  CLI               ;CLEAR INTERRUPT DISABLE BIT
0901 60                    RTS
0902                ;
0902 48             START  PHA
0903 8A                    TXA
0904 48                    PHA
0905 98                    TYA
0906 48                    PHA
0907 A93A                  LDA #COLON
0909 A236                  LDX #SIX
090B A030                  LDY #ZERO
090D EE9BD3                INC LSS           ;INCREMENT SECONDS
0910 CD9BD3                CMP LSS           ;TEST FOR >9
0913 D048                  BNE RETURN
0915 8C9BD3                STY LSS           ;SET LSS TO ZERO
0918 EE9AD3                INC MSS           ;INCREMENT TENS/SECONDS
091B EC9AD3                CPX MSS           ;TEST FOR =6
091E D03D                  BNE RETURN
0920 8C9AD3                STY MSS           ;SET MSS TO ZERO
0923 EE98D3                INC LSM           ;INCREMENT MINUTES
0926 CD98D3                CMP LSM           ;TEST FOR >9
0929 D032                  BNE RETURN
092B 8C98D3                STY LSM           ;SET LSM TO ZERO
092E EE97D3                INC MSM           ;INCREMENT TENS/MINUTES
0931 EC97D3                CPX MSM           ;TEST FOR =6
0934 D027                  BNE RETURN
0936 8C97D3                STY MSM           ;SET MSM TO ZERO
0939 EE95D3                INC LSH           ;INCREMENT HOURS
093C A234                  LDX #FOUR
093E EC95D3                CPX LSH           ;TEST FOR =4
0941 F00D                  BEQ HRS20
0943 CD95D3         HRDD   CMP LSH           ;TEST FOR >9
0946 D015                  BNE RETURN
0948 8C95D3                STY LSH           ;SET LSH TO ZERO
094B EE94D3                INC MSH           ;INCREMENT TENS/HOURS
094E 100D                  BPL RETURN
0950 A232           HRS20  LDX #TWO
0952 EC94D3                CPX MSH           ;TEST FOR =2
0955 D0EC                  BNE HRDD
0957 8C95D3                STY LSH           ;SET LSH TO ZERO
095A 8C94D3                STY MSH           ;SET MSH TO ZERO
095D 68             RETURN PLA
095E A8                    TAY
095F 68                    PLA
0960 AA                    TAX
0961 68                    PLA
0962 60                    RTS               ;DONE
```

I have found three distinct problems of concern when using the present configuration: First, since the machine language program is in RAM, it is possible for it to be overwritten as BASIC consumes more and more workspace. To prevent this, limit your BASIC memory size during the cold start. Second, recall that when the "BREAK" key is pressed, the interrupt disable flag will be set and your display cleared. Therefore, if you hit BREAK you must re-enable the interrupts, as described above.

Lastly, the target of the IRQ vector ($01C0) is in the same page of memory as the stack. I have written BASIC algorithms of such complexity that the

Figure 3: BASIC Listing of Real Time Clock Program.

```
2  REM  REAL TIME CLOCK
5  REM  BY JIM MASON
10  FOR X = 2304 TO 2402
20  READ A
30  POKE X,A
40  NEXT X
50  POKE 448,76: POKE 449,2: POKE 450,9
60  POKE 11,0: POKE 12,9
70  FOR X = 0 TO 32: PRINT : NEXT X
80  PRINT "ENTER TIME (24 HR. FORMAT)": PRINT
90  INPUT "HH,MM";H$,M$
100  FOR X = 0 TO 32: PRINT : NEXT X
110  POKE 54169,58: POKE 54166,58
120 H1$ =  LEFT$ (H$,1):H1 =  ASC (H1$): POKE 54164,H1
130 H2$ =  RIGHT$ (H$,1):H2 =  ASC (H2$): POKE 54165,H2
140 M1$ =  LEFT$ (M$,1):M1 =  ASC (M1$): POKE 54167,M1
150 M2$ =  RIGHT$ (M$,1):M2 =  ASC (M2$): POKE 54168,M2
160  POKE 54170,48: POKE 54171,48
170 X =  USR (X)
180  END
190  DATA 88,96,72,138,72,152,72,169,58,162,54,160,48,238,155
200  DATA 211,205,155,211,208,72,140,155,211,238,154,211,236,154,211
210  DATA 208,61,140,154,211,238,152,211,205,152,211,208,50,140,152
220  DATA 211,238,151,211,236,151,211,208,39,140,151,211,238,149,211
230  DATA 162,52,236,149,211,240,13,205,149,211,208,21,140,149,211
240  DATA 238,148,211,16,13,162,50,236,148,211,208,236,140,149,211
250  DATA 140,148,211,104,168,104,170,104,64
```

stack wrote into $01C0, resulting in a total system crash. Keep equations to a reasonable size or better yet, burn a new monitor ROM so that the IRQ vector points directly to the RTC interrupt subroutine. I have used the second approach with great success.

But on the good side, the time can be modified simply by POKEing the appropriate ASCII value into the proper video location. The time can be read by a BASIC program PEEKing the proper video locations. Cassette loads and saves are not affected since the interrupt subroutine is much shorter than one bit time at 300 baud.

The machine language program is relocatable if you wish to move it to a higher memory location or burn it into a ROM and stick it in the upper 32K as I did. Just remember to adjust your IRQ and USR vectors.

Editor's Note: On the AIM 65, the IRQ interrupt vector at $A400 can be used to point to a user routine like this clock. The corresponding vector on the new PET/CBM is at $0090, and on the old, $0219.

James L. Mason is currently an Electronic Engineer employed by Galt Controls. At home, he is continually developing software and hardware for the Superboard II for application as a residential utility management system.

MICRO

New Publications

(Continued from page 39)

Software

Computer Language Reference Guide With Keyword Dictionary by Harry L. Helms, Jr. Howard W. Sams & Co., Inc. (4300 West 62nd Street, Indianapolis, Indiana 46268), 1980, 110 pages, 5-3/8 × 8½ inches, paperbound.
ISBN: 0-672-21786-4 $6.95

Rather than a fast guide to learning how to program in the various computer languages, this book is a "phrase book" for the "traveler" who is outside the programming language he or she normally uses. The book assumes a working knowledge of one of the programming languages and familiarity with basic computer concepts.

CONTENTS: ALGOL (9 pages); BASIC (15); COBOL (11); FORTRAN (13); LISP (6); Pascal (11); PL/1 (11); Keyword Dictionary (21).

Software Vendor Directory by Micro-Serve, Inc. (250 Cedar Hill Avenue, Nyack, New York 10960), 1981, 196 pages, 8¼ × 11 inches in standard, hardcover, 3-ring binder. This directory of microcomputer software companies, now in its fourth edition, contains 950 software vendors and 4,195 products indexed by 200 software and 80 hardware categories. The directory lists software vendors by name, address, and telephone number and by available software. For cross reference purposes, the editors have assigned each software and hardware vendor a number and each type of software a 3-letter code. The user of the directory can begin at either the chip or hardware level and quickly determine who produces applicable hardware, operating systems, programming software, applications software, books, and periodicals. Or he can turn to the name of a software vendor and learn what type of software the vendor offers and how to reach the vendor. Products are only listed and categorized but not otherwise described. There are no advertisements. For descriptions and purchasing information, a user must call or write the vendor. The directory is updated twice a year (completely reprinted). By itself, it sells for $57.95. With one update, it costs $82.95 and with two, $100.00.

1981 Software Writers Market: 1800 places to sell your software by Kern Publications (190 Duck Hill Road, P.O. Box 1029, Duxbury, Massachusetts 02332), 1981, iii, 180 pages, 8½ × 11 inches, cardstock cover with plastic comb binding.
$45.00

This directory of firms which market and distribute software is designed for the independent software producer looking for a "publisher" or distributor. For each type of distributor, the editors provide information on how the distributor markets software, what kinds are wanted, and how the distributor deals with independent software producers. Where available, royalty rates and contract details are listed. Names, addresses, and telephone numbers of key decision-makers are given for each distributor, except for the final lengthy section in which computer stores are listed by state. For these, only the business name and address is provided.

CONTENTS: Service Bureaus (18 pages); Consulting Companies (16 pages); Hardware Manufacturers (34); Mail order Distributors (24); Book Publishers (14); Computer Magazines (10); Computer Stores (62).

MICRO Resource Update

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

Did you ever wonder just what magazines are rich sources of information on the 6502 microprocessor, 6502-based microcomputers, accessory hardware and software? For several years I have been assembling a bibliography of 6502 references related to hobby and small business systems. The accompanying list of magazines has been compiled from this bibliography. An attempt has been made to give up-to-date addresses and subscription rates for the magazines cited. Subscription rates are for the U.S. Rates to other countries are normally higher.

GENERAL 6502

MICRO
$18.00 per year, 12 issues
P.O. Box 6502
Chelmsford, MA 01824

Compute!
$20.00 per year, 12 issues
P.O. Box 5406
Greensboro, NC 27403

6502 Users' Group Newsletter
21, Argyll Ave.
Luton, Bedfordshire, England

GENERAL COMPUTER

Byte
$19.00 per year, 12 issues
Byte Subscriptions
P.O. Box 590
Martinville, NJ 08836

Computer Shopper
$10 per year, 12 issues
Glenn Patch, Editor
P.O. Box F
Titusville, FL 32780

Computing Today
£ 8.00, 12 issues
Midmags Ltd.
145 Charing Cross Road
London WC2 0EE
England

Creative Computing
$20.00 per year, 12 issues
P.O. Box 789-M
Morristown, NJ 07960

CSRA Computer Club Newsletter
$6.00 per year
P.O. Box 284
Augusta, GA 30903

Dr. Dobb's Journal
$21.00 per year, 12 issues
People's Computer Co.
P.O. Box E
1263 El Camino Real
Menlo Park, CA 94025

GIGO Newsletter
North London Hobby Computer Club
Polytechnic of North London
Holloway, London N78DB
England
Attn: Robin Bradbeer

Interface Age
$18.00 per year, 12 issues
McPheters, Wolfe and Jones
16704 Marquardt Ave.
Cerritos, CA 90701

KB Microcomputing
$25.00 per year, 12 issues
Wayne Green, Inc.
80 Pine Street
Peterborough, NH 03458

Microcomputer Index
$22.00 per year, quarterly
Microcomputer Information Services
2464 El Camino Real, Suite 247
Santa Clara, CA 95051

On Computing
$8.50 per year, quarterly
P.O. Box 307
Martinville, NJ 08836

Personal Computer World
£ 8.00, 12 issues
Sportscene Publishers (PCW) Ltd.
14 Rathbone Place
London W1P 1DE
England

Personal Computing
$14.00 per year, 12 issues
Hayden Publishing Co.
50 Essex Street
Rochelle Park, NJ 07662

Popular Computing
$16.00 per year, 12 issues
P.O. Box 272
Calabasas, CA 91302

Practical Computing
£ 6.00, 12 issues
IPC, Electrical Electronic Press
Dorset House, Stamford St.
London SE1 9LH
England

Purser's Magazine
$12.00 per year, 4 issues
c/o Robert Purser
P.O. Box 466
El Dorado, CA 95623

Recreational Computing
$12.00 per year, 6 issues
People's Computer Co.
P.O. Box E
1263 El Camino Real
Menlo Park, CA 94025

SoftSide
$24.00 per year, 12 issues
P.O. Box 68
Milford, NH 03055

Spreadsheet
$15.00 per year
Visigroup—Visicalc User Group
P.O. Box 1010
Scarsdale, NY 10583

APPLE-RELATED PUBLICATIONS

The Abacus II Newsletter
$18.00 per year, 12 issues
2850 Jennifer Drive
Castro Valley, CA 94546

Apple
$2.00 per issue, quarterly
Apple Computer Co.
10260 Bandley Drive
Cupertino, CA 95014

Apple Assembly Line
$12 per year, 12 issues
c/o Bob Sander-Cederlof
P.O. Box 5537
Richardson, TX 75080

Apple Barrel
$18.00 per year (membership/subs.)
c/o Ed Seeger, Editor
Houston Area Apple Users Group
3609 Glenmeadow Dr.
Rosenberg, TX 77471

Apple Bits
$15.00 per year
$2.00 application fee
NEO Apple Corps
P.O. Box 39364
Cleveland, Ohio 44139

Apple-Can
$20.00 per year, 6 issues
Apple Users Group of Toronto
P.O. Box 696, Station B
Willowdale, Ontario M2K 2P9
Canada

Apple-Com-Post
DM 50.-
Apple User Group Europe
Postfach 4068
D-4320 Hattingen
West Germany
(Printed in German)

Apple Cookbook
$15.00 per year
131 Highland Ave.
Vacaville, CA 95688

Apple-Dayton Newsletter
$18.00 per year
39 Mello Ave.
Dayton, Ohio 45410

The Apple-Dillo
$15.00 per year, 12 issues
c/o Lenard Fein
River City Apple Corps
2015 Ford St.
Austin, TX 78704

Apple For The Teacher
$12.00 per year, 6 issues
5848 Riddio Street
Citrus Hts., CA 95610

AppleGram
$12.00 per year, 12 issues
The Apple Corps of Dallas
P.O. Box 5537
Richardson, TX 75080

The Apple Orchard
$10.00 per year, quarterly
International Apple Core
P.O. Box 2227
Seattle, WA 98111

Apple Peel
$20.00 per year, 12 issues
Chet Lambert, Editor
Apple Corps of Birmingham
1704 Sam Drive
Birmingham, AL 35235

Apple/Sass
$12.00 per year, 12 issues
Honolulu Apple User's Society
P.O. Box 91
Honolulu, HI 96810

Applesauce
$12.00 per year, 6 issues
c/o Earl Rand, Editor
Original Apple Corps
Rolfe Hall 3303, UCLA
Los Angeles, CA 90024

AppleSeed Newsletter
$15.00 per year, 12 issues
P.O. Box 12455
San Antonio, TX 78212

The Apple Shoppe
$12.00 per year, 12 issues
12804 Magnolia
Chino, CA 91710

Applications
AUS $10 per year (plus $10 joining fee)
Apple Users Group
Box 3143, G.P.O.
Sydney 2001, Australia

ByteLines
$12.00 per year, 12 issues
Hi Desert Apple Computer Club
P.O. Box 2702
Lancaster, CA 93534

Call —A.P.P.L.E.
$15.00 per year, 12 issues
$25.00 application fee
304 Main Ave. S., Suite 300
Renton, WA 98055

The Cider Press
$15.00 per year, 12 issues
San Fransisco Apple Core
1515 Sloot Blvd., Suite 2
San Fransisco, Ca 94132

The C.I.D.E.R. Press
$10.00 per year
Apple Computer Information
and Data Exchange of Rochester
369 Brayton Road
Rochester, NY 14616

From The Core
$12.00 per year, 12 issues
Carolina Apple Core
P.O. Box 31424
Raleigh, NC 27622

F.W.A.U.G.
$15.00 per year, about 9 issues
Lee Meador, Editor
Fort Worth Area Apple User Group
1401 Hillcrest Drive
Arlington, TX 76010

The G.R.A.P.E. Vine
$6.00 per year, 12 issues
Group for Religious Apple
Programming Exchange
c/o Stephen Lawson
P.O. Box 283
Port Orchard, WA 98366

The Harvest
$12.00 per year, 10 issues
N. W. Suburban Apple User Group
1015 S. Ridge Rd.
Arlington Heights, IL 60005

L.A.U.G.H.S.
$15.00 per year
c/o Pat Connelly
Louisville Apple User Group
3127 Kayelawn Dr.
Louisville, KY 40220

The Michigan Apple-Gram
$12.00 per year, 10 issues
The Michigan Apple
c/o Marty Burke, Editor
P.O. Box 551
Madison Heights, MI 48071

Mini'App'Les Newsletter
$10.00 per year
Mini'App'Les Apple
Computer User Group
13516 Grand Avenue South
Burnsville, MN 55337

Neat Notes
New England Apple Tree
25 Emerson Street
Medford, MA 02155

Newsletter
$10.00 per year
Apple Bytes of Buffalo
c/o Hank Kolk
171 Tree Haven Road
Buffalo, NY 14215

Nibble
$17.50 per year, 8 issues
S.P.A.R.C.
P.O. Box 325
Lincoln, MA 01773

OKC Apple Times
$10.00 per year, 10-12 issues
c/o Greenbriar Digital Resources
P.O. Box 1857
Edmond, OK 73034

Peelings II
$15.00 per year, 6 issues
The Peelings Co.
945 Brook Circle
Las Cruces, NM 88001

Poke-Apple
$15.00 per year
Apple-Siders
5707 Chesapeake Way
Fairfield, OH 45014

Rubber Apple Newsletter
$12.00 per year, 10 issues
c/o Ken Gabelman
849 Russel Ave.
Akron, OH 44307

The Seed
$18.00 per year, 12 issues
P.O. Box 17467
Denver, CO 80217

Softalk
$10.00 per year, 12 issues
Softalk Publishing, Inc.
10432 Burbank Blvd.
North Hollywood, CA 91601

Southeastern Software Newsletter
$10.00 per year, 10 issues
c/o George McClelland, Editor
6414 Derbyshire Drive
New Orleans, LA 70126

Stems From Apple
$9.00 per year, 11 issues
$2.00 application fee
Apple Portland Program
Library Exchange
c/o Dick Stein
P.O. Box 1608
Beaverton, OR 97075

T.A.R.T.
$15.00 per year, quarterly
The Apple Resource Team
c/o Sid Koerin, Editor
1706 Hanover Ave.
Richmond, VA 23220

Washington Apple Pi
$18.00 per year, 12 issues
P.O. Box 34511
Washington, DC 20034

AIM-RELATED

Interactive
$5.00 for 6 issues
Newsletter Editor
Rockwell International
P.O. Box 3669, RC55
Anaheim, CA 92803

The Target
$6.00 per year, 6 issues
Donald Clem, Editor
RR#2
Spencerville, OH 45887

ATARI-RELATED

A.N.A.L.O.G. Magazine
$10.00 per year, 6 issues
P.O. Box 23
Worcester, MA 01603

Atari Computer Enthusiasts
$8.00 per year
c/o M.R. Dunn
3662 Vine Maple Dr.
Eugene, OR 97405

Purser's Atari Magazine
(available thru dealers only, 2-3 issues per year)
c/o Robert Purser
P.O. Box 466
El Dorado, CA 95623

Iridis
The Code Works
Box 550, 5578 Hollister, Suite B
Goleta, CA 93017

OSI-RELATED

OSIO Newsletter
$15.00 per year
9002 Dunloggin Road
Ellicott City, MD 21043

OSI Users Group
c/o Richard Ellen
12 Bennerley Rd.
London SW11
England

OSI User's Independent Newsletter
$10.00 per year, 6 issues
c/o Charles Curley
6061 Lime Ave. #2
Long Beach, CA 90805

Peek(65)
$12.00 per year, 12 issues
P.O. Box 347
Owings Mills, MD 21117

PET-RELATED

Commodore PET User Group Newsletter
$15.00 per year
Commodore Business Machines, Inc.
3330 Scott Blvd.
Santa Clara, CA 95050

Commodore PET Users Club Newsletter
£ 10.00, 5-8 issues, £ 15.00 overseas
Commodore Information Centre
360 Euston Rd.
London NW1
England

Nieuwegein PET Users Group
Nijpelsplantsoen 252
3431 SR Nieuwegein
The Netherlands
Attn: Hans Tammer or Louis Konings

The Paper
$15.00 per year, 10 issues
Centerbrook Software Designs
Long Island PET Society
98 Emily Drive
Centereach, NY 11720

PET Benelux Exchange
Copytronics
Burg, Van Suchtelenstraat 46
7413 XP Deventer
The Netherlands

Printout
$36.00 (surface mail), 10 issues
$45.00 (airmail)
£ 9.50 (U.K.)
P.O. Box 48
Newbury RG16 OBD
Berkshire, U.K.

The Transactor
$15.00 (Canada) per year, (6-8 issues)
Commodore Systems
3370 Pharmacy Ave.
Agincourt, Ontario M1W 2K4
Canada

SYM-RELATED

Sym-Physis
$10.00 per year, quarterly
$13.50 per year, overseas
Sym-1 Users' Group
P.O. Box 315
Chico, CA 95927

NON-COMPUTER MAGAZINES

EDN (Electronic Design News)
$25.00 per year, 22 issues
Cahners Publishing Co.
270 St. Paul Street
Denver, CO 80206

Popular Electronics
$14.00 per year, 12 issues
One Park Ave.
New York, NY 10016

QST
$18.00 per year, 12 issues
American Radio Relay League
225 Main Street
Newington, CT 06111

Radio-Electronics
$13.00 per year, 12 issues
200 Park Ave., South
New York, NY 10003

73 Magazine
$25.00 per year, 12 issues
P.O. Box 931
Farmingdale, NY 11737

Yacht Racing Programs Wanted

The United States Yacht Racing Union, the National Sports Authority for the racing sailor, has embarked on a program to develop a new *Race Management Manual* for use by race committees everywhere.

One section of the loose-leaf formatted manual (or handbook) will be devoted to various computer and calculator programs and other such aids.

Already we have received a few programs for computers such as one on the rules and several for scoring multi-class regattas, etc.

We earnestly solicit any and all programs readers might have developed relating to sailing, race scoring, handicapping, measurement rules and the like.

A library of such contributions is being maintained at the union's headquarters and contributions should be sent there: USYRU, P.O. Box 209, Newport, Rhode Island 02840.

The listing of the programs in the library will be included in the manual and its frequent up-dates, with appropriate credit to the authors and contributors.

Any questions or comments should be sent to the attention of:

Evans M. Harrell, Chairman
USYRU Race Management Committee
342 Sequoia Drive
Marietta, Georgia 30060

MICRO Software Catalog: XXXVII

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

Name: **The Labors of Hercules**
System: SYM with BAS-1 or KIM 8K BASIC at 2000 H.
Memory: 16K
Language: BASIC
Hardware: Terminal using standard serial I/O ports on SYM or KIM
Description: An adventure game in which you attempt the twelve labors of Hercules. Kill the Lernaean hydra, clean the Augean stables, and bring back the flesh-eating mare of Diomedes. Attempt these tasks and nine others just as Hercules did. You communicate with the computer using one and two word commands.
Copies: Just released
Price: $10.00 on cassette tape, ppd. in U.S. only.
Author: Lee Chapel
Available: Lee Associates
2349 Wiggins Ave.
Springfield, Illinois 62704

Name: **Wall Street**
System: OSI C1P/Superboard/C4P
Memory: 8K RAM
Language: Microsoft BASIC
Hardware: OSI C1P/C4P
Description: Game-type simulation for 1 to 6 players. Each tries to make his fortune in the stock market. Includes gains, losses, stock splits, stock market crash, etc. Great for teaching stock market theory or for just plain *fun*.
Copies: New
Price: $9.95 cassette 300 or 600 Baud
Author: C. Powell III
Available: Software Plus+
1818 Ridge Avenue
Florence, Alabama 35630

Name: **ASTRO-SCOPE™: The Electronic Astrologer™**
System: Apple II or TRS-80
Memory: 32K for screen version, 48K for printout version.
Language: For Apple II, Applesoft in ROM with DOS 3.2. For TRS-80, Disk BASIC 2.3.
Hardware: For Apple II, 1 disk with screen version, 2 disks with printout version. For TRS-80, 2 disks with both versions.
Description: Your complete birth chart read electronically by two well-known astrologers. Not a generalized reading of your sign, but the kind of horoscope a private astrologer would erect, based on your date, time and place of birth and computed to a precision within one-tenth of a degree or better. The planets, signs and houses of one particular birth are analyzed in a text of 1500 words or more, using the modern, psychological approach characteristic of the best in astrology today.
Copies: As needed
Price: Screen version $30.00. Printout version $200.00 (includes license to reproduce textual material commercially).
Authors: Steve Blake and Rob Hand
Available: AGS Software
Box 28
Orleans, Massachusetts 02653

Name: **Pascal Level 1**
System: Apple II
Memory: 48K and ROM Applesoft (compiler); 8K min (run time)
Language: Applesoft and machine language
Hardware: Disk II
Description: This Pascal system consists of a subset of the standard Pascal as defined by Jensen and Wirth. It includes the structured programming features: IF-THEN-ELSE, REPEAT-UNTIL, FOR-TO/DOWNTO-DO, WHILE-DO, CASE-OF-ELSE, FUNCTION and PROCEDURE. It also includes the pseudo array MEM to allow memory PEEKs and POKEs. Now you can learn the language that is slated to become the successor to BASIC. Pascal Level 1 is a complete package that allows you to create, compile and execute programs written in Pascal. The source and object codes are automatically saved on diskette. Sample programs and a user's manual are included.
Price: $35.00 on diskette
Author: Hal Clark
Available: On-Going Ideas
RD #1, Box 810
Starksboro, Vermont 05487

Name: **5 More Great Games!**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II Plus, Disk II
Description: Includes *Turn 'em Loose!*, *Mystery Code*, *Depth Charge!*, *The Mine Fields of Normalcy*, and *Deep Sea Treasure*. These are some of our newest and best games. Each one is great fun, Hi-Res. Best explosion sounds of any software in Applesoft. Machine language sound effects. There's enough action, suspense, and challenge to keep you going for months!
Copies: Many
Price: $29.95 (or $9.95 for any *one* of the above games). Includes game cards, disk, instructions.
Available: Avant-Garde Creations
P.O. Box 30161,
Dept. MCC
Eugene, Oregon 97403

Name: **Capital Assets Management System**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: Disk II, printer of 80-columns or greater
Description: CAMS provides a simple and accurate means for the determination of asset depreciation, investment credit and investment credit recapture amounts. User may select from 8 depreciation methods and print detailed reports in either 80- or 132-column formats. Depreciation is performed on a date-to-date basis rather than just monthly. Investment credit/recapture is performed automatically by CAMS, scanning each file. User determined subtotaling is also supported, as are individual reports. An advanced editor allows trial runs on depreciation methods. Changes to all fields are possible. CAMS records 23 pieces of information on each asset, including GL account numbers and liberal notes. (CP/M version available soon.)
Price: $99.50 (dealer inquiries invited)
Author: Tracy Valleau
Available: Innerface Business Systems
P.O. Box 834
Pacific Grove, California 93950

MICRO Hardware Catalog

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

Name: **PSSBC-A Power Supply**
System: Rockwell AIM 65
Description: Designed to Rockwell's specifications for the AIM 65 single board computer, this unit supplies 5 volts at 2 amps maximum, regulated, and 24 volts at .5 amps average (2.5 amps maximum) unregulated. The 5 volt output is short-circuit-proof and an overvoltage protection (crowbar) circuit protects the circuitry of the attached computer. The supply is enclosed in an attractive all metal case with switch, pilot light and fuse on the front panel. The cable from power supply to computer is supplied.
Warranty: Against defects in materials and workmanship for 90 days.
Price: $64.95 plus shipping. VISA/MC accepted.
Available: CompuTech
Box 20054
Riverside, California
92516

Name: **CD-23-4 OSI to SA4008 Interface Board**
System: Ohio Scientific C3-C (CD-23 systems)
Hardware: Hard Disk Controller to Hard Disk Interface
Description: A hard disk interface board which allows users to interface from one to four Shugart SA4008 Hard Disks to one OSI Computer through the existing controller board.
Price: $845.00 list
Available: TEACO, Inc.
P.O. Box E
2117 Ohio Street
Michigan City, IN 46360

Name: **MEM 4 and MEM 8**
Description: System Peripherals has recently announced their 4K and 8K static memory board for the AIM-65 microcomputer. This is a low power memory board that is plug-compatible with the AIM-65 expansion connector and requires no mother board or other hardware.
Price: $169.00 for MEM 8 (8K)
$109.00 for MEM 4 (4K)
(Introductory prices.)
Available: System Peripherals
P.O. Box 971, Dept. M.
Troy, Michigan, 48099

Name: **P.I.E.-C**
System: PET/CBM, all versions
Description: The P.I.E.-C is a Parallel Interfacing Element between the IEEE-488 port of the PET/CBM computers and any parallel-input ASCII printers. The attractive custom enclosure and direct computer mounting will make your system look professional rather than messy. Because the P.I.E.-C has parallel output with 2 handshaking lines it is compatible with the Epson printers, NEC Spinwriter, IDS 'Paper Tigers', Anadex printers, and of course all Centronics printers. There's no extra power supply because the +5v is obtained directly from the printer. The P.I.E.-C can respond to any of the IEEE-488 primary addresses of the PET/CBM computer systems by simply setting the interfacing switches. The conversion of non-standard PET/CBM codes to true ASCII codes is also switch selectable. The IEEE-488 port of the PET/CBM is extended using the same type card edge. This allows the cable that connects the floppy disks to the computer to be connected to the P.I.E.-C instead.
Price: $119.95 fully assembled with case, code converter and 6' printer cable.
Available: LemData Products
P.O. Box 1080
Columbia, Maryland
21044

Name: **Micromodem II**
System: Apple II
Language: Apple BASIC and Apple Pascal
Hardware: Low speed modem
Description: Complete direct connect data communications system for Apple II and Bell & Howell computers. Features 110 and 300 baud, full or half duplex, with auto dial and auto answer capabilities.
Price: $399.00
Available: Hayes Microcomputer Products, Inc.
5835A Peachtree Corners East
Norcross, Georgia 30092
(404) 449-8791
(Contact address above for nearest retail dealer.)

Name: **VOLTECTOR(R) Series 6**
Hardware: Same
Description: A plug-in style transient, surge, and EMI protector.
Price: $79.50 list
Available: Pilgrim Electric Company
29 Cain Drive
Plainview, New York
11803

Name: **Apple-Crate™**
Hardware: Apple II & II Plus
Description: The "Apple-Crate" is a quality desk-top rack designed to house Apple computer components. It's finished in Hawthorne walnut that is both scratch- and stain-resistant and looks like an expensive piece of furniture.
Price: $59.95
Available: Softsel
4079 Glencoe Ave.
Marina del Rey,
California 90291

Name: **SPS 1-500-24 Standby Power Supply Unit**
Description: Self-contained, reliable power source for use in brownout or blackout. Plug-in unit attaches to regular power source and connected to device requiring protection. Unit generates a regulated quasi sine AC wave from sealed gelled electrolyte battery in less than 25 milliseconds.
Price: $650.00
Available: Welco Industries, Inc.
9027 Shell Road
Cincinnati, Ohio 45236

Name: **16 Channel, 12-bit, Data Logger Interface**
System: AIM 65
Memory: 4K
Language: BASIC
Hardware: AIM 65 plus Columbus Instrument's Data Logger Interface.
Description: Accurately keeps track of laboratory work in medical, industrial, and scientific fields without having to load programs from tape or disk. EPROM resident, auto-booting feature starting AIM as a data logger once the power is on.
Available: Columbus Instruments Int. Corp.
950 N. Hague
Columbus, Ohio 43204

MICRO

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

6502 Bibliography: Part XXXVII

952. Apple/Sass 2, No. 9 (December, 1980)

Misevic, Bruno V., "Dice Roll," pg. 9.
A dice roll program for the Apple.

Misevic, Brúno V., "Low Cost Telephone Dialer and Phone Book," pg. 11.
Hardware and software for the Apple.

Niimi, Dennis S., "Demuffin Corrected," pg. 14-16.
Fixes for the Demuffin program to transfer DOS 3.3 programs to DOS 3.2.1 format.

Ward, Dennis, "Dennis Ward's Display," pg. 19.
A special program for the Apple.

953. The Transactor 2, No. 8 (January/February, 1980)

Anon., "Re-Dimensioning Arrays," pg. 1-2.
Tips on re-defining an array on the PET.

Anon., "Bits and Pieces," pg. 2-4.
Dynamic loading, cursor positioning, monitors, etc. for the PET.

Anon., "POP a Return and Your Stack Will Feel Better," pg. 10-11.
How to jump out of a routine on the PET.

Anon., "Supermon 1.0," pg. 15-22.
A machine language program in RAM which links itself to the built-in PET ROM monitor.

Garbutt, W.T., "RS-232C: An Overview," pg. 23-28.
All about RS-232 and the PET I/O ports.

VanDuinen, T., "Program Plus," pg. 30-36.
Managing programs and routines on the PET.

Anon., "Relocate and Save," pg. 37.
A short utility for the PET.

Brown, B., "Routines for the PET," pg. 38-40.
Several short programs and tips.

954. The Transactor 2, No. 9 (March/April, 1980)

OEI, Robert, "LIST in Lower Case," pg. 1.
Sequence to cause PET to list in lower case/upper case.

Anon., "Printer Tabbing," pg. 3.
Tabbing on the PET printer.

Gardner, L.D., "More on Printer Output," pg. 3-5.
A routine for using Centronics printers with the PET.

Butterfield, Jim, "Input and Output from PET Machine Language," pg. 10.
Utility hints for the PET.

Maclean, Bill, "An Instring Utility for the 16/32K PET," pg. 11.
A utility to change a substring within a main string.

Butterfield, Jim, "PET as an IEEE-488 Logic Analyzer," pg. 12-13.
Routine and technique to show the current status of four of the GPIB control lines plus a log of the last nine characters on the bus.

Butterfield, Jim, "Cross-Reference," pg. 18-22.
A program to do cross-referencing of a BASIC program, on the PET.

Berezowski, David, "Better Auto Repeat," pg. 23.
A repeat key program for the PET.

Hook, David A., "The 'Unwedge' — A Tape APPEND and RENUMBER Program," pg. 24-38.
A useful utility for the PET, in BASIC and machine code.

Barnes, Paul, "Restore Data Line Program," pg. 39.
This routine to restore the data line pointer of the PET at a line other than the first.

Rossland, S. Donald, "Machine Language Case Converter," pg. 42-43.
A PET machine language routine to convert strings to the correct upper/lower case condition for printing on CBM printers with the original ROM.

955. The Transactor 2, No. 10 (May/June, 1980)

Anon., "Remainder," pg. 1.
A special case of the MID$ function on the PET.

Troup, Henry, "Controlling Garbage Collections," pg. 4.
How to force an early garbage collection, at the start of the input, on the PET.

McDonald, John, "More on Screen Print," pg. 8.
Stretch the 40-column PET screen to 80 columns on the printer.

Troup, Henry, "True ASCII Output," pg. 9-11.
Basic and machine language routines for the PET.

Hoogstraat, J., "PET 2040 Disk Buffer I/O Routine," pg. 12-28.
Information on PET I/O procedures.

Dean, Sheldon H., "PET to Heathkit H14 Printer Serial Interface," pg. 29-32.
Hardware article on the PET interface.

Troup, Henry, "Filestatus," pg. 33.
A short routine to check the PET file status.

Butterfield, Jim, "BASIC 4.0 Memory Map," pg. 34-41.
Hex and decimal locations of PET functions in BASIC 4.0.

956. The Transactor 2, No. 11 (July/August, 1980)

Butterfield, Jim, "Text Editor," pg. 8-10.
A simple line oriented editor for the PET.

White, Don, "High Resolution Graphics for the PET," pg. 12-21.
Adapting the "Visible Memory" to the PET.

Hook, D., "Card Printing Utility," pg. 22-27.
Utility for printing playing cards on the PET.

Butterfield, Jim, "Simple 8010 Modem Program," pg. 28-29.
A program to output the PET to an ASCII system.

957. OSIO Newsletter 2, No. 2 (February, 1980)

Wallis, T.L., "Memory Map of OS65U and Location of Various Parameters," pg. 3-8.
Hex and decimal locations of the Ohio Scientific OS65U.

958. OSIO Newsletter 2, No. 3 (March, 1980)

Anon., "Some C1P Routines," pg. 1-2.
Several subroutines for the C1P micro.

Schwartz, Danny, "C1P Sketchpad," pg. 3.
Drawing program for the C1P lets you see the video screen as a sketch pad.

959. OSIO Newsletter 2, No. 4 (April, 1980)

Boardman, J.B., "Serial Monitor ROM," pg. 4-6.
The OSI 65A monitor at FE00 dissembled.

960. OSIO Newsletter 2, No. 6 (June, 1980)

Popenoe, Chuck, "Message Center," pg. 4.
A message program for using a C1P as a bulletin board.

Anon., "Fitting a Format," pg. 5-7.
Tips on writing formatting routines for OSI micros.

961. OSIO Newsletter 2, No. 7 (July, 1980)

Brounstein, Sid, "Challenger on the Phone," pg. 2-5.
Telecommunications interface program for the OSI 550 board.

Mason, Jim, "Real Time Clock," pg. 6.
A clock program using the OSI 600 board.

Morgenstein, David, "PEEK(15908)," pg. 7.
Tips for using OS-U and a line printer.

962. OSIO Newsletter 2, No. 8 (August, 1980)

Kirshner, Joe, "OS-65D Notes," pg. 1-2.
A number of interesting uses for PEEK(64513).

Morganstein, D., "Indirect ASCII Files," pg. 3-4.
A tutorial for files on the OSI systems.

Randal, John, "Program PRONLY," pg. 5-7.
How to store programs efficiently on disks.

963. OSIO Newsletter 2, No. 9 (September, 1980)

Callaghan, Bill and Kupperian, Jim, "Modems," pg. 5-7.
Hardware and software program for modem operation using the OSI, C1P or C2P.

964. The Apple Orchard (Fall, 1980)

Bishop, Bob, "Apple II Hi-Res Graphics: Resolving the Resolution Myth," pg. 7-10.
Explaining some limitations of Apple high resolution graphics.

Rowe, Pete, "The Mysterious Orange Vertical Line," pg. 11.
The orange line at the left side of the Hi-Res Apple screen is explained.

Spurlock, Loy, "Understanding Hi-Res Graphics," pg. 12-21.
How to include text in your Hi-Res graphics program on the Apple.

Crossley, Johm, "ASCII, EBCDIC, and the Apple," pg. 31.
Selectively convert Apple's output to EBCDIC or convert incoming EBCDIC to ASCII with this routine.

Anon., "Yes! There Is A Fix for APPEND in DOS 3.2 (and 3.2.1)!", pg. 31.
Fix up the APPEND routine on the Apple DOS with an End of File marker.

Anon., "RFI: The F.C.C. and Your Apple," pg. 32-35.
Tips on improving the suppression of spurious radiation from the Apple.

Kellner, Jo, "Pascal Operand Formats; or, The Secret Life of a Variable," pg. 38-40.
All about Pascal variables on the Apple.

Anon., "Auto-Run Apple Without DOS," pg. 42-44.
How to start up your non-disk Apple program without a disk.

Crossley, John, "Initializing Apple Peripherals with POKES," pg. 43.
Lists of the Apple POKES needed to initialize the memory locations used by various interfaces.

Anon., "AppleWriter Modification for Lower Case Display," pg. 43.
A software modification allowing AppleWriter to be used with the Paymar Lower Case Adapter.

Budge, Joseph H., "Inside Initialization," pg. 49-52.
A tutorial for the Apple user, relating to disk operation.

Kamins, Scot, "Locksmythe and the Dedicated Programmer, or — Writing User-Proof Interactive Code," pg. 54-58.

Anon., "Linking Machine Language Routines to Applesoft Programs," pg. 61.
A technique showing how to hide a binary program that will follow the Applesoft listing but which will not appear in the listing.

Silverman, Ken, "Don't Overload Your Apple II," pg. 67-69.
Tabulation of the voltage and current requirements of various Apple interface cards and accessories.

965. The Paper 3, Issue 6/7 (Fall, 1980)

Breed, Alex, "PET's Not-So-Random RND," pg. 9.
A discussion of problems involved in generating random numbers.

Eisner, Gerry, "Meddling with Middle String," pg. 14-15.
A tutorial on the MID$ function on the PET.

Bromley, J.R., "A Better Screen Copy," pg. 16-17.
A screen dump to printer program for the PET.

Sherwood, D., "Cassette Survival Hints," pg. 18-19.
Tips on using the PET cassette system including tape quality, tape storage, cassette heads and alignment, etc.

Busdiecker, Roy, "Auto Repeat Keys, Version Two," pg. 22-23.
Another program to provide repeat keys on the PET.

966. Apple Bits 2, No. 8 (October, 1980)

Chilton, Peter, "Pascal/Fortran," pg. 4.
A brief comparison of Pascal, Applesoft and Z-80 machines.

Kovalik, Dan, "Taking the Mystery and Magic Out of Machine Language," pg. 5-6.
A tutorial on Apple Hi-Res machine language.

Koehler, John, "BASIC Basics," pg. 8.
Program based on the implementation of Fisher's algorithm for the internal rate of return of an investment.

967. Softside 3, No. 1 (October, 1980)

Laurence, Matthew, "An Apple One Liner," pg. 4.
A mini program for the Apple.

968. Apple Assembly Line 1, Issue 1 (October, 1980)

Sander-Cederlof, Bob, "Programming Notes," pg. 2-3.
How to add and subtract one (incrementing and decrementing) on the Apple.

Sander-Cederlof, Bob, "General Message Printing Subroutine," pg. 4-8.
Formatting messages in assembly language on the Apple.

Matzinger, Bob, "Using the Paymar Lower Case Adapter with S-C Assembler II Version 4.0," pg. 4.
A machine language routine for the Apple.

Sander-Cederlof, Bob, "Hardware in All 6502 Chips!", pg. 10-11.
A bug in the 6502 and a suggested fix.

969. The Apple Peel 2, No. 4 (April, 1980)

Brown, Tom, "Disk File Hints," pg. 2.
How your disk file program can determine whether file name already exists in the Apple disk.

970. The Apple Peel 2, No. 5 (May, 1980)

Selig, David, "Onerr Routines," pg. 5.
A hint on using error routines on the Apple.

Chipchase, Frank, "Better Use of Apple II Renumber and Merge Program," pg. 6.
A short tutorial on these Apple routines.

971. The Apple Peel 2, No. 7/8 (August, 1980)

Hertzfield, Andy and Larsen, LeRoy, "Free Space Formalized," pg. 3.
A basic program for utilizing the free space machine language routine for the Apple.

Knaster, Scott, "Using the 'Old Monitor ROM' with the Language System," pg. 4.
A procedure for enhancing the use of your language system.

Anon., "APPEND Fix in DOS 3.2 and 3.2.1," pg. 6.
How to patch up the missing end-of-file marker in the Apple disk operating system.

Brown, Tom, "Fireworks," pg. 7.
A Hi-Res graphics program for the Apple.

Anon., "Soundex," pg. 9.
A routine to use in search routines that will give approximate matches in place of requiring exact matches.

Hyde, Bill, "Lo-Res Printout," pg. 10.
A subroutine to print out the Apple Lo-Res screen in a primative fashion.

Hartley, T., "Catalog Program," pg. 13.
Routine to set up a catalog to hold 181 file names on the Apple disk.

Buchler, Dan, "Justification Routine," pg. 13-14.
Routine to right justify all output to the Apple screen.

972-A. OSIO Newsletter 2, No. 10 (October, 1980)

Kirshner, Joe, "OS-65D Notes," pg. 1-2.
Change BEXEC on the OSI system to include a program list.

Anon., "65412," pg. 2.
How to use location 65412 in conjunction with the WAIT command on the OSI systems.

Versace, Mike, "True C8P OS-U Backspace," pg. 3-4.
A backspace working on both the Challenger III and the C8P.

Solntseff, Nicholas, "Right Justification Revisited," pg. 6-7.
A right justification routine for OSI systems.

972-B. OSIO Newsletter 2, No. 12 (December, 1980)

Kirshner, Joe, "OS-65D Notes," pg. 3-5.
A tutorial on sequential files for OSI systems.

973. Rubber Apple Users Group 3, No. 7 (October/November/December, 1980)

Gabelman, Ken, "Disk Structures IV," pg. 1-9.
More including a listing for "the Invisible Mailing List."

974. Atari Computer Enthusiasts 1, Issue 2 (Fall, 1980)

Goff, S., "Electronic Christmas Card," pg. 1.
A graphics program for the Atari.

Dunn, M.R., "Edit Subroutine," pg. 2-3.
Copy and disk file edit for the Atari.

Dunn, M.R., "Typewrite," pg. 3.
Atari program to turn the computer into a typewriter.

Dunn, M.R., "Disk Utilities," pg. 3.
Two disk directory routines for the Atari, Disk Menu and Disk Directory Subroute.

Dunn, M.R., "Speak Atari," pg. 6.
Several uses of the Atari GOSUB function.

Goff, Stacy, "Listen Atari," pg. 7.
Connect your Atari to your stereo for fantastic sound.

975. Appleseed Newsletter 2, No. 4 (October-December, 1980)

Wright, Don, "BASIC-RWTS Interface Bug Shot," pg. 7.
Fixing the minor bugs in DOS 3.3 regarding the IBSTAT parameter and the RWTS Link program, on the Apple.

Connelly, Pat, "Lo-Res to Hi-Res," pg. 11-13.
A graphics program for the Apple.

Anon., "Hex to Decimal Conversion," pg. 14.
A simple technique to use the Apple monitor for numbers conversions.

Pump, Mark, "Apple II DOS Internals," pg. 14-19.
Memory map and tutorial on the Apple DOS 3.2.

976. Apple Bits 2, No. 9 (November, 1980)

Kvalik, Dan, "Taking the Mystery and Magic Out of Machine Language," pg. 5-6.
More about machine language and Hi-Res graphics with several listings for the Apple.

Koehler, John, "BASIC Basics," pg. 7.
Tips on the operation of floating-point BASIC.

977. Softside 3, No. 2 (November, 1980)

Pelczarski, Mark, "Apple Programming Notes," pg. 8.
Several hints for Apple users.

Ward, Dennis, "One-Liner," pg. 19.
A one line graphics routine for the Apple.

Pelczarski, Mark, "Softside's Data Base: Part Three," pg. 30-31.
Sorting routines, Apple and Atari.

Bohlke, David, "Engineer," pg. 50-51.
A game for the Atari.

Hausman, Rob, "Keyboard Organ," pg. 62-63.
A machine language routine for the Apple.

Hays, Tim, "Trench," pg. 66-67.
A game for the Atari.

Truckenbrod, Joan, "Computer Aided Drawing an Design: Rotation Techniques," pg. 74-75.
A tutorial and program listing for Hi-Res Appl graphics.

ADVERTISERS' INDEX

JUNE 1981

Webster, Ron, "Boing!", pg. 82-83.
A graphics program for the Atari.

978. The Apple Barrel 2, No. 8 (November, 1980)

Kramer, Mike, "Name Swap Subroutine," pg. 5.
Swap first and last names in your File Cabinet, for the Apple.

Kramer, Mike, "Printer Activate/Deactivate Subroutines," pg. 6.
Tabbing past column 40 using certain interface boards and printers.

Anon., "Apple/Pascal Library," pg. 10-11.
Catalog of Apple/Pascal programs on several disk volumes.

Meador, Lee, "DOS 3.2 Disassembly," pg. 14-20.
Part five of this series is on the Apple Disk II controller.

979. Apple Assembly Line 1, Issue 2 (November, 1980)

Sander-Cederlof, Bob, "Variable Cross Reference for Applesoft Programs," pg. 2-8.
A useful tool when you are writing large Applesoft programs.

Sander-Cederlof, Bob, "Assembly Source on Text Files," pg. 9-14.
A program to allow you to save a source program on a text file so that it can be used in another editor or assembler.

Sander-Cederlof, Bob, "A Simulated Numeric Key-Board," pg. 15-16.
A program to turn part of your Apple keyboard into a simulated numeric key-board.

980. The Apple Peel 2, No. 11 (November, 1980)

Anon., "Addresses in the Apple II DOS," pg. 3.
Some key addresses in Apple DOS 3.1 and 3.2.

Brown, Tom, "POKE Salad," pg. 7-8.
Using Applesoft ROM routines. The Ampersand and its use in calling up a number formatting code.

981. Peelings II 1, No. 4 (November/December, 1980)

Staff, "Software Reviews," 40 pgs.
A review of a variety of Apple programs including a section which reviews and evaluates seven word processors for the Apple.

982. Applesauce 2, No. 2 (November/December, 1980)

Fisher, Bill, "Schematic for Apple Telephone Dialer," pg. 2.
A schematic of hardware to use the I/O output port of the Apple as a telephone dialer.

Ender, Philip B., "Recovering a Deleted Pascal File," pg. 4-5.
A technique for recovering Pascal files like the Disk Zap or Lazarus utilities in BASIC.

Ender, Philip B., "Double Density Disk Storage in Pascal," pg. 6.
Double the data on a diskette by packing your Pascal variables or records.

Rivas, Lou P., "Applewriter Patches to Support: 1) 96-Character ASCII Input 2) Dan Paymar Lower Case -Mod," pg. 7-8.
A mod for the Applewriter word processor.

Amromin, Joel L., "Slowlist Patches," pg. 11.
A modification to permit a slowlist routine to work on Applesoft listings.

Anon., "Easy Hex to Decimal Conversion," pg. 11.
A simple technique for number conversion using the Apple monitor resources.

Campbell, Bill, "Catalog Utility," pg. 12-13.
A quick disk-based Apple utility to provide convenient housekeeping on your disks.

Stearns, Brian, "REM Remover," pg. 14-15.
Make your Integer BASIC listings easier to read, for the Apple.

Ras, Henry, "Disk Menu Program," pg. 17-20.
Condense your Apple Disk Catalog by eliminating the sector count.

983. Call — Apple 3, No. 9 (November/December, 1980)

Golding, Val J., "Window on the World," pg. 7-9.
A tutorial on the use of Text Screen Windows.

Connelly, Pat, "Animation with Data Arrays," pg. 11-17.
Create, compress and decompress Apple data array shapes.

Lingwood, David A., "Overlaying in Applesoft," pg. 19-20.
Too much code, too little space. Or, how to get a 20K program to run in 8K on the Apple.

Rettke, Dick, "FNC, A Non-Flashing Cursor for Your Apple II," pg. 23-27.
If the flashing cursor bothers you, cool it!

Golding, Val J., "The Last Word in Hex Converters," pg. 32-33.
Several number converters for the Apple.

Billard, Stephen L., "DOS 3.3, the Language Card and the Apple II," pg. 37-38.
Discussion of techniques to improve the utility of the multi-lingual Apple.

Caloyannides, Michael A., "Fortran for the Apple Computer: Two Alternative Approaches," pg. 41-43.
A discussion of implementing Fortran on the Apple.

Huelsdonk, Bob, "Making BASIC Behave," pg. 43-47.
A tutorial for the Apple, using a home inventory program example.

Reynolds, Lee, "Applesoft Variable LIST Statement," pg. 58-59.
A list utility for the Apple.

Lee, Scott, "Muffin Catalog Supplement," pg. 62.
A utility to increase the convenience of using the MUFFIN program to convert disk files from DOS 3.2 to 3.3.

Herzberg, Norman P., "OUTLINE: A Program to Print REMS," pg. 63-64.
A short utility to list only the comments contained in a program.

Sander-Cederlof, Bob, "A Problem with the VAL(A$) Function," pg. 65.
A fix for a bug in the VAL function in Applesoft.

984. The C.I.D.E.R Press (Rochester) 2, No. 6 (November/December, 1980)

Berube, Jim, "Disk Utilities for DOS 3.2 and 3.3," pg. 7-9.
A series of routines for servicing the needs of Apple disks.

Hall, John, "Locating Errors in Pascal Programs," pg. 15.
A short tutorial for Pascal users.

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.

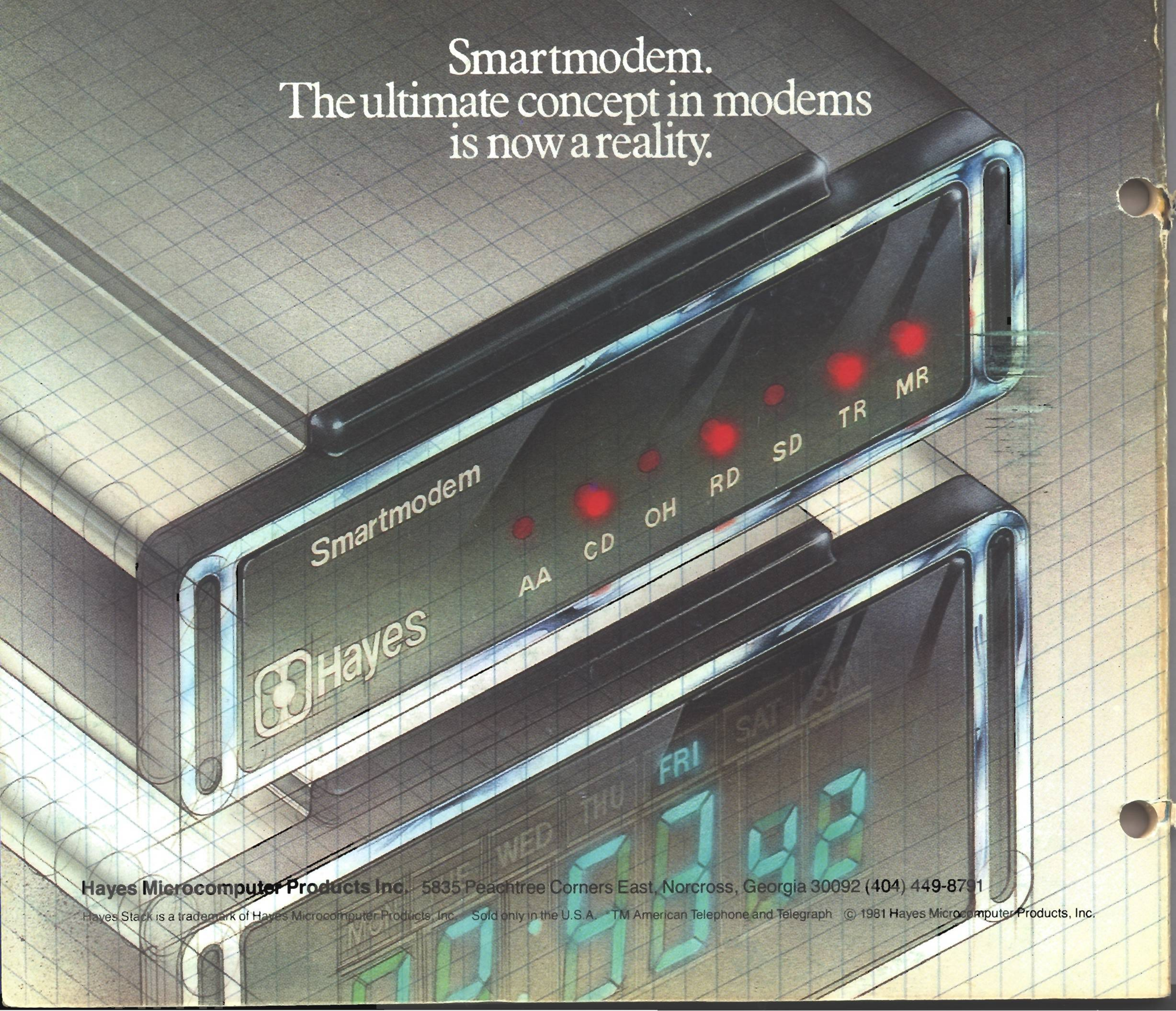
Smartmodem, Smart Buy. Professional quality features. Versatile performance. A full two-year limited warranty. A suggested retail price of only $279.

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