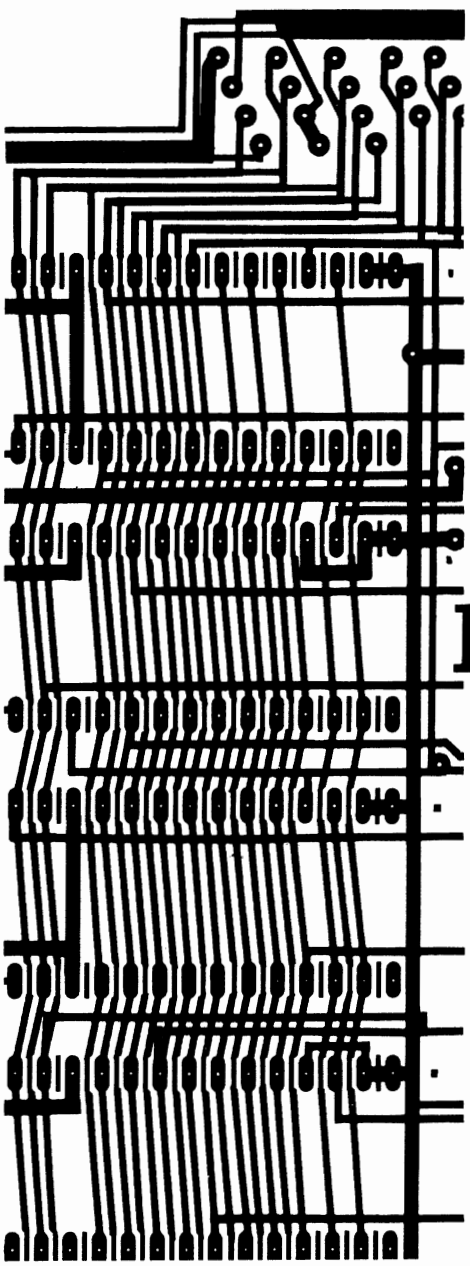
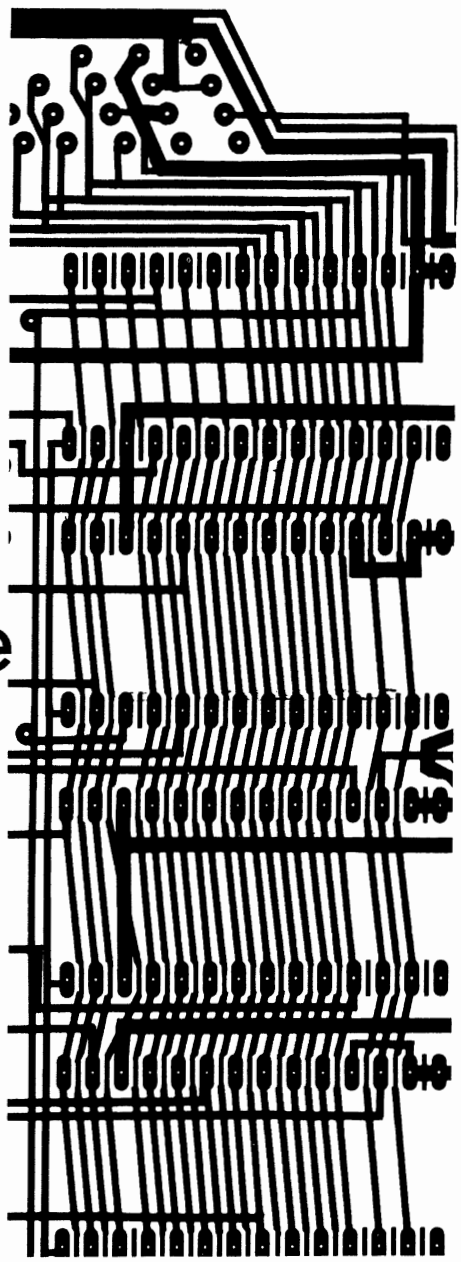
APRIL/MAY, 1980. ISSUE 1. $2.00

compute II.

The Single-Board COMPUTE™



The 6502/1802 Resource



The Single-Board 6502

Eric Rehnke

You asked for it-you got it-your own magazine. How 'bout that?

Let's give Robert Lock a resounding 'Well Done' for giving KIM, AIM, SYM and OSI users our own magazine to expand into. Let's also understand that since we now have a greater vehicle for expressing ourselves *in,* we have an even greater responsibility for expressing ourselves. And since **compute II** pays for your time as well as putting your name up in lights, you have no excuse but to turn that writer on that lives and breathes inside every computer freaque. **compute II NEEDS YOU!!!**

EPROM SIMULATOR

Ever since I first saw an ad for the Pragmatic Design's 'Debug Memory' (DBM-1) board I became fascinated by the design concept of this rather unique RAM board.

Basically, the DBM-1 is a 2Kx8 RAM board for the S-100 bus which can simulate a 2K ROM or EPROM (or 2-1K ROMS) to a target system.

In other words, the same 2K block of RAM that appears to the development system as normal random-access memory, appears to the target system as a 2K ROM/EPROM through a cable plugged into that ROM socket.

It can achieve this nature by viture of its dual-port RAM design. This means that the same memory can be accessed from two separate systems each with its own address and data bus.

The benefit of this type of arrangement becomes obvious when you consider that everything that is written to the 2K block of RAM in the development system also appears in the ROM/EPROM address space of the target system.

Now, instead of developing a program inside your large development system, then burning an EPROM, and installing it in the target system to see if it works, you simply write the program into the RAM on the "Debug Memory" board.

It will then appear in the target system and can be tested immediately. If there's a problem, no EPROM to erase and reprogram, simply write the corrected software to the DBM-1. The updated software automatically appears in the target system.

Saves lots of time!!!

The biggest benefit of the EPROM simulator will be gained when developing software for small, dedicated controllers which have no built-in monitor software of their own. But, using the simulator will also save lots of time when writing programs for semi-smart single-board computers like AIM or SYM, for example.

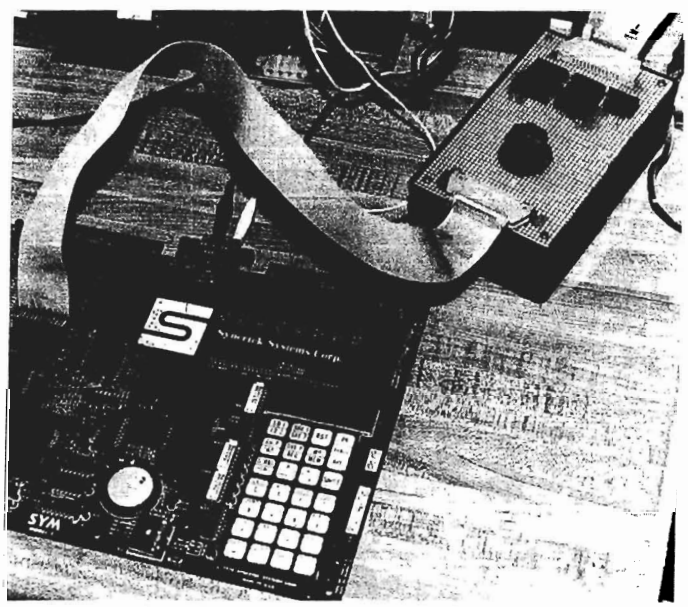
Another, not so obvious, use of the simulator is for developing a character generator for a video board. The video board can even be plugged into the development system itself.

(Most memory-mapped video boards also use this dual-port RAM concept in their design).

Of course, for the simulator to function most efficiently, the assembler in the development system must be able to assemble an object program to a location in memory other than specified by the program counter equate in the source code of that program. This is called 'assembling with offset'.

Say the EPROM socket in the target system resides at $F800-$FFFF in that system while the EPROM Simulator is addressed at $C000 in the development system. To be able to have a program assembled to run at $F800 while actually residing at $C000, it needs to be assembled with an offset. Get it?

My assembler (from HDE) has this capability sc I decided to design my own EPROM Simulator. (I couldn't use the DBM-1 since it was designed for the S-100 bus).



EPROM SIMULATOR shown plugged into a SYM for easy software development.

Also, my version is rather more simplified than the DBM-1 since there were some features I didn't feel I needed - like trap address comparators, daisy chaining, etc.

I built it on an HDE prototyping card and designed it to simulate the 2708, the TI 2716 or the Intel 2716. This should satisfy most of the requirements. Later, when 4K EPROMS get cheaper, I'll build a version to simulate them.

So far, I've used the simulator to speed up software development for my SYM system, look forward to using it with my AIM system, and will use it to develop a PET character generator for an upcoming video board design.

As I have found, the EPROM Simulator is a development tool which can really save time and improve the efficiency of the system designer.

MORE FROM HDE

I have a difficult time keeping up with this company at times. They're surely not resting on their past accomplishments.

The first thing they've done is add conditional assembly directives to their already formidable 2-pass, disk-based assembler.

Besides the conditional assembly directives. IFZ (IFZERO), .IFN (IFNOTZERO), and .EIF (END IF), there are other special directives like .SOR (SORT SYMBOL TABLE), and .COV (INSTRUCTION COUNT) which enable the HDE Assembler to offer capability approaching that of the 6502 Cross Assembler available on time-sharing systems. HDE's Assembler actually comes out on top in the area of conditional assembly (that capability is not available in the Cross Assembler). The only noticeable features of the Cross Assembler which aren't available in the HDE counterpart are the symbol table cross referencing facility and the macro pre-processor. As soon as HDE adds these capabilities to their system (and they're in the works, according to Hudson Digital Electronics), they will have even more features than the Cross Assembler.

Besides that, HDE added the CHAINING function to their already much enhanced version of MICROSOFT Basic. This lets you run Basic programs which are actually too large to fit into your memory. According to HDE, a disk file system is next on the list for their not-so-basic BASIC.

I've been playing with the new conditional assembler for several days, their enhanced Basic for about a month, and continue to be impressed with the software (and hardware) this company produces.

As you may know, I have a KIM system with a couple of full-size HDE floppies and can wholeheartedly recommend this company's stuff to anyone who wants a darn good 6502 development system. (HDE, POB 120, Allamuchy, NJ 07820 PHONE 201-362-6574)

TIPS FROM READERS

From Christopher J. Flynn (2601 Claxton Dr., Herndon, VA 22070)...

KIMSI fans, if they don't already know, should get a copy of Forethought's application note A15. It describes the use of the popular EXPANDORAM dynamic memory board in the KIMSI. A two chip refresh controller is added to the KIMSI breadboard area. The refresh controller causes the 6502 to wait when refresh is needed thus slowing things down a little. I have had the EXPANDORAM running for two or three months now with no problems. It's a very good way to get a lot of RAM for a moderate price without requiring a huge power supply.

From Sam Sturgis, (86 Fisher St., Medway, MA 02053)...

Micro-Ade Assembler users can fix a bug in version 1.0 by changing location $2AF9 from $49 to $48. This will allow use of the symbolic argument modifiers /, +, and - with the define byte (=) psuedo instruction. Without this change, the modifiers are ignored by the assembler, making it extremely difficult (if not impossible) to construct symbolic tables.

From an anonymous reader. . .

AIM users who would like to have darker print from the printer can parallel R21 with an additional 4.7K resistor. I'm not sure how this affects the life of the printhead, but I'm sure Rockwell wouldn't approve of this mod.

HARDWARE REVIEW

SPEAK & SPELL INTERFACE (available from East Coast Micro Products, 1307 Beltram CT, Odenton, MD. 21113).

The thought of having your computer actually speak to you has probably crossed about every hobbyists mind at one time or another. The fact that very few of us ever hear a word uttered from our computers is probably due more to the cost of such speech output devices and not to their availability as most of them are in the $400-$600 range.

When Texas Instruments introduced the SPEAK & SPELL for about $50, I had a feeling that a truly "cheap" speech output device was just around the corner.

I forget just how long that "corner" turned out to be, but it must have been about a year before Dave Kemp of East Coast Micro Products introduced his SP-1 interface for for the SPEAK & SPELL.

Dave initially sent me his information package which acquianted me with the SP-1 interface and discussed some of the basics of the SPEAK & SPELL's method of speech synthesis.

The most important thing that I learned was that

"...SP-1 does not turn SPEAK & SPELL into a black box speech synthesizer which can be used to add voice output to a users' chess, bridge, or Star Trek program. It is a tool which will allow the serious experimenter to investigate speech synthesis at a cost far below other commercially available synthesizer boards."

Later, I got the chance to review the SP-1 interface package.

The documentation for the SP-1 turned out to be quite a bit more than I expected. Only 2 of the 28 pages were devoted to assembling the interface board which indicated that this package was intended for the advanced experimenter. The software drivers were presented in well commented source listings as well as detailed explanations of how the SP-1 operates and a special section on SPEAK & SPELL theory of application. Software flow charts and a bibliography were also included.

I rate the documentation as excellent. Plenty of information here.

I hooked the interface to my SYM, loaded the hex dump software and shortly thereafter, SYM was giving me a vocal hex dump of its memory contents.

Shades of DEMON SEED. SYM was actually talking to me!!!

Other software was included to enable one to pull word data out of SPEAK & SPELL for analysis, but I never got around to using it. It would have been almost anti-climactic after hearing SYM actually talk.

The vocal hex dump software is the only piece of "black-box" software you get with the SP-1 that you can plug in and use. It's a great demo of the possibilities. The other software included is meant to aid the experimenter in working with the SPEAK & SPELL and developing things further.

By the way, no modifications were necessary to the SPEAK & SPELL as the SP-1 plugged into an internal edge connector.

The SP-1 interface package sells for $49.00 and is available directly from East Coast Micro Products.

If you want to tinker around with speech synthesis at a low price, check out the SP-1.

SOFTWARE REVIEW

Assembler/text Editor (sold by M.S.S. Inc., POB 2034, Marshall, TX 75670).

What can you get for $25.00 these days? Well, if you like to tinker with software, you could get the beginnings of an interesting little assembler/editor. Or, you could get an education in what makes an assembler tick.

Actually, you do get more than just the beginnings of an assembler. It's almost complete.

What's all this nonsense you ask? OK, I'll try to describe it.

For $25.00 you get a standard, no frills line editor and 2-pass assembler that resides in slightly over 2K of memory. The assembler mnemonics are close to MOS Technology's standard. How close? The actual mnemonics correspond perfectly. It's the operand formats that differ. I'm sure you could get used to it, though.

The best thing about the package is that you get a source listing for it included for no extra money. It's not very well commented, but it's better than nothing. A serious student of the 6502 dialect should have little problem figuring out what's going on.

The reason I said that the package is ALMOST complete is because there's no built-in means of saving or loading assembler source code to/from cassette (or other mass storage device). If you want to do that, you'll have to add it yourself. They do, however present an example of a routine to add a new command to the system so, if you know what you're doing, you could probably add the necessary cassette support routines.

The fact that the source listing is included opens a rather exciting possibility. It's possible to make the assembler into a cross-assembler for some other CPU-like the 1802, for example. Then you could get that 1802 Simulator program by Dann McCreary and have a complete 1802 development system on your 6502!!!

Does that turn you on?

This assembler and its associated documentation is NOT for beginners! But, could be a good value if you know what you're getting into.

BLUE SKY CORNER

If you also have an amateur radio operator's license, you have the opportunity to connect your computer to some radio gear and communicate with fellow amateurs through a rather sophisticated satellite-born radio repeater system.

Since the FCC has approved the use of ASCII for some satellite communications, the use of computers is a natural. And once you have a computer hooked up, all kinds of things are possible.

How 'bout a fully automatic, cross country communications network?

Is there anyone out there working along these lines? Or, do you know of any groups or publications dedicated to such an end? I'd sure like to hear from you.

Remember the Adventure Game for KIM? (Issue 3, COMPUTE)

It's $24.95 and is available from ARESCO, Box 1142, Columbia, MD 21044 ©

Here, in full, is Gene's Column 1. A portion of this column was printed in Issue 3 of COMPUTE. We thought we'd start at the beginning for Issue #1 of compute II. RCL

Nuts and Volts

Gene Zunchak
Niagara Micro Design, Inc.
1700 Niagara Street
Buffalo, N.Y. 14207

With this article, I hope to begin a series of dissertations on the art and joy of doing it yourself. But first let me introduce myself. I'm Gene Zumchak. I graduated with an M.E.E. from Cornell in 1968. I paddled across Cayuga Lake and got my first job with a likable maverick named Robert Moog of Moog Synthesizer fame. I left my first, and one of the only legitimate jobs I've had in 1970. I moved to Buffalo and since that time have worked for a series of little flake outfits trying to find a winner, but at the same time, avoid having to work for a living. So far I've managed neither, but sure have had a lot of fun. But alas I think I'm getting closer. I now call myself Niagara Micro Design, Inc., and although the pay isn't that hot, the boss let's me play golf whenever the whim takes me (between thaws in July).

Anyway, I started drooling over micros in the early seventies when they first appeared in the electronic design magazines. Then in '75 an unknown semi company called MOS Technology made a big two month splash in the magazines (never to be heard from again) prior to a Western show. I clipped a coupon, sent in $25 and became the proud owner of a 40-legged centipede called a 6502. Now to give you some idea of just how fantastic that was, the 8080 chip set (8080, 8224, 8228) was selling for over $200. A few days after the chip arrived, I got a flyer advertising a single-board computer for just a few bucks more than the naked 8080 chip set. I ran to the bank, withdrew my life savings and sent in coupon number two becoming an even prouder owner of KIM-1 serial no. 00005. Soon afterwards, three friends and myself contracted to design and build five smart printer systems for Honeywell, without a TTY or assembler, let alone a development system. With little more than that KIM and a Simpson VOM we delivered five microprocessor controlled printers. Today, the same job would be a lot easier, but my "development system" is still just an expanded KIM. Anyone, in fact, can put together a development system for only a couple hundred dollars over the price of a KIM, SYM, AIM or other system. Putting together that development system is one of the topics I hope to get to in the future.

Presently I am working on a book entitled "Microcomputer Design and Maintenance" with the guidance of Jon Titus of the "Blacksburg Group" who hopefully will get it published. The topics presented in this series of articles in Compute will touch on some of the material in that book, as well as different material.

A dedicated micro system can be put together for under a $100. Why tie up your Apple or Pet to turn the furnace on, run your electric train, or program your wife's loom? Why not use your system as a tool to crank out dedicated controllers? I hope to show you how. Enough B.S. Let's start talking about hardware.

Read/Write Timing

The most important consideration in hardware design is read/write timing. It is not a complicated topic, but many "designers" avoid confronting it by surrounding a CPU with family chips (usually expensive) using circuits right out of the manuals. There's nothing wrong with the fancy family chips if you really need them. Oftentimes the most attractive chip may belong to another family. If you understand read/write timing, however, you may indeed be able to use a foreign chip.

Write Timing

The terms "reading" and "writing" always reflect the direction of data flow from the perspective of the processor. Thus in a "write" operation, data is presented by the processor to some external device, memory or output, and locked into that device. A bit of memory or output is a flip-flop. In memory, the output of the flip-flop can be read back into the processor. In "output", the output of the flip-flop is connected to the world. (In some programmable devices, an output can often be read back into the processor.)

The usual type of flip-flop used with a processor is the "D type" flip-flop. A D flip-flop has a "D" or data input, and a clock (strobe) input which is an edge-sensitive function. That is, data is presented to the D input and is transferred to the output when the active edge occurs (usually positive going). An edge triggered flip-flop's output can change only on a clock edge. An example is the TTL 7474 dual D flip-flop. A variation of this is the transparent latch. It too has a D or data input, but a Gate input, instead of a strobe. When the gate is true, the output follows the data and is transparent to the data. Data is locked into the flip-flop on a false going gate edge. The 7475 is a quad transparent latch. In both types of flip-flops, data is locked in with a clock or gate edge.

In any latching operation, the following sequence occurs: Data is presented to an input, a locking edge occurs, and finally the data is removed. In general, the data to be written exists before and after the

locking edge. We are now ready to define the important parameters of a write operation. The "set-up" time is the minimum time the data must be present before the locking clock edge occurs. The "hold" time is how long the data must remain after the locking edge has gone away. The set-up and hold times for a 7474 flip-flop are only 20 and 5 ns. respectively. Since these times are so short, TTL latches can always be assured of working with MOS processors.

6502 Write Timing

A 6502 clock cycle is read cycle when the R/W line is high, and a write cycle when the R/W line is low. A cycle is divided into two (more or less) symmetrical halves. In the first half, the ϕ1 clock is high. The R/W line and the address lines change ϕ1. In ϕ2, data transfers occur. According to the spec sheet, the delay from the fall of ϕ2 to the beginning of the rise of ϕ1 can be zero (no max spec given). For zero delay, the ϕ1 clock is approximately ϕ2. However for external use, ϕ2 and ϕ2 are generally used. ϕ1 should not be used as a substitute for ϕ2. Figure 1. shows write timing for the 6502.

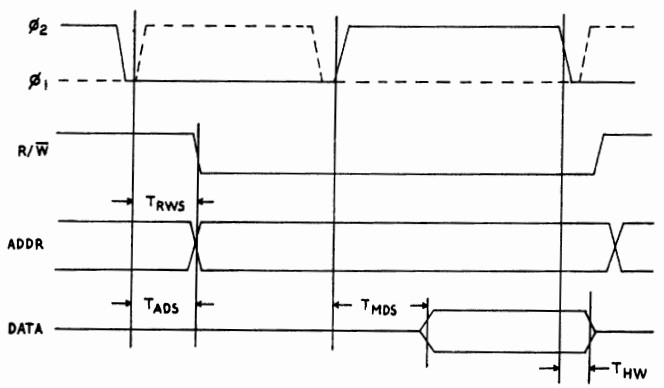


Figure 1. 6502 Write Timing

The R/W and address lines have a setup time (T_{RWS} and T_{ADS}) as 300 ns. maximum after the beginning of ϕ1. Data is available in a maximum of 200 ns. after the rise of ϕ2 (T_{MDS}). ϕ2 has a minimum width of 430 ns with a one micro second clock (1MHz). Thus data is available a minimum of 230 ns. before the fall (locking edge) of the ϕ2 clock. The data is held beyond the fall of ϕ2 for a minimum of 30 ns. (T_{HW}). Thus the 6502 is guaranteed to write successfully to any device with a set-up time requirement of 230 ns. or less, and a hold time requirement of 30ns. or less. Implicit in the timing is that the falling edge of ϕ2 is the locking edge. The 6502 generates no write strobe. A write strobe must be fabricated by NANDing ϕ2 with the inverted R/W signal, R/W. This gives a strobe that goes low during ϕ2 only for a write cycle. In family devices, ϕ2 and R/W are applied separately and the gating is performed internally.

6502 Read Timing

In a read operation, an external device puts its data on the data lines and it is locked internally into the 6502 at the end of ϕ2. This timing is shown in figure 2.

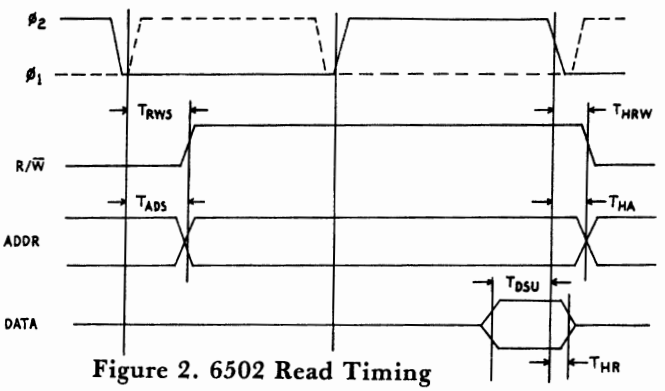


Figure 2. 6502 Read Timing

As in a write cycle, the address and R/W set-up time is a maximum of 300 ns. into ϕ1. The data set-up time is a minimum 100 ns. before ϕ2 (T_{DSU}). In addition, the data must be held a minimum of 10 ns. past ϕ2. Both the set-up and hold times for the 6502 in a read operation are quite short making it easy to read I/O devices from any MOS family.

When a device is read, the data is gated onto the bus by a read gate generated from the R/W line and a decoded address. The strobing is done internally by the processor. A gating signal allows the data to overlap the strobe. It will be seen in many 6502 systems that read gates often incorporate ϕ2. It would appear that this would cut off the data at strobe time, and violate the hold time requirement. First of all, the tri-state gate buffering the data probably has a delay of at least 15 ns. Secondly, if ϕ2 has gone through one or two gates of buffering, it will occur 15 to 30 ns. after the ϕ2 seen at the 6502. Thus the data will remain on the data bus 30 to 45 ns after ϕ2, even though ϕ2 appears to be used to cut data off.

Interfacing Non-family Devices

Meeting the read requirements of the 6502 is easy for almost any I/O device in any family. The problem occurs when the 6502 tries to write to the relatively sluggish devices of other families. Some of these chips have relatively long set-up and hold time requirements for a write operation. The set-up times are not often a problem, however. In order to be on the safe side, the chip manufacturer often quotes a rather conservative minimum hold time spec (like 100ns.) while quoting a typical spec of 30 or even 0 ns. To insure reliable performance, you must meet the minimum spec. The 6502 will guarantee only a 30 ns write hold time. What do you do when you want to write to a Western Digital 1771 floppy

controller chip with a longer hold time requirement? There are a number of ways to overcome this problem. First, however, we should consider an additional complication.

Earlier we saw that a little delay in the ϕ2 clock when used in a read gate was not a bad thing. However, delay in ϕ2 when used to generate a write strobe could be bad news. The 6502 provides for only 30 ns beyond the ϕ2 that it sees. A strobe generated from a delayed ϕ2 may in fact occur after the data has actually gone away. In this case, delay in the data path is beneficial. Any delay in ϕ2 greater than the delay in the data path takes away from the effective write hold time. Since few designers understand, let alone consider read/write timing, it is a wonder that most microcomputer systems work at all. In a typical single-board computer system, ϕ2 is generated on the board, buffered on some motherboard, and further buffered on the individual boards which plug into the motherboard. It is really questionable whether this buffering is really helpful. The fortunate thing for us all is that the hold time of the 6502 chips and the hold time requirement for I/O chips is almost always very conservatively speced.

Let us consider some approaches for getting a little extra hold time. Figure 3. shows perhaps the safest way. The data bus write buffer consists of a transparent latch followed by a tri-state gate. The latch is gated with ϕ2 so that the data to be written is locked into the latch at the end of ϕ2. The R/W line

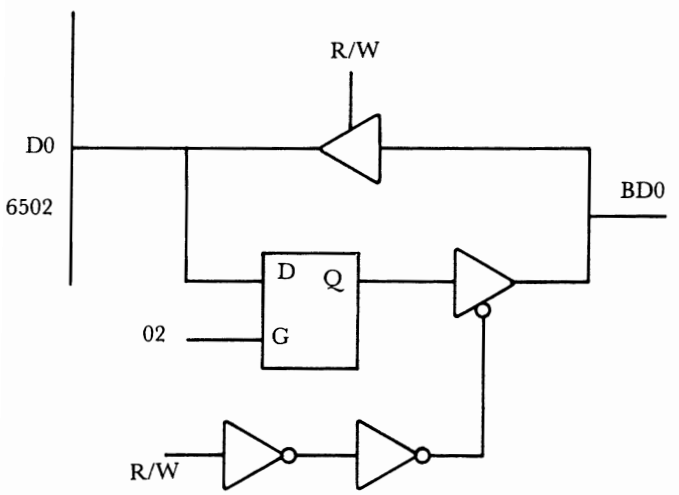


Figure 3. Write Hold Time Extender

ordinarily extends well into ϕ1 before changing. In fact the R/W line is not guaranteed to change for at least 300 ns. (T_{RWS}). To be on the safe side, we can always insert several gate delays in R/W so that the latched data will extend well beyond the end of ϕ2. This approach costs the addition of a pair of quad transparent latch chips, which may be very cheap insurance for reliable operation.

Another approach to getting a longer write hold time is to generate a write strobe which finishes earlier than ϕ2. This can be accomplished by hitting a one-shot with the regular write strobe. In this method, however, the longer hold time is at the expense of set-up time. If you have lots of set-up time to waste there's no problem. The width of the pulse will have to be adjusted carefully. Figure 4. shows the shortened write strobe, W* and set-up time trade-off.

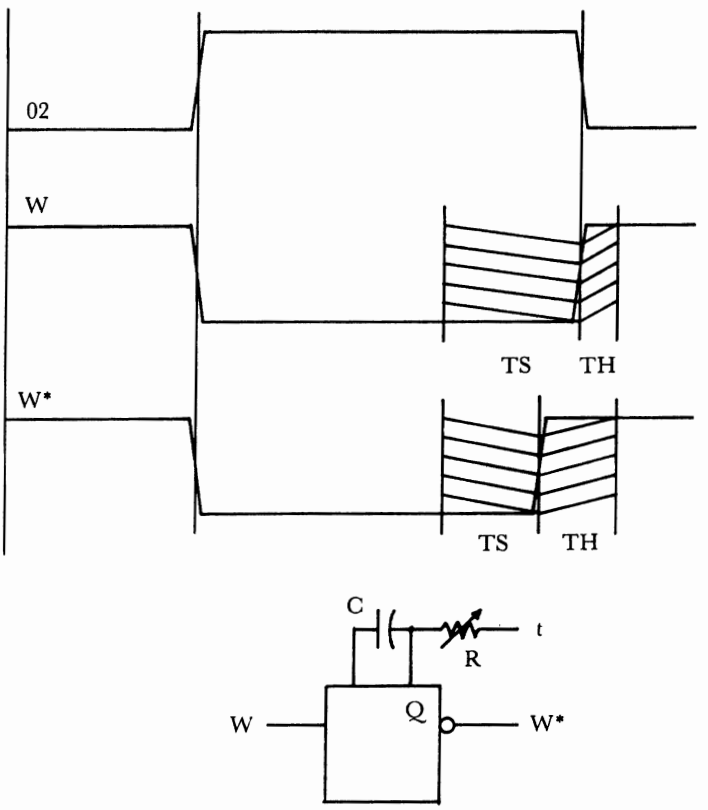


Figure 4. Shortened Write Strobe

Access Time

When considering RAM or ROM, we need to consider access time, which is the time it takes the data to reach the output after the address is stable. For the 6502 at 1MHz, the addresses are good 300 ns. into ϕ1. The data must be ready 100 ns. before the end of ϕ2. This gives us about 600 ns. of available access time. (The spec sheet guarantees 575.) If we could only steal another 75 ns., we could use much cheaper 650ns. RAM and EPROM. If we pay a $5 premium for a 2MHz 6502, and run it at 1MHz, we can get an extra 150 ns. from a shortened address set-up time. The few extra dollars for the faster processor could save a lot more bucks ion a system with lots of memory.

In summary, no serious, or at least no creative design can be undertaken without an understanding of read/write timing. It requires accommodating the set-up and hold time requirements of the I/O devices with the corresponding times of the processor. Tricks like latching the write data can be used to overcome any discrepancies. The information is available from the device spec sheet-. ©

An Upgrade for KIM MICROCHESS 1.0

Garold R. Stone
P.O. Box 153
Annapolis Junction, MD. 20701

If you have Peter Jennings' MICROCHESS program for the KIM-1 microcomputer you can teach it to play a significantly better game of chess without adding a single byte of expansion memory. This article describes a "patch" I have written for MICROCHESS which gives the computer a more flexible opening game and two new strategies for the middle and end game. Just load your copy of MICROCHESS, enter my code from the accompanying program listing along with the chess opening sample from table one, and play chess. There are no changes in the way you run the program. (For a description of the MICROCHESS program see KB, August 1978, page 74). For clarity I will use the term MICROCHESS only to refer to the original program as written by Peter Jennings. I will say "patch" to refer to the changes I am describing here.

Off the Shelf

The MICROCHESS I bought from Micro-ware Ltd. opens the game by playing from a pre-selected list of moves for a user chosen chess opening (Roy Lopez, French Defence, etc.). That opening list also contains one anticipated opponent move for each computer move. Things go well as long as the opponent makes the anticipated replies. But a human opponent seldom does that -- at least I don't. As soon as I make a novel move MICROCHESS permanently abandons the opening list. Whenever MICROCHESS is forced to quit the opening list too early, coherent development of pieces stops, the queen usually comes out too early, an ill-prepared attack is launched, and the computer loses its ability to castle (because castling is only possible from the opening list).

Compromises in 1.1K

Mr. Jennings points to these problems in his excellent documentation manual:

> "A major problem in the analysis is that there is only one strategy which is used for the opening, the middle game and the end game. This involves a considerable compromise of three different types of play."

The single strategy used by MICROCHESS is best suited for the middle game, where the capture of pieces dominates. In order to add a dynamic opening strategy which would emphasize the development and positioning of pieces, I had to settle for my own set of compromises, as you'll see. I should point out that Mr. Jennings seems to have surmounted this problem in the other versions of MICROCHESS he has written for microcomputers with more memory, such as the PET, TRS-80, and the APPLE.

The Opening

Table 1 shows my data format for eight opening development moves. Unlike in MICROCHESS, anticipated opponent replies are **not** listed. On each turn the **patched** program evaluates all of the computer's available moves. The available move which comes out with the highest evaluation is compared with the evaluation for the next legal move in my opening list and the higher of the two is selected as the computer's move for that turn. The development move is usually selected because its evaluation is always boosted by a threshold factor. I set the threshold factor high enough so that only moves with a significantly higher evaluation can override the development move. The higher the threshold, the more likely it is that the development move will be selected for that turn. Thus, the computer follows an opening game plan, responds to significant attack threats or capture opportunities, and then continues to carry out the opening game plan on the next turn by consulting the opening list again.

Books on chess openings and opening game strategy can serve as guides in writing new lists of development moves. Choose openings which are general in nature and do not depend on specific moves by the opponent. Specify each development move by giving the piece (variable DEVP), the square of origin (FROM), and the destination (TO), using the same notation as in MICROCHESS (see tables 2 and 3). Openings for white and black will require separate notation. Fill all unused locations in the opening list with the magic number 1F (hexadecimal), which causes those locations to be skipped because they are off the board.

Castling

As in MICROCHESS the computer's castling move must be completed for it by moving its rook after the computer signals castling by moving its King the necessary two squares. My added programming will prevent castling if the computer's King is off its starting square or if it would end up in check. The other rules for castling are not checked, however. If the computer castles illegally, then the move must be refereed. The simplest way is to use the "touch-move" rule -- once a player touches a piece it

Table 1
Opening Move Data

ADDR	VARIABLE	MOVE	WHITE	BLACK	COMMENT
00C3	.FACTOR		05	05	THRESHOLD FACTOR
00C4	.DEVP-1	N-KB3	06	06	PIECE
00C5	.FROM		01	06	ORIGIN
00C6	.TO		22	25	DESTINATION
00C7	.DEVP-2	P-KN3	0A	0A	PIECE
00C8	.FROM		11	16	ORIGIN
00C9	.TO		21	26	DESTINATION
00CA	.DEVP-3	B-KN2	04	04	PIECE
00CB	.FROM		02	05	ORIGIN
00CC	.TO		11	16	DESTINATION
00CD	.DEVP-4	P-K3	0F	0F	PIECE
00CE	.FROM		13	14	ORIGIN
00CF	.TO		23	24	DESTINATION
00D0	.DEVP-5	0-0	00	00	PIECE (KING SIDE CASTLE)
00D1	.FROM		03	04	ORIGIN
00D2	.TO		01	06	DESTINATION
00D3	.DEVP-6	K-QB3	07	07	PIECE
00D4	.FROM		06	01	ORIGIN
00D5	.TO		25	22	DESTINATION
00D6	.DEVP-7	P-Q4	0E	0E	PIECE
00D7	.FROM		14	13	ORIGIN
00D8	.TO		34	33	DESTINATION
00D9	.DEVP-8	(NO	1F	1F	
00DA	.FROM	(MOVE)	1F	1F	
00DB	.TO		1F	1F	

See Tables 2 and 3 for coding of Pieces and Squares

must be moved. Thus, the computer would have to move its King somewhere else, and you would enter that move for it. If there are no legal moves left for the King, then the computer must resign. This situation seldom comes up because I write openings which castle early enough to avoid the risk and annoyance of an illegal attempt.

Program Flow

What follows is a description of how the patched program works. MICROCHESS subroutines which are not defined in my accompanying program listing are in bold letters.

Whenever it is the computer's turn to move, MICROCHESS command loop **CHESS** calls my version of subroutine GO (see 03A2 in the program listing). MICROCHESS uses the value of a variable called STATE to keep track of what it's doing. State 4 guides the generation and evaluation of the computer's available moves. There are other states for generating potential opponent replies, etc. MICROCHESS subroutine **GNMX** (see 03AA) initializes some variables called "counts" for evaluating moves and then generates all moves available to the computer on that turn. **GNMX** calls MICROCHESS subroutine **JANUS** to calculate and evaluate the counts for each trial move. Based on the value in STATE, **JANUS** decides what to do next -- generate potential opponent replies for evaluation, calculate exchanges of pieces, etc. **JANUS** changes the value in STATE as it goes.

Table 2
Microchess Piece Notation and Storage

CODE	PIECE	MEMORY LOCATION COMPUTER	OPPONENT
00	KING	0050	0060
01	QUEEN	0051	0061
02	KING ROOK	0052	0062
03	QUEEN ROOK	0053	0063
04	KING BISHOP	0054	0064
05	QUEEN BISHOP	0055	0065
06	KING KNIGHT	0056	0066
07	QUEEN KNIGHT	0057	0067
08	KR PAWN	0058	0068
09	QR PAWN	0059	0069
0A	KN PAWN	005A	006A
0B	QN PAWN	005B	006B
0C	KB PAWN	005C	006C
0D	QB PAWN	005D	006D
0E	Q PAWN	005E	006E
0F	K PAWN	005F	006F

Table 3
Board Notation

Computer

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

OPPONENT

Note: Whether playing White or Black, the Computer's starting squares are always 00 through 17. Be sure to orient the playing board so that the lower left corner is black. The White Queen should be on a white square and the Black Queen should be on a black square.

Table 4
New Variables Used

ADDR	VARIABLE	COMMENT
00C3	.FACTOR	Threshold factor for opening moves
00DC	.OMOVE	MICROCHESS opening move flag
00DC	.OMOVE	Base for opening move array
00EF	.BKMOB	Number of legal moves for Opponent King
00F0	.BIAS	Receives threshold factor for legal list move

JANUS and portions of **GNMX** call each other recursively, again and again, until all of the computer's available moves have been evaluated in the light of all possible opponent replies. By the time program control returns from that very first call to subroutine **GNMX,** one move has emerged with an evaluation higher than all the others.

Then my patch searches the opening move list from the beginning to find the first piece (variable DEVP) which is still where it is supposed to be (FROM) (see 03B1). The move by this piece to its destination (TO) is checked for legality by a call into the middle of MICROCHESS subroutine **CMOVE.**

If the list move is legal, then the threshold factor (FACTOR) is stored in the variable BIAS for later use (see 03D8). MICROCHESS subroutine **JANUS** is called to do the counts for this list move and for the opponent's potential replies.

To evaluate these counts **JANUS** calls up my version of subroutine STRATEGY (see 1780-17C1). This is where the evaluation of the list move is boosted by adding the threshold factor which was stored earlier in the variable BIAS. Actually, this same subroutine STRATEGY is used by **JANUS** to evaluate any trial move but BIAS is always zero except for legal list moves. If the selected list move is not legal, then **JANUS** is not called to evaluate it, and no more list moves will be tried for that turn. This ensures that moves from the opening list are made in the order you wrote them. After the last list move has actually been moved, the variable OMOVE is set to zero and the opening list is ignored for the rest of the game (see 03AF).

As you exit subroutine STRATEGY you enter that portion of MICROCHESS which compares the evaluation of the current trial move with that of the best move so far, saving the better of the two as the new best move so far. This is also where MICROCHESS tests for check or checkmate before returning to **JANUS.** Control then passes to the MICROCHESS subroutine which takes the best trial move and actually moves it (see 03E3). The computer's move is flashed on the KIM display and the program returns to the MICROCHESS command loop, ready for the opponent to enter his move.

Middle and End Game

MICROCHESS sees only one and a half moves ahead. With this limited horizon it has trouble finding and closing in on the opposing King. To compensate for this I give a bonus of two points for moves inside a zone which surrounds the opposing King and moves along with it. The computer's Pawns and King do not get the bonus (see 179D).

Another strategy encourages moves which hem in the opposing King, in preparation for checkmate. The value of any trial move is decreased by the number of safe moves it leaves for the opposing King. This is the same as adding a point for each square denied to the opposing King. Since MICROCHESS calls subroutine **JANUS** to evaluate only legal moves, it was easy enough to put a subroutine call inside **JANUS** which would increment a mobility count (BKMOB) for each legal move found for the opponent King when the computer is checking for opponent reply moves during state zero (see 0112, 17D9,179A).

Both strategies come into play only after the opening list has been emptied, so as not to interfere with the development of pieces during the opening game (see 1796).

Evaluation

I approached move evaluation in much the same way as in MICROCHESS -- adding and subtracting weighted counts representing captures, position, and mobility for both sides. I did not use some of the counts generated by MICROCHESS and I created the new ones I described above. Given the severe memory restrictions, my goal was an evaluation formula which emphasizes immediate and tangible factors, such as position and the values of pieces captureable during the current turn. Less immediate factors, such as overall attack strengths, are given fractional weighting. These become influential only after more significant factors have cancelled each other out.

For now I've had to be satisfied with just breaking MICROCHESS of its habit of throwing away its pieces by occasionally making bad decisions about captures where pieces are exchanged. In my patch any piece the computer wants to capture must be greater than or equal to the most valuable piece the computer would lose by making that move (variable BMAXC). Only trial moves which pass this admittedly simplistic test are given an extra 20 hex points (see 17B1). There is more that could be done, like making better use of the MICROCHESS counts for exchanges involving up to three captures per side.

I hope I've made my point. All you need is a shoe horn and you can slip just about any changes you want into the 1.1K KIM MICROCHESS. You may pinch a few toes in the process, but the result is a KIM that plays better chess. By trying to "upgrade" MICROCHESS I really learned to appreciate what an excellent piece of work it is.

MICROCHESS is available on KIM cassette with documentation manual from Micro-Ware Ltd., 496 Albert St., Suite 7, Waterloo, Ontario, Canada, N2L 3V4

Abbreviated Instructions for Loading and Running MICROCHESS 1.0 UPGRADE

Load:

Enter (RS) to reset KIM
Enter (AD) 00F1 (DA) 00 to reset decimal flag
Enter (AD) 17F9 (DA) C1 to enter tape ID for program segment
Enter (AD) 1873 (GO) to start read routine of KIM
Press "Play" on cassette player
STOP recorder when display shows: 0000
Enter (RS) (AD) 1873 (GO) to read second program segment (same label "C1")
STOP recorder when display shows: 0000
Enter (RS) (GO) to start program execution

Playing:

Enter (C) on KIM hexpad keyboard to reset program for new game
Enter (PC) (for "play chess") because KIM plays first
After KIM gives its move, enter your move as FROM-TO according to the board notations in table 3 of the article. Keep typing until your move shows correctly, then enter (F) (PC).

```
                    0110              .BA $3A2
03A2- A2 04         0120 GO           LDX #$04      ; RESET BEST EVALUATION
03A4- 86 FA         0130              STX *BESTV    ;   SO FAR
03A6- 86 B5         0140              STX *STATE    ; STATE = 4; TRAIL MOVES
03A8- A2 12         0150              LDX #$12      ; ZERO COUNTERS & BIAS
03AA- 20 02 02      0160              JSR GNMX      ; GENERATE TRAIL MOVES
03AD- A4 DC         0170              LDY *OMOVE    ; OPENING LIST DONE?
03AF- 10 32         0180              BPL NODEVP    ;   - YES, MID-GAME
03B1- A0 E6         0190              LDY #$E6      ;   - NO, NEXT DEVP
03B3- C8            0200 NEXT         INY
03B4- C8            0210              INY  ;          INDEX OF DEVP
03B5- 84 DC         0220              STY *OMOVE    ; OPENING LIST EMPTY?
03B7- 10 2A         0230              BPL NODEVP    ;   - YES, MID-GAME
03B9- B6 DC         0240              LDX *DEVP,Y   ;   -NO, NEXT DEVP
03BB- 86 B0         0250              STX *PIECE
03BD- B5 50         0260              LDA *BOARD,X        ; DEVP LOCATION
03BF- C8            0270              INY  ;          INDEX OF  FROM
03C0- 48            0280              PHA  ;          (SAVE DEVP LOCATION)
03C1- 98            0290              TYA  ;          TRANSFER INDEX OF
03C2- AA            0300              TAX  ;            FROM INTO X
03C3- 68            0310              PLA  ;          DEVP LOCATION IN ACCUM
03C4- D5 DC         0320              CMP *FROM,X   ; DEVP AT ORIGIN?
03C6- D0 EB         0330              BNE NEXT      ;   - NO, GET NEW DEVP
03C8- E8            0340              INX  ;          INDEX OF  TO
03C9- B5 DC         0350              LDA *TO,X     ; CHECK LEGALLITY OF DEVP
03CB- 20 D1 02      0360              JSR CMOVE     ; FROM .FROM TO .TO
03CE- 30 13         0370              BMI NODEVP    ; NEQ = ILLEGAL MOVE
03D0- A6 B0         0380              LDX *PIECE    ;   - LEGAL MOVE
03D2- E0 08         0390              CPX #$08      ; IS PIECE A PAWN
03D4- 30 02         0400              BMI LEGAL     ;   NEG = NOT PAWN
03D6- 70 0B         0410              BVS NODEVP    ; SET = ILLEGAL PAWN CAPTURE
03D8- A6 C3         0420 LEGAL        LDX *FACTOR   ; LEGAL OPENING MOVE!!
03DA- 86 F0         0430              STX *BIAS     ; SET BIAS TO FACTOR
03DC- A2 04         0440              LDX #$04      ; EVALUATE OPENING MOVE
03DE- 86 B5         0450              STX *STATE    ;   AND PUT IT IN BESTV
03E0- 20 00 01      0460              JSR JANUS     ;   IF ITS THE BEST MOVE
03E3- A6 FA         0470 NODEVP       LDX *BESTV    ;   SO FAR
03E5- E0 0F         0480              CPX #$0F      ; RESIGN OR STALEMATE IF
03E7- 4C C2 17      0490              JMP CONT      ;   BESTV TOO LOW
                    0500 ;
                    0510              .BA $17C2
17C2- 90 12         0520 CONT         BCC MATE      ; (ORIGINAL MICROCHESS
17C4- A6 FB         0530 MV2          LDX *BESTP    ;   CODING)
17C6- B5 50         0540              LDA *BOARD,X        ; MOVE AND DISPLAY THE
17C8- 85 FA         0550              STA *BESTV    ;   BEST MOVE
17CA- 86 B0         0560              STX *PIECE
17CC- A5 F9         0570              LDA *BESTM
17CE- 85 B1         0580              STA *SQUARE
17D0- 20 4B 03      0590              JSR MOVE
17D3- 4C 00 00      0600              JMP CHESS     ; END COMPUTER'S TURN
17D6- A9 FF         0610 MATE         LDA #$FF      ; RESIGN OR
17D8- 60            0620              RTS
                    0630 ;
                    0640              .BA $1780
1780- A9 80         0650 STRATEGY     LDA #$80      ; EVALUATION = 80 + OR - SCORE
1782- 18            0660              CLC
1783- 65 EB         0670              ADC *WMOB     ; COMPUTERS'S MOBILITY
1785- 4A            0680              LSR A
1786- 18            0690              CLC
1787- 69 40         0700              ADC #$40      ; RESET EVAL TO 80 +OR- SCORE
1789- 65 ED         0710              ADC *WCC      ; COMPUTER'S ATTACK STRENGTH
```

```
178B- 38          0720              SEC
178C- E5 E5       0730              SBC *BCC      ; OPPONENT'S ATTACK STRENGTH
178E- 4A          0740              LSR A
178F- 4A          0750              LSR A         ; MOBILITY X 1/16
1790- 4A          0760              LSR A         ; ATTACK STRENGTH X 1/8
1791- 18          0770              CLC
1792- 69 70       0780              ADC #$70      ; RESET EVAL TO 80 +OR- SCORE
1794- 65 F0       0790              ADC *BIAS     ; ZERO UNLESS DEVP MOVE
1796- A4 DC       0800              LDY *OMOVE    ; NEGATIVE IF STILL DEVP
1798- 30 17       0810              BMI CAPTEST   ; MID-GAME IF POSITIVE
179A- 38          0820              SEC  ;          DEDUCT MOBILITY OF THE
179B- E5 EF       0830              SBC *BKMOB    ;   OPPONENT'S KING
179D- A6 B0       0840              LDX *PIECE    ; BONUS FOR MOVE INTO
179F- CA          0850              DEX  ;          OPPONENT'S KING ZONE
17A0- E0 07       0860              CPX #$07      ; NOT FOR COMPUTER'S KING
17A2- B0 0D       0870              BCS CAPTEST   ;   OR PAWNS
17A4- 48          0880              PHA  ;          (SAVE EVALUATION)
17A5- A5 60       0890              LDA *BK       ; LOCATION OF OPPONENT'S KING
17A7- 38          0900              SEC
17A8- E9 38       0910              SBC #$38      ; CALCULATE KING ZONE
17AA- C5 B1       0920              CMP *SQUARE   ; MOVE INTO ZONE?
17AC- 68          0930              PLA  ;          (RESTORE EVALUATION)
17AD- B0 02       0940              BCS CAPTEST   ; CARRY CLEAR IS IN ZONE
17AF- 69 02       0950              ADC #$02      ;   ADD BONUS, NEAR KING
17B1- A6 DD       0960 CAPTEST      LDX *WCAP0    ; IF COMPUTER'S CAPTURE
17B3- E4 E4       0970              CPX *BMAXC    ;   IS NOT GREATER THAN
17B5- 90 03       0980              BCC QUIT      ;   OR EQUAL OPP, QUIT
17B7- 18          0990 MOVEOK       CLC  ;          PASSES CAPTURE TEST
17B8- 69 20       1000              ADC #$20      ; POINTS FOR GOOD MOVE
17BA- 65 DD       1010 QUIT         ADC *WCAP0    ; POINTS FOR CAPTURE
17BC- 38          1020              SEC  ;          POINTS FOR OPPONENT'S
17BD- E5 E4       1030              SBC *BMAXC    ;   MAX CAPTURE IN REPLY
17BF- 4C 77 03    1040              JMP CKMATE    ; TEST FOR CHECKMATE
                  1050 ;
                  1060              .BA $17D9
17D9- D0 06       1070 BKMOVE       BNE OUTBK     ; RTS IF STATE NOT ZERO
17DB- C9 00       1080              CMP #$00      ; RTS IF NOT OPP KING'S
17DD- D0 02       1090              BNE OUTBK     ;   MOVE
17DF- E6 EF       1100              INC *BKMOB    ; COUNT LEGAL OPP KING
17E1- 60          1110 OUTBK        RTS  ;          MOVES
                  1120 ;
                  1130              .BA $0112
0112- E0 00       1140              CPX #$00      ;COUNT LEGAL REPLY MOVES
0114- 20 D9 17    1150              JSR BKMOVE    ;   FOR OPPONENT'S KING
0117- EA          1160              NOP
                  1170 ;
                  1180              .BA $200
0200- A2 11       1190              LDX #$11      ; CLEAR COUNTERS, NOT BIAS
                  1200              .EN
```

PROGRAM TRANSFERS (PET TO KIM)

Joseph A. Dilts
Assoc. Prof. of Chemistry
Univ. of North Carolina
Harvey B. Herman
Prof. of Chemistry
Univ. of North Carolina

In a recent 6502 User Notes editorial (Issue #14, P. 27) you propose a method to transfer BASIC programs from PET to KIM. Your editorial prompted us to document our method for your readers, as we have been doing this for almost a year now.

There are several possible methods to communicate between computers. One way, memory to memory transfer using parallel ports, has already been published ("KIM-1 Talks to PET", PET User Notes, Vol. 1 #5, p 6). However, as you point out, tokens for various Micros of BASICs differ. Consequently, a simple transfer will not work for BASIC programs.

Another way, the way we picked, is to send an ASCII version of the program from the IEEE port of the PET to the serial interface of the KIM. This method requires both an IEEE/RS232 serial adapter and a RS232/KIM adapter. We used an IEEE adapter manufactured by Connecticut Microcomputer (PET ADA 1200 $170 assembled) which one of us (J.A.D.) uses with a word processing program (CMC). We constructed a modified version of the RS232/KIM adapter published in your journal (Issue #4, p 6) to complete the connection from PET to KIM.

It is very important to set the slide switches on the PET ADA 1200 properly. One combination that works for more than one application is:

1 parity bit
2 stop bits (10010 in switches 1-5)
7 characters even parity

The baud rate of the KIM serial line can be set with a serial terminal and the reset/rubout sequence or with the KIM keyboard (change locations $17F2/$17F3, see 6502 User Notes. Issue #6, p 11 for typical values.) We worked with a 300 baud terminal and the IEE/RS232 interface set at 300 baud.

The first method we tried was similar to the suggestion in your editorial. KIM BASIC was brought up as always using the terminal. A NEW command was given, the terminal was carefully disconnected from KIM and the PET/adapter substituted (pin 2 on the KIM to pin 3 on the PET/adapter). A PET program which we wished to transfer to KIM was loaded and listed to the IEEE bus using the following sequence (on the PET).

```
OPEN 6, 6
CMD 6
LIST
PRINT #6
CLOSE 6
```

The terminal was reconnected and the program listed on KIM. To our horror we found this method generally only transferred the **first** program line as apparently a delay or hand shaking is necessary after the first carriage return. One of us (HBH) uses an X-off/X-on sequence to properly load paper tapes on KIM thereby circumventing the same problem. We could not use this method here and we did not see an easy way to modify the PET ADA 1200 adapter for handshaking.

We tried another approach; one that eventually worked well. Since we saw the problem as necessitating a delay after carriage return, perhaps software, rather than hardware, could accomplish this. A BASIC program was written which could read an ASCII file from tape and send each character to the IEEE bus individually. The program adds a proper delay after the carriage return character is sent and gives KIM BASIC enough time to digest each line.

It is easy to make any PET BASIC program into a file on cassette tape after it is loaded.

```
OPEN 1, 1, 1
CMD 1
LIST
PRINT #1
CLOSE 1
```

The PET program, GET, (shown in the figure) which reads the file is loaded next. (Line 45 is incomplete, see below for reason, and needs GOSUB 60 after the colon). The KIM terminal is disconnected and PET connected to KIM. GET is executed on the PET and causes the program file, made previously, to be sent character for character over the IEEE bus to KIM, inserting delays when necessary. KIM's terminal is reconnected and if the PET program happens to be directly compatable with KIM's BASIC, it can be run immediately without modification.

Some minor problems may arise:

1. KIM's BASIC has a line buffer limitation of 72 characters. Lines longer than 71 characters are truncated by our program to fit this line length (c.f. line 45 above) and have to be reformulated after transfer.
2. KIM uses " " and "@" as special characters for character and line delete. If the PET

program uses these characters it may be advantageous to temporarily change locations $2440 and $243C (in our version of 9 digit KIM BASIC) to other ASCII characters to avoid trouble.

3. KIM and PET do not have exactly the same BASIC language. Neither do they have the same hardware. Some commands in PET BASIC will have to be translated to properly operate on KIM. Locations specific to PET will have to be relocated to be compatable with KIM. It may not be practical to transfer all programs. Users will have to use their judgement.

Even though we recognized that problems exist we have successively moved very long PET programs (close to 8K) and made them operate on KIM. As a short, possibly bad, example we transferred the GET program itself to KIM and listed it on a KIM operated teletype (110 baud). This program cannot execute on KIM because of software and hardware differences. It also illustrates the line length buffer limitation (line 45). As long as users are aware of the potential trouble spots they should have no difficulty.

We have also used the PET ADA 1200 adapter in other applications. The adapter was originally designed to directly drive a printer off the IEEE bus. When the terminal is not in proximity to the PET but is near a telephone and modem it is possible to make listings over phone lines. We have used the Pennywhistle 103 Modem to transmit PET programs and output on a high channel frequency, to terminals whose modem is receiving on the same frequency. A simple list to the IEEE bus (with CMD) or data output of the bus is all that's required. If the terminal also requires an extra delay after carriage return more bits per character (9) could suffice or the GET program used. The remote device could also be a computer. However, we have not tried this. The best part of the remote printing operation is to watch the faces of the folks in the computer room when they spy a strange (to them) dialect of BASIC printing on their terminal.

GET PROGRAM

```
10 OPEN1,1,0
15 OPEN6,6
17 FORI=1to500:NEXTI
18 PRINT#6,CHR$(15)
20 GET #1,C$
30 IF (ST) AND 64 THEN 500
40 IFASC(C$)=13THEN PRINT L$:PRINT#6,L$:L$="":FOR
   I=1TO500:NEXTI:GOTO20
45 L$=L$+C$:IFLEN(L$)=71THENPRINTL$:PRINT#6,L$:L$=
   "":FORI=1TO500:NEXT:
50 GOTO20
60 GET#1,C$:IF(ST)AND64THEN500
70 IFASC(C$)=13THENRETURN
80 GOTO60
500 CLOSE1
OK
```

Part 1: Implementing the IEEE-488 Bus on a SYM-1

DESIGNING AN IEEE-488 RECEIVER WITH THE SYM

Larry Isaacs, COMPUTE. Staff

This article is the first in a series on the use of a single board computer as a dedicated interface. In this section I will describe the design of an interface connecting a Spinwriter to the PET IEEE-488 Port using a SYM-1. If you have a need for an interface or controller, but not much experience using single board computers, this series should provide some pointers on how to go about implementing one.

In this article, the discussion of the IEEE-488 Bus will be limited to that which is relevant to the PET, and to how the PET sends data to a printer. Also, when the software to be presented is too general to give the actual assembly language, it will be given in PASCAL. The listings should be readable, even if you haven't had much exposure to PASCAL. The names used in the PASCAL listings will correspond to the names used in the assembly language listings. The following notes should help if you haven't seen PASCAL before.

1. PASCAL uses ": =" for the assignment operator; " =" is used only for comparisons.
2. The ";" is used to separate statements.
3. When statements are enclosed between a "begin" and an "end;", it means that that block of statements may be treated as if they were one statement. The programs are indented to help show which "end;"'s go with which "begin"'s.

DIVIDE AND CONQUER: A STARTING POINT

One of the most effective ways to handle design problems is to successively divide the required functions into small sets of sub-functions. Once the complexity of a sub-function has been reduced to a manageable level, then it is implemented.

The first division of the PET-to-Spinwriter interface is shown in Listing 1.

Listing 1.

```
program PETTOSPINWRITERINTERFACE;

procedure INIT; begin...end; {initialization}
procedure PRINT; begin...end; {send chr. to Spinwriter}
procedure CYCLE; begin...end; {get byte from IEEE}
procedure INTERFACE; begin...end; {main interface software}
begin {PET to Spinwriter Interface}
   INIT;
   INTERFACE
end.
```

Here, the interface task has been divided into four sub-functions. The task of the INIT procedure will become apparent as the other parts of the software are written. The INTERFACE procedure will contain the intelligence of the interface. The exact function of INTERFACE can't be determined yet, so this sub-function will be dealt with later.

The PRINT and CYCLE sub-functions will be used by INTERFACE to communicate with the PET and the Spinwriter. Unlike the others, the functions of PRINT and CYCLE are sufficiently narrow in scope to be implimented at this point. Both will involve dealing with hardware as well as software. But once done, most of the hardware details will be taken care of.

PRINT

The purpose of this routine is to handle all of the requirements for communicating with the Spinwriter. To do this, one must first consult the Spinwriter and SYM-1 documentation:

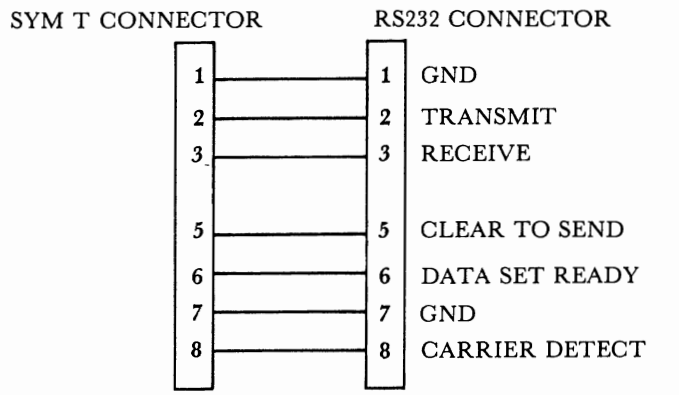
Our Spinwriter has a serial interface. This means we can use the serial interface software provided in the SYM-1 Monitor to send characters to the Spinwriter. The Spinwriter Product Description manual reveals that CARRIER DETECT (pin 8 on the RS232 connector), DATA SET READY (pin 6), and CLEAR TO SEND (pin 5) must be high (between +3 and +12 volts) for the Spinwriter to operate. This was simple to take care of since the SYM-1 provides this voltage at the corresponding locations of the T connector.

The Product Description manual also reveals a way of increasing throughput by using the ETX/ACK protocol. This makes use of the 256 character receive buffer found in the Spinwriter. You use this protocol by sending data blocks of up to 254 characters followed by an ETX character (control C). When the Spinwriter withdraws the ETX character from the receive buffer, it transmits an ACK character (control F) to indicate the buffer is empty and ready for another block of characters. This will allow the SYM to transmit at 1200 baud, and let the Spinwriter print at its maximum speed. All of this leads to Figure 1 which shows how to attach the required RS232 connector to the SYM.

After the proper initialization, the OUTCHR subroutine in the SYM Monitor can be used to send characters to the Spinwriter, and the INCHR

subroutine to receive the ACK character involved with the protocol.

Figure 1. SYM to Spinwriter Hardware



The assembly language for PRINT is shown in Listing 2.

Listing 2

```
0111- 20 47 8A  1570 PRINT   JSR OUTCHR  ;PRINT AND INC. .COUNT
0114- E6 00     1580         INC *COUNT
0116- D0 0C     1590         BNE RETURN
0118- A9 03     1600 ACK     LDA #$03    ;ASCII ETX
011A- 20 47 8A  1610         JSR OUTCHR
011D- 20 58 8A  1620         JSR INCHR   ;WAIT FOR ACK
0120- A9 02     1630         LDA #$02
0122- 85 00     1640         STA *COUNT
0124- 60        1650 RETURN  RTS
```

CYCLE

The function of CYCLE is to read the byte on the data lines during a byte transfer cycle on the IEEE bus. In some cases, the INTERFACE sub-function will need to know the state of some of the other signals during the transfer. CYCLE should therefore sample the signal lines as well.

All the information needed for the IEEE part of the interface can be found in the Commodore CBM manual. The information in this section will deal only with the byte transfer cycle. The remaining information will be presented in the next part of this article. In the discussion below, reference is made to active and inactive devices. An active device is simply one which is participating in the current transfer cycle. Before continuing, you may want to refer to Table 1 which lists the IEEE signals and a brief description of their function. In this table, Listener refers to the receiving device, and Talker refers to the sending device.

The IEEE bus make use of three handshake signals. These are the NRFD, NDAC, and DAV lines. When the CYCLE routine is entered, both the PET, the SYM, and any other active devices are expecting a byte transfer to take place. This means that NRFD and NDAC are low, and NDAV is high. At this point CYCLE sets NRFD high, indicating the SYM is ready to proceed with the byte transfer. Since the NRFD signal line is Wire-ORed, any active device can hold the NRFD line low. This means the cycle doesn't proceed until all active devices indicate they are ready.

Once NRFD goes high, the PET responds by placing the byte to be transferred on the DIO lines and then setting DAV low to indicate valid data. When CYCLE sees DAV go low, it should read the data lines and then sample the signal lines. Now CYCLE sets NDAC high to indicate that the data has been accepted. The NDAC line is also Wire-ORed, so the other active devices must indicate they have accepted the data before the cycle can finish.

When the PET sees the NDAC line go high, it sets DAV low. Once CYCLE sees the DAV line go high, it resets NDAC to the low state completing the cycle. Now CYCLE returns to the calling software. Refer to Listing 3 for the assembly language for this routine.

Listing 3

```
00E7- A9 03     1390 CYCLE   LDA #$03
00E9- 8D 00 A8  1400         STA @2IORB   ;NRFD=1 NDAC=0
00EC- 2C 00 A8  1410 @1      BIT @2IORB   ;TEST DAV
00EF- 70 FB     1420         BVS @1       ;BRANCH IF DAV=1
00F1- 6A        1430         ROR A
00F2- 8D 00 A8  1440         STA @2IORB   ;NRFD=0 NDAC=0
00F5- AD 01 A8  1450         LDA @2IORA
00F8- 49 FF     1460         EOR #$FF
00FA- 85 02     1470         STA *DATA
00FC- AD 00 A8  1480         LDA @2IORB
00FF- 85 01     1490         STA *SIGNALS
0101- A9 00     1500         LDA #$00
0103- 8D 00 A8  1510         STA @2IORB   ;NRFD=0 NDAC=1
0106- 2C 00 A8  1520 @2      BIT @2IORB
0109- 50 FB     1530         BVC @2       ;BRANCH IF DAV=0
010B- A9 01     1540         LDA #$01
010D- 8D 00 A8  1550         STA @2IORB   ;NRFD=0 NDAC=0
0110- 60        1560         RTS
```

TABLE 1

NAME	SET BY	DESCRIPTION
DI01-DI08	Talker	Data Input/Output. These lines carry the commands and data.
NRFD	Listener	Not Ready for Data. When low, it means the device is not ready to receive data. It is set high when the device is ready.
DAV	Talker	Data Valid. When high, it means the data on the data lines is not valid. It is set low once all NRFD goes high and valid data has been placed on the data lines.
NDAC	Listener	Not Data Accepted. When low, it means that the data has not been accepted. It is set low once DAV goes low and the data has been latched.
ATN	Talker	Attention. Signals that the byte on the DIO lines is a command.
EOI	Talker	End Or Identify. Signals that the last data byte is being transferred.
IFC		Interface Clear. Resets all devices.

Improved Pulse Counting Software For The 6522 VIA

Marvin L. De Jong
Dept. of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

Ever since I began playing with the 6522 I have been trying to find a program that would use the 6522 to count pulses for an *exact* one second interval. By exact I mean one million clock cycles, not one million plus or minus several instruction intervals. Of course, it should be noted that if the system clock frequency is not *exactly* one Megahertz then an error of several instruction intervals may not be particularly important. In this connection, the measurements I have made of clock frequencies on a few KIM-1s and one AIM 65 show that errors of several hundred parts per million are not unusual, so if your twenty-four hour clock runs slow or fast, do not be surprised.

In any case, assuming that the system clock frequency is precise to say one part per million, the program supplied in this note will count pulses for an interval that is as precise as the system clock frequency. The assembly language program to count pulses for exactly one second (one million clock cycles) is given in Table 1, and the simple interface circuit it requires is given in Figure 1. A BASIC program to convert the pulse count to decimal and display it is given in Table 2. This program works on my AIM 65, and it will probably have to be modified for other machines.

The assembly language program in Table 1 makes use of the T1 timer in its one-shot mode with PB7 enabled. That is, the T1 timer is programmed to produce a time interval of 50,000 clock cycles, and during that interval of 0.05 s PB7 is held at logic zero. Refer to Figure 1 and note that when PB7 is at logic zero the pulses from some external device will be gated to PB6, the pulse-counting pin for the T2 counter/timer. In order to produce pulse-counting intervals that are longer than 0.05 s, the T1 timer is reloaded and started N times, where N is an eight-bit number stored in a memory location labeled CNTR in Table 1. Thus, if N = 2 the counting interval is 0.1 s, if N = 20 the counting interval is 1.0 s, and if N = 200 the counting interval is 10 s. These numbers must be converted to hexadecimal numbers before using them in the program.

While T1 is timeing-out it is read continuously so that it may be reloaded and started after *exactly* 50,000 clock cycles. This prevents PB7 from reaching logic one any time during the N timing intervals. If we were to allow T1 to time-out and then reload and start it, PB7 would toggle from logic zero to logic one and back to logic zero, with the possibility of producing an extraneous count on PB6. Thus, the program loop starting from REPEAT in Table 1 and ending with DUMMY in the same listing is *tuned* to take exactly 50,000 clock cycles. Each time through the loop N is decremented, until it reaches zero at which time T1 is finally allowed to time-out for the last time.

When T1 times out for the last time, no more pulses will reach PB6. At this time the interrupt flag register (IFR) on the 6522 is read first. If the T2 flag is set, then the pulse count was greater than \$010000 (65536_{10}) because the T2 counter was initially loaded with \$FFFF. If the T2 interrupt flag (IFR5) is set, then the most-significant byte, PLUSHI, of the pulse-count storage locations is incremented. Otherwise it is cleared. After this operation, the T2 counter is read and the resulting pulse counts are loaded into PLSMI, the middle byte of the three-byte pulse-count storage locations, and PLSLO, the least-significant byte of the pulse-count storage locations. The program then uses a JMP instruction to return to the BASIC calling program given in Table 2. Other BASICs may use a different return technique.

The most obvious application of pulse counting is a simple frequency meter. The programs and interface described here will count at a maximum pulse count of 131,071 counts during whatever counting interval (0.1 s, 1.0 s, or 10 s) you choose. Note that 131,071 = \$01FFFF. Other applications include voltage-to-frequency converters and temperature-to-frequency converters. Commercial tachometer pickups produce a pulse rate that is proportional to the angular velocity (RPM) of a rotating shaft. The 6522 can be used to measure this pulse rate and the microcomputer can convert it to rotations per minute. The 6522 can also be interfaced to Geiger counters (GM tubes) or scintillation detectors to count nuclear events. There are a variety of new transducers appearing (temperature, light intensity, pressure) that can be used with a V/F converter to produce a pulse rate that is directly proportional to the physical quantity being measured. Although direct analog-to-digital (A/D) conversion is faster than pulse counting, it usually requires a much more sophisticated interface. In applications where speed is not a problem, investigate the possibility of using this simple program and interface.

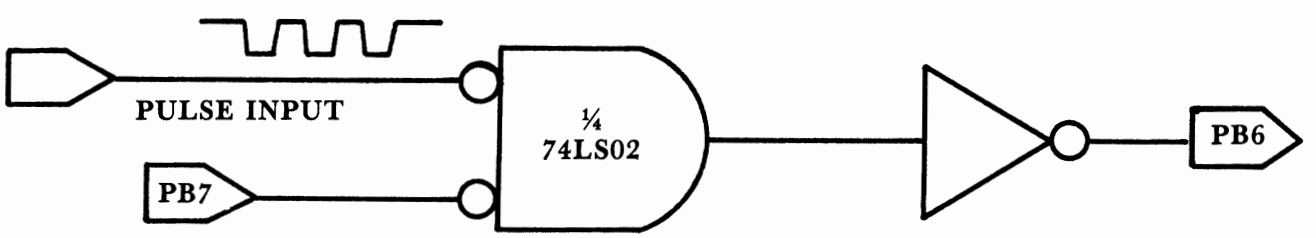
Table 1. Simple pulse counting program for the 6522.

```
$0F00 A9 01      START   LDA $80      Make PB7 an output pin by loading
$0F02 8D 02 A0           STA PBDD     one into the data direction register.
$0F05 A9 A0              LDA $A0      Set up the ACR so T1 runs once, PB7
$0F07 8D 0B A0           STA ACR      enabled, and T2 counts pulses.
$0F0A A9 14      HERE    LDA $14      Set up counter to do 20 ($14) intervals
$0F0C 85 30              STA CNTR     of 0.05s, totaling one second.
$0F0E A9 FF              LDA FF       Initialize T2 to start with
$0F10 8D 08 A0           STA T2LL     $FFFF and count down.
$0F13 8D 09 A0           STA T2CH     T2 is now ready to count when PB7
$0F16 A9 4F              LDA $4F      goes to logic zero.
$0F18 8D 04 A0           STA T1LL     Set up T1 to count 5000 clock
$0F1B A9 C3      REPEAT  LDA $C3      pulses. $C34F + 1 = 50000.
$0F1D 8D 05 A0           STA T1LH     Start T1, PB7 to logic zero.
$0F20 AD 05 A0   WAIT    LDA T1CH     Read the T1 counter, high-order byte.
$0F23 D0 FB              BNE WAIT     Wait until it is zero. These
$0F25 AD 04 A0   LOOP    LDA T1CL     instructions are part of a tuned
$0F28 C9 19              CMP $19      loop designed to wait exactly 50000
$0F2A B0 F9              BCS LOOP     cycles before starting T1 again.
$0F2C C6 30              DEC CNTR     The loop is repeated until the
$0F2E EA                 NOP          contents of CNTR = 0.
$0F2F 90 00              BCC DUMMY    These two dummy instructions tune
$0F31 D0 E8      DUMMY   BNE REPEAT   the loop.
$0F33 A9 00              LDA $00      Clear the most-significant byte of
$0F35 85 33              STA PLSHI    the pulses counted.
$0F37 AD 0D A0           LDA IFR      Read the IFR to see if count went
$0F3A 29 20              AND $20      through zero. Mask bits other than
$0F3C F0 02              BEQ OVER     T2 flag. If it was set, add $010000
$0F3E E6 33              INC PLSHI    to pulse counter.
$0F40 38         OVER    SEC          Otherwise, set carry flag and
$0F41 A9 FF              LDA $FF      perform subtraction to see how many
$0F43 ED 09 A0           SBC T2CH     pulses were counted.
$0F46 85 32              STA PLSMI    Result into middle byte of pulse
$0F48 A9 FF              LDA $FF      counter.
$0F4A ED 08 A0           SBC T2CL
$0F4D 85 31              STA PLSLO    Result into low-order byte of pulse.
$0F4F 4C D1 C0           JMP BASIC    Return to BASIC.
```

Table 2. Counting Pulses with a BASIC program.

```
10 REM THIS PROGRAM REQUIRES THE MACHINE LANGUAGE ROUTINE IN TABLE 1.
20 POKE 04,00: POKE 05,15
30 Y = USR(0)
40 X = PEEK(49) + 256*PEEK(50) + 65536*PEEK(51)
50 PRINT X; "PULSES PER SECOND"
60 GO TO 30
70 END
```

Figure 1.



Interface circuit for the pulse-counting program of Table 1. The inverter can be implemented with one of the other gates on the 74LS02 chip. The incoming pulse train must be at TTL logic levels.

PERFECT AIM

ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER

- Professional Appearance
- Striking Grey and Black Color Combination
- Protects Vital Components

ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65

- All Switches Accessible
- Integral Reset Button Actuator
- Easy Paper Tape Replacement

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC

- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

EASILY ASSEMBLED

- Absolutely No Alteration of AIM-65 Required
- All Fasteners Provided
- Goes Together in Minutes

AVAILABLE FROM STOCK

- Allow Three to Four Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME _____

STREET _____

CITY _____ _____

STATE _____ ZIP _____

SAE 1-1 PLEASE SHIP PREPAID _____ SAE 1-1(s)
@ $43.50 each
California Residents Please Pay
$46.33 (Includes Sales Tax)

SAE 1-2 PLEASE SHIP PREPAID _____ SAE 1-2(s)
@ $46.50 each
California Residents Please Pay
$49.52 (Includes Sales Tax)

enclosures group

771 bush street
san francisco, california 94108

***TM Rohm & Hass** **Patent Applied For**

PRINTING A SYMBOL TABLE FOR THE AIM-65 ASSEMBLER

Richard F. Olivo
Biological Sciences, Smith College
Northampton, MA 01063

The assembler for Rockwell's AIM 65 makes assembly-language programming very convenient, particularly in conjunction with the excellent editor that is part of AIM 65's monitor. However, the assembler does not include an option to print the symbol table, although it does create such a table in memory. The following program is one way of decoding and printing the symbol table. In revising a program, a print-out of the symbol table can be very helpful.

On entering the AIM 65 assembler from the monitor, you are asked for the addresses that start and end the symbol table. The assembler places your answers in zero-page addresses 3A, 3B ("FROM") and 3E, 3F ("TO"). After assembly, the total number of symbols is available in addresses 0B, 0C (in high, low order). The symbol table itself consists of sequential eight-byte entries. The first six bytes of each entry are the symbol name, in ASCII characters (the assembler enters spaces if the symbol is less than six characters), and the last two bytes are the symbol's address, in hex notation.

The program to print the table reads through the table using indirect addressing indexed by Y. It establishes the variable ADDR (at locations 00 and 01), which provides the address of the first character of the current symbol. ADDR is initially set equal to the address in "FROM (3A, 3B); it is incremented by eight after each symbol is printed. For each symbol, the Y register is incremented from zero to seven to access the successive bytes of that symbol.

A second variable, COUNT (addresses 02 and 03), keeps track of the number of symbols that remained to be printed. COUNT is initially set equal to one less than the total number of symbols (from addresses 0B and 0C), and it is decremented by one after each symbol is printed. After COUNT reaches zero (the last symbol is numbered zero, which is why the initial count is one less than the total), the program exits and prints the total number of symbols in hex notation. The program uses AIM monitor subroutines to print the ASCII and hex characters. It also turns the AIM printer on and off at the start and end of the table, which I find very handy.

The listing given below places the program at locations 0200-027D, which are available on every AIM 65. The program could of course be placed in other memory locations, and it would be very convenient in a PROM. At the end of the listing, the program was run to list its own symbol table.

```
--------------------
==0000 BLANK=$E83E

==0000 CRLOW=$EA13

==0000 EQUAL=$E7D8

==0000 PRIASC=$E97A

==0000 PRIFLG=$A411

==0000 PRIHX2=$EA46

==0000 ADDR=0
==0000 COUNT=ADDR+2
==0000
       *=$0200
--------------------
INITL ADDR,COUNT,Y

==0200
;"FROM" = 3A,3B
==0200 SYMTBL
A53A   LDA $3A
8500   STA ADDR
A53B   LDA $3B
8501   STA ADDR+1
;ADDR ACCESSES TABLE

A50B   LDA $0B
8502   STA COUNT
A50C   LDA $0C
8503   STA COUNT+1
==0210
;COUNT=SYMBOLS TO GO
C603   DEC COUNT+1
;FIRST SYMB=0, NOT 1

A000   LDY #0
;INDX 8 BYTES/SYMBOL
A980   LDA #$80
8D11A4 STA PRIFLG
;TURN PRINTER ON
2013EA JSR CRLOW
2013EA JSR CRLOW
;SKIP 2 LINES AT TOP
--------------------
MAIN LOOP

==021F SYMLP
B100   LDA (ADDR),Y
C006   CPY #6
;BYTES 0-5 =ASCII
F007   BEQ SPACE
;PRINT 6 ASCII CHAR.
207AE9 JSR PRIASC
C8     INY
4C1F02 JMP SYMLP

;PRINT SPACE & EQUAL
==022C SPACE
48     PHA
203EE8 JSR BLANK
20D8E7 JSR EQUAL
203EE8 JSR BLANK
;NEXT 2 BYTES = HEX
68     PLA
2046EA JSR PRIHX2
C8     INY
B100   LDA (ADDR),Y
==023D
2046EA JSR PRIHX2
2013EA JSR CRLOW
;HAVE PRINTED 1 LINE
--------------------
DECR COUNT & TEST

C603   DEC COUNT+1
A9FF   LDA #$FF
C503   CMP COUNT+1
;FF = BORROW
D006   BNE NXTADR
C602   DEC COUNT
==024D
C502   CMP COUNT
;FF = DONE
F012   BEQ DONE
--------------------
UPDATE ADDRESS

==0251 NXTADR
18     CLC
A500   LDA ADDR
;LOW BYTE
6908   ADC #8
8500   STA ADDR
A501   LDA ADDR+1
;HIGH BYTE
6900   ADC #0
8501   STA ADDR+1
A000   LDY #0
4C1F02 JMP SYMLP
--------------------
PRINT TOTAL & EXIT

==0263 DONE
2013EA JSR CRLOW
A50B   LDA $0B
2046EA JSR PRIHX2
A50C   LDA $0C
2046EA JSR PRIHX2
;PRINT TOTAL, SKIP LN
2013EA JSR CRLOW
==0273
2013EA JSR CRLOW
A900   LDA #0
8D11A4 STA PRIFLG
;TURN PRINTER OFF
4C82E1 JMP $E182
;JUMP TO MONITOR

       .END

BLANK  = E83E
CRLOW  = EA13
EQUAL  = E7D8
PRIASC = E97A
PRIFLG = A411
PRIHX2 = EA46
ADDR   = 0000
COUNT  = 0002
SYMTBL = 0200
SYMLP  = 021F
SPACE  = 022C
NXTADR = 0251
DONE   = 0263

000D
```

EXCERT, INCORPORATED

* * * AIM-65 * * *

SPECIAL

A65-4AB AIM-65 w/4K RAM
Assembler & BASIC ROM **$595**

P/N		QTY 1-9
A65-1	AIM-65 w/1K RAM	$375
A65-4	AIM-65 w/4K RAM	$450
A65-A	Assembler ROM	$85
A65-B	BASIC ROM	$100

SPARE PARTS (When Available)

A65-P	Printer	$40
A65-D	Complete Display Bd.	$65
	w/Exchange of Old Bd.	$40
A65-K	Keyboard	$40

ACCESSORIES

P/NO. **QTY 1-9**

Power Supplies (fully AIM-65 Compatible)

PRS3 +5V at 3A, +24V at 1A w/mtg hardware, cord, etc. **$65**
PRS4 +5V at 2A, +24V at .5A w/mtg hardware, cord, etc. **$50**

From The Enclosure Group

ENC1 AIM-65 case w/space for PRS3/PRS4 .... **$45**
ENC1A AIM-65 case w/space for PRS3/PRS4 and one expansion board **$49**

Cases with Power Supplies

ENC3 ENC1 w/PRS3 mounted inside **$115**
ENC3A ENC1A w/PRS3 mounted inside..... **$119**
ENC4 ENC1 w/PRS4 mounted inside..... **$100**
ENC4A ENC1A w/PRS4 mounted inside..... **$104**

From The Computerist, Inc.

MCP1 Mother Plus[tm] - Dual 44 pin mother card takes MEB1, VIB1, PTC1, fully buffered, 5 expansion slots underneath the AIM..... **$80**
MEB1 Memory Plus[tm] - 8K RAM, 8K PROM sockets, 6522 I/O chip and programmer for 5V EPROMS **(w/cables $215)**..... **$200**
PTC1 Proto Plus[tm] - Prototype card same size as KIM-1 MEB1, VIB1..... **$40**
VIB1 Video Plus[tm] - Video bd w/128 char, 128 user char, up to 4K display RAM, light pen and ASCII keyboard interfaces w/cables.... **$245**

P/NO. **QTY 1-9**

From Seawell Marketing, Inc.

MCP2 Little Buffered Mother[TM]-Single 44 pin (KIM-4 style) mother card takes MEB2, PGR2, PTC2 and PI02. Has on board 5V regulator for AIM-65, 4 expansion slots. Routes A&E signals to duplicates on sides w/4K RAM **$199**
MEB2 SEA 16[TM]-16K static RAM bd takes 2114L w/regulators and address switches 16K..... **$325**
PGR2 Prommer[TM]-Programmer for 5V EPROMS w/ROM firmware, regulators, 4 textool sockets, up to 8 EPROMS simultanously, can execute after programming **$299**
PI02 Parallel I/O Bd w/4-6522's **$260**
PTC2 Proto/Blank[TM]-Prototype card that fits MCP2 ..... **$49**
PTC2A Proto/Pop[TM]-w/regulator, decoders, switches **$99**

From Optimal Technology

ADC1 A/D: 8 channels; D/A: 2 channels. Requires ±12v to ±15 volts @ 100 ma and 2 I/0 ports from user 6522 ..... **$115.00**

Miscellaneous

TPT2 Approved Thermal Paper Tape 5/165 rolls **$10**
MEM6 6/2114 RAM Chips **$45**

CLOSE-OUT!

From Beta Computer

MEB3 32K Dynamic Memory Card w/on bd DC to DC converters (5V only .8A max) **$399**

SYSTEMS

We specialize in assembled and tested systems made from the above items. Normally, the price will be the total of the items, plus $5 for shipping, insurance and handling. Please call or write for exact prices or if questions arise.

Higher quantities quoted upon request.
COD's accepted.
Add $5 for shipping, insurance, and handling.
Minnesota residents add 4% sales tax.

Mail Check or Money Order To:
EXCERT, INC.
Educational Computer Division
P.O. BOX 8600
WHITE BEAR LAKE, MN. 55110
612-426-4114

Hard Copy Graphics For the Kim

Keith Sproul
1368 Noah Road
North Brunswick, New Jersey 08902

There are many different video boards out these days, each with its own advantages and each designed for a different system. Some video boards are 'Byte Mapped', that is they display an ASCII character for every byte in memory. The other type of video board is a 'Bit Mapped' screen, this type displays one 'pixel', a dot on the television screen, for each bit in memory. Both of these types have advantages. The byte mapped screens display a character at a time and are good for high speed text applications and sometimes rough graphics. The bit mapped screens by definition give you higher resolution graphics. With a bit mapped video board you can do professional quality graphics. Characters are still possible on a bit mapped screen, but they have to be 'drawn'.

Micro Technology Unlimited of Manchester, New Hampshire makes a 'VISIBLE MEMORY' video board that is of the second type. This video board displays 200 rows of 40 bytes across. At 8 bits to the byte, this produces a graphic display of 320 by 200 pixels. That is fairly high resolution, even higher than 'High Res' on the APPLE. Besides being a good graphics board, this 'Visible Memory' doubles as 8K of memory when the graphics is not needed.

Hal Chamberlin, who is well known in the microcomputer industry and also works for M.T.U., wrote a 'V.M. Support Package' for this board. This package does everything from plotting points to drawing characters on the screen. This 'software character generator' enables the user to display text in any format, including sub-scripted and super-scripted characters. The user can also redefine his own characters or the entire character set. This enables the use of languages other than English to be displayed on the screen, a feature that is very rarely seen.

All of this is fine and pretty on the screen, but what if hard copy is needed? Plotters can be used to produce hard copy, but they are extremely expensive and are dedicated to plotting. Plotters also require extensive software to run properly. A different alternative is to use a DIABLO HyTerm II terminal. This terminal produces about the best print quality available and has built-in graphics capabilities. By using the graphics mode, the image from a bit mapped screen can be 'plotted' on the terminal, bit for bit, producing the exact image that was on the screen. The program at the end of this article is a 6502 assembler routine to print the entire contents of the 'Visible Memory' screen onto a DIABLO HyTerm II. The ideas involved can be adapted to other graphics boards as well,

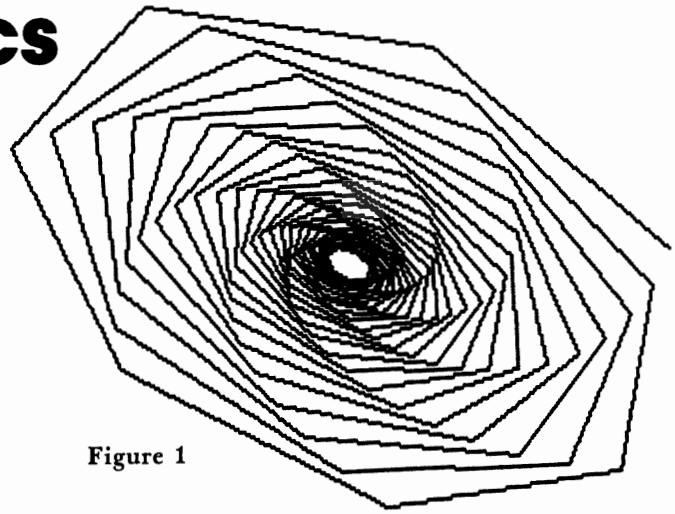


Figure 1

but will only work with 'bit mapped' video boards such as the M.T.U. 'Visible Memory'. Figure 1 was drawn on a M.T.U. 'Visible Memory' video board and then 'plotted' on a DIABLO using this program.

This program has been somewhat optimized for speed because of the length of time that is required for printing the images. Further optimization is possible such as adding reverse printing, and using other specialized features of the DIABLO. This was omitted to keep the program small and simple.

The screen is processed in four steps, the Page, the Line, the Byte, and the Bit. The program prints a Page by calling the Line subroutine 200 times. This routine prints a Line by calling the Byte routine 40 times, which in turn prints a byte by calling the Bit or Dot routine 8 times. The Line routine checks for the end of the line, only printing up to the last non-zero byte. It also skips any completely blank line, immediately going onto the next line to decrease printing time. The Byte routine shifts the byte to the left to determine if each bit is a 'one' or a 'zero'. The program can take up to an hour to print a complicated design. The average picture takes approximately 25 minutes. The reason for the big difference in time is that to print a 'one', a 'period' and a 'space' have to be printed, and to print a 'zero', just a space has to be printed, so a 'one' takes twice as long as a 'zero'.

This same set of ideas can be used to digitize pictures using a light sensitive diode fastened to the printing mechanism of the DIABLO. Instead of reading from the Visible Memory and writing to the plotter, the process is reversed, reading from the DIABLO and writing to the Visible Memory. This method was discussed in October 1979 issue of 'Dr. Dobb's Journal'. I hope to experiment with this idea in the future but have not had time to do it yet.

Of course this is not the cheapest way to go, but a DIABLO can be used for a lot more than just plotting. It can also be used as a normal hardcopy device for text processing or just listings whereas a plotter can only be used for one thing, plotting.

For reference only:
This is the list of escape sequences understood by the Diablo HyTerm II printer. The one(s) with '*' are used in this program.

<esc>	1		SET HORIZONTAL TAB
<esc>	2		CLEAR ALL TABS
<esc>	3	*	SETS GRAPHICS MODE
<esc>	4		CLEARS GRAPHICS MODE
<esc>	5		FORWARD PRINTING
<esc>	6		SETS REVERSE PRINTING
<esc>	7		not used
<esc>	8		CLEAR INDIVIDUAL TAB
<esc>	9		SET LEFT MARGIN
<esc>	0		SET RIGHT MARGIN
<esc>	A		PRINT IN RED
<esc>	B		PRINT IN BLACK
<esc>	D		NEGATIVE 1/2 LINE FEED
<esc>	U		1/2 LINE FEED
<esc>	<ht>(n)		ABSOLUTE HORIZONTAL TAB
<esc>	<vt>(n)		ABSOLUTE VERTICAL TAB
<esc>	<lf>		NEGATIVE LINE FEED

DIABLO Plot Routine

This program takes the image on an M.T.U. Visible Memory Board and plots that image on a DIABLO HyTerm II printer. The routine has some intelligence in that it checks for the end of each line and only plots up to the end of the line, going on with the next line when finished. If the entire line is blank, it skips that line and advances to the next line.

The program waits for a character to be typed at the keyboard of the Diablo so that you can straighten the paper before it starts printing, be sure to type a character that does not print on the terminal (i.e. a space).

NOTE: graphics mode (and some of the other modes) are cleared when the Diablo receives a (cr) (hex $0D), so graphics mode is reset at the beginning of every line.

```
 48          00E0                    LINADR  =       $00E0           ; ADDR OF CURRENT LINE
 49          00E2                    BCOUNT  =       $00E2           ; BYTE POINTER
 50          00E4                    ACC     =       $00E4
 51
 52          2000                    VM.ORG  =       $2000           ; Visible Memory Start Address
 55                                  SBTTL   Plot Program
 56
 57          0200                            =       $0200
 58
 59  0200    4C   09   02    START:  JMP     P.PAGE          ; TAILORING VECTORS
 60  0203    4C   A0   1E    OUTCHR: JMP     $1EA0           ; KIM PRINT CHAR ROUTINE (OR YOUR OWN)
 61  0206    4C   5A   1E    GETCHR: JMP     $1E5A           ; KIM GET CHAR ROUTINE (OR YOUR OWN)
 62
 63  0209                    P.PAGE:                         ; PLOT PAGE
 64  0209    20   06   02            JSR     GETCHR          ; WAIT FOR CHAR TO ALLOW SETTING UP OF PAPER
 65  020C    20   B8   02            JSR     CRLF            ; Put Print Wheel at LEFT Column
 66  020F    A2   08                 LDX     #8              ; 8 Lines down the Page (CHANGE IF DESIRED)
 67  0211    A9   0A         LINFD:  LDA     #$0A            ; <lf>
 68  0213    20   A6   02            JSR     PRTCHR
 69  0216    CA                      DEX
 70  0217    D0   F8                 BNE     LINFD
 71
 72  0219    A9   00                 LDA     #VM.ORG\
 73  021B    85   E0                 STA     LINADR          ; INIT LINADR to VM.ORG
 74  021D    A9   20                 LDA     #VM.ORG^
 75  021F    85   E1                 STA     LINADR+1
 76
 77  0221    20   6F   02            JSR     INIPLT          ; INIT Printer for PLOTTING
 78  0224    A2   C8                 LDX     #200            ; 200 LINES/PAGE
 79
 80  0226                    PLOT:                           ; PLOT PAGE
 81  0226    A0   27                 LDY     #39             ; INIT INDEX Pointer
 82  0228    B1   E0         CHKBLK: LDA     (LINADR),Y      ; TEST FOR END OF LINE
 83  022A    D0   05                 BNE     EOL
 84  022C    88                      DEY
 85  022D    D0   F9                 BNE     CHKBLK          ; CHECK NEXT BYTE
 86  022F    F0   06                 BEQ     NXTLIN          ; Advance to next LINE
 87                                                          ; IF ALL 40 BYTES = $00
 88  0231    C8              EOL:    INY                     ; MAKE Y = NUMBER OF BYTES TO PRINT
 89  0232    84   E2                 STY     BCOUNT          ; SAVE # OF BYTES IN LINE (THAT AREN'T ZERO)
 90  0234    20   7A   02            JSR     P.LINE          ; PLOT the LINE at (LINADR)
 91
 92  0237    18              NXTLIN: CLC
 93  0238    A5   E0                 LDA     LINADR
 94  023A    69   28                 ADC     #40             ; 40 BYTES/LINE
 95  023C    85   E0                 STA     LINADR
 96  023E    A5   E1                 LDA     LINADR+1
 97  0240    69   00                 ADC     #0
 98  0242    85   E1                 STA     LINADR+1
 99  0244    CA                      DEX
100  0245    F0   1A                 BEQ     EXIT
101                                                         ; End of Line
102  0247    A9   0D                 LDA     #$0D            ; <cr>
103  0249    20   A6   02            JSR     PRTCHR
```

```
104  024C  A9  00        LDA   #$00          ; <nul>
105  024E  20  A6  02    JSR   PRTCHR
106  0251  A9  00        LDA   #$00          ; <nul>
107  0253  20  A6  02    JSR   PRTCHR
108  0256  20  6F  02    JSR   INIPLT        ; INITIALIZE TO GRAPHICS MODE
109  0259  A9  0A        LDA   #$0A          ; <lf>
110  025B  20  A6  02    JSR   PRTCHR        ; Advance Paper 1/48 IN.
111  025E  4C  26  02    JMP   PLOT          ; Go BACK & Do NEXT LINE
112
115  0261  A9  0D   EXIT: LDA   #$0D          ; <cr>
116  0263  20  A6  02    JSR   PRTCHR        ; Clear GRAPHICS Mode
117  0266  A9  0C        LDA   #$0C          ; <ff>
118  0268  20  A6  02    JSR   PRTCHR        ; Advance to top of next page
119  026B  60            RTS                 ; IF USED AS A SUBROUTINE
120  026C  4C  4F  1C    JMP   $1C4F         ; OTHERWISE EXIT TO SYSTEM MONITOR
121                                          ; (KIM WARM START)
122
124                 .SBTTL  Plot Subroutines
127
128                                          ; INIT the DIABLO to PLOT MODE
129  026F  A9  1B   INIPLT: LDA   #$1B          ; <esc> 3
130  0271  20  A6  02    JSR   PRTCHR        ; Escape Sequence for PLOT MODE
131  0274  A9  33        LDA   #'3
132  0276  20  A6  02    JSR   PRTCHR
133  0279  60            RTS
134
135
136  027A                P.LINE:                 ; PLOT 1 LINE (320 BITS or 40 BYTES)
137                                          ; (or the # of BYTES in BCOUNT)
138  027A  A0  00        LDY   #0            ; INIT Index Pointer
139  027C  98   PLINE1:  TYA                 ; SAVE Y
140  027D  48            PHA
141  027E  B1  E0        LDA   (LINADR),Y
142  0280  20  8B  02    JSR   P.BYTE        ; PLOT the BYTE at (LINADR),Y
143  0283  68            PLA                 ; RESTORE Y
144  0284  A8            TAY
145  0285  C8            INY                 ; Advance Pointer to next BYTE
146  0286  C6  E2        DEC   BCOUNT
147  0288  D0  F2        BNE   PLINE1
148  028A  60            RTS                 ; Return when finished
149
151
152  028B                P.BYTE:                 ; PLOT 1 BYTE
153  028B  48            PHA                 ; SAVE ACC
154  028C  A0  08        LDY   #8            ; 8 BITS/BYTE
155  028E  68   PBYTE1:  PLA                 ; RESTORE ACC
156  028F  0A            ASL   A             ; SHIFT BIT into CARRY
157  0290  48            PHA                 ; RE-SAVE ACC
158  0291  20  99  02    JSR   P.BIT         ; PLOT THE BIT
159  0294  88            DEY                 ; KEEP TRACK OF BITS DONE
160  0295  D0  F7        BNE   PBYTE1        ; Do all 8 BITS
161  0297  68            PLA                 ; RESTORE STACK
162  0298  60            RTS
163
164
165  0299  90  05   P.BIT:  BCC   PBLANK        ; IF BIT = 0
166  029B  A9  2E        LDA   #'.
167  029D  20  A6  02    JSR   PRTCHR        ; IF BIT = 1
168  02A0  20  A4  02   PBLANK: JSR   OUTSPA        ; Advance Print wheel 1/60 IN.
169  02A3  60            RTS
170
173                  SBTTL  System Subroutines
174
175  02A4  A9  20   OUTSPA: LDA   #'            ; PRINT A <space>
176                                          ; FALL THROUGH TO PRTCHR
177
178                                          ; PRINT ASCII CHAR SAVING A, X, & Y
179  02A6  85  E4   PRTCHR: STA   ACC           ; SAVE ACC
180  02A8  48            PHA
181  02A9  8A            TXA                 ; SAVE X
182  02AA  48            PHA
183  02AB  98            TYA                 ; SAVE Y
184  02AC  48            PHA
185  02AD  A5  E4        LDA   ACC
186  02AF  20  03  02    JSR   OUTCHR        ; KIM OUTPUT ROUTINE (OR SYSTEM OUTPUT)
187  02B2  68            PLA
188  02B3  A8            TAY                 ; RESTORE Y
189  02B4  68            PLA
190  02B5  AA            TAX                 ; RESTORE X
191  02B6  68            PLA                 ; RESTORE ACC
192  02B7  60            RTS
193
```

```
195  02B8  48          CRLF:  PHA              ; SAVE ACC
196  02B9  8A                 TXA              ; SAVE X
197  02BA  48                 PHA
198  02BB  A9  0D             LDA   #$0D       ; SUBROUTINE TO
199  02BD  20  03  02         JSR   OUTCHR     ; PRINT <cr>,<lf>
200  02C0  A9  0A             LDA   #$0A
201  02C2  20  A6  02         JSR   PRTCHR
202  02C5  A2  04             LDX   #4         ; OUTPUT 4
203  02C7  A9  00       NULL: LDA   #$00       ; <nul> ($00)
204  02C9  20  03  02         JSR   OUTCHR
205  02CC  CA                 DEX
206  02CD  D0  F8             BNE   NULL
207  02CF  68                 PLA              ; RESTORE X
208  02D0  AA                 TAX
209  02D1  68                 PLA              ; RESTORE ACC
210  02D2  60                 RTS              ; RTS X = X  Y = Y  A = A
213        0000               . END
```


24 Hour Clock for SYM-1 BASIC

A. M. Mackay
600 Sixth Avenue West,
Owen Sound, Ontario, Canada
N4K 5E7

Load this program in your SYM-1 and enter G 0FB8. It will start a clock, display the memory you should enter to protect the program, then automatically transfer you to BASIC. Be sure to enter the amount of memory shown on the CRT. If you don't, you will lose the program.

The clock sits at the very top of your memory, and will not interfere with your BASIC - in fact, you won't even know it's there unless you call it. But it will be there when you want it for control operations, time delays for games, or whatever. You can even use your CRT as a time-of-day clock. It will keep ticking away until you hit "reset" or turn your SYM off.

As written, it is a 24 hour clock. If you want a 12 hour clock, change the contents of location 0FAD to "C0".

This clock uses timer 1 of U29 to interrupt the program every 50 ms. "COUNT" totals these interrupts, and after each 20 interrupts one is added to "SECS". Timer 2 and the input ports of U29 are not affected, and can be used for other purposes.

The program is written for a SYM-1 with 4K memory. It can be relocated upwards by changing all the "0F" bytes to "1F" or whatever. Also, change the bytes in locations 0FF9 to 0FFC. For example, for 8K, change these bytes to 37 35 34 38. The formula is 131 less than the number of free bytes usually displayed after going to BASIC, with a "3" before each digit.

The program as written keeps the time, starting at 0h, 0m, 0s from the time you enter G 0FB8. If you prefer time of day, enter the following (for 4K):

POKE &"0F7D", hour (in 24 hour time), CR
POKE &"0F7E", minute, CR
POKE &"0F7F", second, then hit CR on the exact second.

To use the clock in a BASIC program, enter the following BASIC command:

```
10 H = PEEK(&"0F7D"):M = PEEK(&"0F7E")
:S = PEEK(&"0F7F")
```

Then use the variables H, M and S as you require them. To use your CRT as a clock, use the following program:

```
10 - as in the paragraph above.
20 PRINT CHR$(12)
30 PRINT H; ":";M;":";S
40 FOR X = 1 TO 785: NEXT
50 GOTO 10
60 END
```

If you can turn cursor off, the clock will look much better.

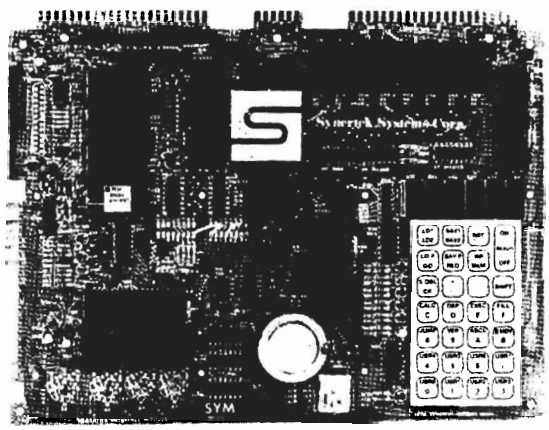
The actual clock program ends at 0FDE. The rest of the program puts out the memory requirements on the CRT. If you don't want to transfer to BASIC, or if you don't have a CRT, put 4C 00 80 in locations 0FDF - 0FE1. You can then access the time by going to M 0F7D (hour), 0F7E (mins) and 0F7F (secs).

If anyone wants it, perhaps I'll write a routine for a future issue that will display the time on the SYM-1 LED readouts.

```
0010             .BA $0F7C    ;THIS PROGRAM STARTS A
0020 ACCESS      .DE $8B86    ;24 HOUR CLOCK AND
0030 OUTVEC      .DE $A663    ;DISPLAYS THE AMOUNT OF
0040 IRQVEC      .DE $A67E    ;REMAINING MEMORY TO BE
0050 CLRINT      .DE $AC04    ;ENTERED WHEN REQUESTED
0060 T1CH        .DE $AC05    ;THEN AUTOMATICALLY
0070 T1LL        .DE $AC06    ;TRANFERS TO BASIC.
```

SYM-1, 6502-BASED MICROCOMPUTER

- FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to-use
- ALL LSI IC'S ARE IN SOCKETS
- 28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS
- EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY
- KIM-1* HARDWARE COMPATIBILITY
  The powerful 6502 8-Bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.
- THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user, expandable to five on-board.
- 4K BYTE ROM RESIDENT MONITOR and Operating Programs.
- Single 5 Volt power supply is all that is required.
- 1K BYTES OF 2114 STATIC RAM onboard with sockets provided for immediate expansion to 4K bytes onboard, with total memory expansion to 65, 536 bytes.
- USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs
- ENHANCED SOFTWARE with simplified user interface
- STANDARD INTERFACES INCLUDE:
  —Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1* compatible, Hi-Speed 1500 Baud)
  —Full duplex 20mA Teletype Interface
  —System Expansion Bus Interface
  —TV Controller Board Interface
  —CRT Compatible Interface (RS-232)
- APPLICATION PORT: 15 Bi-directional TTL Lines for user applications with expansion capability for added lines
- EXPANSION PORT FOR ADD-ON MODULES (51 I/O Lines included in the basic system)
- SEPARATE POWER SUPPLY connector for easy disconnect of the d-c power
- AUDIBLE RESPONSE KEYPAD



Synertek has enhanced KIM-1* software as well as the hardware. The software has simplified the user interface. The basic SYM-1 system is programmed in machine language. Monitor status is easily accessible, and the monitor gives the keypad user the same full functional capability of the TTY user. The SYM-1 has everything the KIM-1* has to offer, plus so much more that we cannot begin to tell you here. So, if you want to know more, the SYM-1 User Manual is available, separately.

SYM-1 Complete w/manuals $229.00
SYM-1 User Manual Only 7.00
SYM-1 Expansion Kit 60.00

Expansion includes 3K of 2114 RAM chips and 1-6522 I/O chip.

SYM-1 Manuals: The well organized documentation package is complete and easy-to-understand.

SYM-1 CAN GROW AS YOU GROW. It's the system to BUILD-ON. Expansion features that are available:

BAS-1 8K Basic ROM (Microsoft) $ 89.00
KTM-2 (Complete terminal less monitor) 319.00

QUALITY EXPANSION BOARDS DESIGNED SPECIFICALLY FOR KIM-1, SYM-1 & AIM 65

These boards are set up for use with a regulated power supply such as the one below, but, provisions have been made so that you can add onboard regulators for use with an unregulated power supply. But, because of unreliability, we do not recommend the use of onboard regulators. All I.C.'s are socketed for ease of maintenance. *All boards carry full 90-day warranty.*

All products that we manufacture are designed to meet or exceed industrial standards. All components are first qualtiy and meet full manufacturer's specifications. All this and an extended burn-in is done to reduce the normal percentage of field failures by up to 75%. To you, this means the chance of inconvenience and lost time due to a failure is very rare; but, if it should happen, we guarantee a turn-around time of less than forty-eight hours for repair.

Our money back guarantee: If, for any reason you wish to return any board that you have purchased directly from us within ten (10) days after receipt, complete, in original condition, and in original shipping carton; we will give you a complete credit or refund less a $10.00 restocking charge per board.

VAK-1 8-SLOT MOTHERBOARD

This motherboard uses the KIM-4* bus structure. It provides eight (8) expansion board sockets with rigid card cage. Separate jacks for audio cassette, TTY and power supply are provided. Fully buffered bus.

VAK-1 Motherboard $129.00

VAK-2/4 16K STATIC RAM BOARD

This board using 2114 RAMs is configured in two (2) separately addressable 8K blocks with individual write-protect switches.

VAK-2 16K RAM Board with only 8K of RAM (½ populated) $239.00
VAK-3 Complete set of chips to expand above board to 16K 125.00
VAK-4 Fully populated 16K RAM 325.00

VAK-5 2708 EPROM PROGRAMMER

This board requires a +5 VDC and ±12 VDC, but has a DC to DC multiplyer so there is no need for an additional power supply. All software is resident in on-board ROM, and has a zero-insertion socket.

VAK-5 EPROM Programmer w/2708 adapter $249.00
VAK-5A Single voltage 2716 adapter 45.00

VAK-6 EPROM BOARD

This board will hold 8K of 2708 or 2758, or 16K of 2716 or 2516 EPROMs. *EPROMs not included.*

VAK-6 EPROM Board $119.00

VAK-7 COMPLETE FLOPPY-DISK SYSTEM (Oct '79)

VAK-8 PROTYPING BOARD

This board allows you to create your own interfaces to plug into the motherboard. Etched circuitry is provided for regulators, address and data bus drivers; with a large area for either wire-wrapped or soldered IC circuitry.

VAK-8 Protyping Board $39.00

POWER SUPPLIES

ALL POWER SUPPLIES are totally enclosed with grounded enclosures for safety, AC power cord, and carry a full 2-year warranty.

FULL SYSTEM POWER SUPPLIES

This power supply will handle a microcomputer and up to 65K of our VAK-4 RAM. ADDITIONAL FEATURES ARE: Over voltage Protection on 5 volts, fused, AC on/off switch. Equivalent to units selling for $225.00 or more.

Provides +5 VDC @ 10 Amps & ±12 VDC @ 1 Amp
VAK-EPS Power Supply $119.00
VAK-EPS/AIM provides the same as VAK-EPS plus 24V unreg. 149.00

KIM-1* Custom P.S. provides 5 VDC @ 1.2 Amps and +12 VDC @ .1 Amps
KCP-1 Power Supply $39.00

SYM-1 Custom P.S. provides 5 VDC @ 1.4 Amps
VCP-1 Power Supply $39.00

*KIM is a product of MOS Technology

RNB ENTERPRISES INCORPORATED

2967 W. Fairmount Avenue
Phoenix AZ. 85017
(602)265-7564

master charge VISA

Add $2.50 for shipping and handling per order.

```
                    0080 ACR        .DE $AC0B    ;TIME IS NOT DISPLAYED UNTIL
                    0090 IFR        .DE $AC0D    ;REQUESTED BY BASIC PROGRAM.
                    0100 IER        .DE $AC0E
                    0110 BASIC      .DE $C000
0F7C- 14            0120 COUNT      .BY $14      ;SET COUNT TO 20
0F7D- 00            0130 HOUR       .BY $00      ;START TIME AT 00 HOURS
0F7E- 00            0140 MINS       .BY $00      ;               00 MINUTES
0F7F- 00            0150 SECS       .BY $00      ;               00 SECONDS
                    0170 ;***INTERRUPT ROUTINE***
0F80- 48            0190 CLOCK      PHA
0F81- CE 7C 0F      0200            DEC COUNT    ;SEE IF 1 SEC HAS PASSED
0F84- D0 2D         0210            BNE EXIT     ;IF NO, EXIT
0F86- A9 14         0220            LDA #20      ;IF YES, -
0F88- 8D 7C 0F      0230            STA COUNT    ; RESTORE COUNT AND
0F8B- EE 7F 0F      0240            INC SECS     ; ADD 1 TO SECS.
0F8E- AD 7F 0F      0250            LDA SECS     ;SEE IF 60 SECS HAS PASSED
0F91- 38            0260            SEC
0F92- E9 3C         0270            SBC #60
0F94- D0 1D         0280            BNE EXIT     ;IF NO, EXIT
0F96- 8D 7F 0F      0290            STA SECS     ;IF YES RESET SECS TO 0 AND
0F99- EE 7E 0F      0300            INC MINS     ;ADD 1 TO MINS
0F9C- AD 7E 0F      0310            LDA MINS     ;SEE IF 60 MINS HAS PASSED
0F9F- E9 3C         0320            SBC #60
0FA1- D0 10         0330            BNE EXIT     ;IF NO, EXIT
0FA3- 8D 7E 0F      0340            STA MINS     ;IF YES RESET MINS TO 0 AND
0FA6- EE 7D 0F      0350            INC HOUR     ;ADD 1 TO HOUR
0FA9- AD 7D 0F      0360            LDA HOUR     ;SEE IF 24 HOURS HAS PASSED
0FAC- E9 18         0370            SBC #24      ;IF NO,
0FAE- D0 03         0380            BNE EXIT     ;   EXIT
0FB0- 8D 7D 0F      0390            STA HOUR     ;IF YES, RESET HOUR TO 0
0FB3- AD 04 AC      0400 EXIT       LDA CLRINT   ;ENABLE TIMER INTERRUPT
0FB6- 68            0410            PLA
0FB7- 40            0420            RTI
                    0440 ;***INITIATE TIMER***
0FB8- 20 86 8B      0460 START      JSR ACCESS   ;DISABLE WRITE PROTECT SYS. RAM
0FBB- A9 80         0470            LDA #$80     ;SET IRQ
0FBD- 8D 7E A6      0480            STA IRQVEC   ;  VECTOR
0FC0- A9 0F         0490            LDA #$0F     ;   TO
0FC2- 8D 7F A6      0500            STA IRQVEC+$1        ;  "CLOCK"
0FC5- A9 C0         0510            LDA #$C0     ;SET BITS 6 AND 7
0FC7- 8D 0B AC      0520            STA ACR      ;FOR FREE-RUNNING MODE
0FCA- 8D 0E AC      0530            STA IER      ;AND T1 INTERRUPT ENABLE
0FCD- AD 0D AC      0540            LDA IFR      ;CLEAR T1 FLAG BIT 6 BUT
0FD0- 29 BF         0550            AND #$BF     ;DON'T DISTURB OTHER
0FD2- 8D 0D AC      0560            STA IFR      ;IFR BITS
0FD5- A9 50         0570            LDA #$50     ;SET
0FD7- 8D 06 AC      0580            STA T1LL     ; TIMER
0FDA- A9 C3         0590            LDA #$C3     ;  FOR 1/20 SEC AND
0FDC- 8D 05 AC      0600            STA T1CH     ;   START TIMER
                    0620 ;***BASIC MEMORY OUT***
0FDF- A2 00         0640 DISPMEM    LDX #$00     ;START OF
0FE1- BD F2 0F      0650 OUTMEM     LDA TABLE,X  ; ROUTINE
0FE4- 20 63 A6      0660            JSR OUTVEC ;
0FE7- E8            0670            INX  ;         TO
0FE8- C9 04         0680            CMP #$04 ;
0FEA- F0 03         0690            BEQ GOBAS    ; DISPLAY
0FEC- 4C E1 0F      0700            JMP OUTMEM   ; REMAINING
0FEF- 4C 00 C0      0710 GOBAS      JMP BASIC    ; MEMORY
0FF2- 0D 0A 4D      0720 TABLE      .BY $0D $0A 'MEM. 3965' $0D $0A $04
0FF5- 45 4D 2E
0FF8- 20 33 39
0FFB- 36 35 0D
0FFE- 0A 04
                    0730            .EN  ;END OF PROGRAM
```

Songs in the Key of KIM

George W. Hawkins
200 Old Country Road
Melville, NY 11746

I was fascinated by Richard Martin's Four part Harmony (Cheap) program in issue 16 and decided to try coding some other songs. I have made no attempt to code to conserve memory so far. Note that I have 2K of additional memory on my KIM-1 starting at hex 0400. The tempo and note addresses for my four songs are:

Song	0007-000F
Jingle Bells	15 FF 01 97 03 8F 04 87 05
Deck the Halls	13 FF 01 15 03 23 04 31 05
Shangri-La (only 2 parts)	26 FF 01 FF 01 D5 02 D5 02
Love is Blue	26 FF 01 AD 03 BB 04 C9 05

Jingle Bells

```
200  5B 0F 00 01 36 0F 00 01 3C 0F 00 01 44 0F 00 01
210  5B 2F 00 01 5B 07 00 01 5B 07 00 01 5B 0F 00 01
220  36 0F 00 01 3C 0F 00 01 44 0F 00 01 51 2F 00 01
230  51 0F 00 01 51 0F 00 01 33 0F 00 01 36 0F 00 01
240  3C 0F 00 01 48 2F 00 01 2D 0F 00 01 28 0F 00 01
250  2D 0F 00 01 33 0F 00 01 3C 0F 00 01 36 2F 00 01
260  5B 0F 00 01 5B 0F 00 01 36 0F 00 01 3C 0F 00 01
270  44 0F 00 01 5B 2F 00 01 5B 0F 00 01 5B 0F 00 01
280  36 0F 00 01 3C 0F 00 01 44 0F 00 01 51 2F 00 01
290  51 0F 00 01 51 0F 00 01 33 0F 00 01 36 0F 00 01
2A0  3C 0F 00 01 2D 0F 00 01 2D 0F 00 01 2D 0F 00 01
2B0  2D 0F 00 01 28 0F 00 01 2D 0F 00 01 33 0F 00 01
2C0  3C 0F 00 01 44 3F 00 01 36 0F 00 01 36 0F 00 01
2D0  36 1F 00 01 36 0F 00 01 36 0F 00 01 36 1F 00 01
2E0  36 0F 00 01 2D 0F 00 01 44 0F 00 01 3C 0F 00 01
2F0  36 3F 00 01 33 0F 00 01 33 0F 00 01 33 17 00 01

300  33 07 00 01 33 0F 00 01 36 0F 00 01 36 0F 00 01
310  36 07 00 01 36 07 00 01 36 0F 00 01 3C 0F 00 01
320  3C 0F 00 01 36 0F 00 01 3C 20 2D 1F 00 01 36 0F
330  00 01 36 0F 00 01 36 1F 00 01 36 0F 00 01 36 0F
340  00 01 36 1F 00 01 36 0F 00 01 2D 0F 00 01 44 0F
350  00 01 3C 0F 00 01 36 3F 00 01 33 0F 00 01 33 0F
360  00 01 33 17 00 01 33 07 00 01 33 0F 00 01 36 0F
370  00 01 36 0F 00 01 36 07 00 01 36 07 00 01 2D 0F
380  00 01 2D 0F 00 01 33 0F 00 01 3C 0F 00 01 44 30
390  00 40 00 40 00 C0 FF 01 00 10 36 10 00 10 36 10
3A0  00 10 36 10 00 10 36 10 00 10 36 10 00 10 36 10
3B0  00 10 33 10 00 10 33 10 00 10 33 10 00 10 33 10
3C0  00 10 33 10 00 10 33 10 00 10 33 10 00 10 33 10
3D0  00 10 36 10 00 30 36 10 00 10 36 10 00 10 36 10
3E0  00 10 36 10 00 10 36 10 00 10 36 10 00 10 33 10
3F0  00 10 33 10 00 10 33 10 00 10 33 10 00 10 33 10

400  00 10 33 0F 00 01 33 40 00 10 36 10 00 30 2D 10
410  00 10 28 10 00 10 24 10 00 10 28 10 00 10 2D 10
420  00 10 28 10 00 10 24 10 00 10 28 10 00 10 28 10
430  00 10 2D 10 00 10 2D 10 00 10 28 10 00 10 28 10
440  00 10 28 10 00 10 28 0F 00 01 33 10 00 20 2D 10
450  00 10 28 10 00 10 24 10 00 10 28 10 00 10 2D 10
460  00 10 28 10 00 10 24 10 00 10 28 10 00 10 28 10
470  00 10 2D 10 00 10 2D 10 00 10 30 0F 00 01 28 1F
480  00 01 2D 1F 00 01 2D 30 00 40 00 40 00 C0 97 03
490  00 10 44 10 00 10 44 10 00 10 44 10 00 10 44 10
4A0  00 10 44 10 00 10 44 10 00 10 44 10 00 10 44 10
4B0  00 10 3C 10 00 10 3C 10 00 10 51 10 00 10 51 10
4C0  00 10 51 10 00 10 51 10 00 10 44 10 00 30 44 10
4D0  00 10 44 10 00 10 44 10 00 10 44 10 00 10 44 10
4E0  00 10 44 10 00 10 44 10 00 10 44 10 00 10 3C 10
4F0  00 10 3C 10 00 10 51 10 00 10 51 0F 00 01 5B 40

500  00 10 44 10 00 30 36 10 00 10 36 10 00 10 36 10
510  00 10 36 10 00 10 36 10 00 10 36 10 00 10 36 10
520  00 10 36 10 00 10 33 10 00 10 33 10 00 10 36 10
530  00 10 36 10 00 10 36 10 00 10 30 10 00 10 33 0F
540  00 01 2D 10 00 20 36 10 00 10 36 10 00 10 36 10
550  00 10 36 10 00 10 36 10 00 10 36 10 00 10 36 10
560  00 10 36 10 00 10 33 10 00 10 33 10 00 10 36 10
570  00 10 36 0F 00 01 33 1F 00 01 33 1F 00 01 36 30
580  00 40 00 40 00 C0 8F 04 44 0F 00 01 5B 0F 00 01
590  5B 0F 00 01 5B 0F 00 01 44 0F 00 01 5B 0F 00 01
5A0  5B 0F 00 01 5B 0F 00 01 44 0F 00 01 5B 0F 00 01
5B0  44 0F 00 01 5B 0F 00 01 66 0F 00 01 51 0F 00 01
5C0  66 0F 00 01 51 0F 00 01 79 0F 00 01 51 0F 00 01
5D0  79 0F 00 01 51 0F 00 01 5B 0F 00 01 5B 0F 00 01
5E0  5B 0F 00 01 5B 0F 00 01 5B 0F 00 01 5B 0F 00 01
5F0  5B 0F 00 01 5B 0F 00 01 44 0F 00 01 5B 0F 00 01

600  44 10 00 10 44 0F 00 01 5B 0F 00 01 5B 0F 00 01
610  5B 0F 00 01 44 0F 00 01 5B 0F 00 01 5B 0F 00 01
620  5B 0F 00 01 44 0F 00 01 5B 0F 00 01 44 0F 00 01
630  5B 0F 00 01 66 0F 00 01 51 0F 00 01 66 0F 00 01
640  51 0F 00 01 79 0F 00 01 51 0F 00 01 79 0F 00 01
650  51 0F 00 01 5B 0F 00 01 5B 0F 00 01 5B 0F 00 01
660  5B 0F 00 01 51 1F 00 01 48 1F 00 01 44 0F 00 01
670  5B 0F 00 01 44 10 00 10 44 0F 00 01 44 0F 00 01
680  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
690  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
6A0  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
6B0  44 0F 00 01 44 0F 00 01 79 0F 00 01 44 0F 00 01
6C0  5B 0F 00 01 48 0F 00 01 44 0F 00 01 44 0F 00 01
6D0  51 0F 00 01 44 0F 00 01 79 0F 00 01 44 0F 00 01
6E0  79 0F 00 01 40 0F 00 01 5B 0F 00 01 44 0F 00 01
6F0  48 10 00 10 44 0F 00 01 44 0F 00 01 44 0F 00 01

700  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
710  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
720  44 0F 00 01 44 0F 00 01 44 0F 00 01 44 0F 00 01
730  44 0F 00 01 79 0F 00 01 44 0F 00 01 5B 0F 00 01
740  48 0F 00 01 44 0F 00 01 44 0F 00 01 51 0F 00 01
750  40 0F 00 01 44 1F 00 01 48 1F 00 01 44 30 00 40
760  00 40 00 C0 87 05
```

Shangri-La

```
200  00 10 51 10 4C 10 44 10 51 10 5B 30 00 10 51 10
210  4C 10 44 10 3C 40 00 10 51 10 4C 10 44 10 51 10
220  5B 1F 00 01 4C 10 3C 10 39 10 44 10 3C 10 39 10
230  33 10 30 18 33 08 33 40 33 0F 00 01 33 0F 00 01
240  33 0F 00 01 33 0F 00 01 33 10 2B 1F 00 01 33 0F
250  00 01 33 10 2B 1F 00 01 33 10 39 10 30 20 39 10
260  3C 2F 00 01 3C 0F 00 01 3C 10 33 1F 00 01 3C 0F
270  00 01 3C 10 33 1F 00 01 39 10 44 30 5B 10 3C 40
280  00 10 51 10 4C 10 44 10 51 10 5B 30 00 10 51 10
290  4C 10 44 10 3C 40 00 10 51 10 4C 10 44 10 51 10
2A0  5B 1F 00 01 4C 10 3C 10 39 10 44 10 3C 10 39 10
2B0  33 10 30 18 33 08 33 40 33 2F 00 01 5B 10 3C 10
2C0  39 10 44 10 3C 0F 00 01 39 10 33 10 30 10 2B 10
2D0  33 40 00 C0 FF 01 79 3F 00 01 66 2F 00 01 79 10
2E0  72 3F 00 01 4C 1F 00 01 51 1F 00 01 79 3F 00 01
2F0  66 3F 00 01 5B 1F 00 01 4C 1F 00 01 51 1F 00 01

300  40 1F 00 01 3C 40 3C 0F 00 01 3C 0F 00 01 3C 0F
310  00 01 3C 0F 00 01 39 3F 00 01 40 3F 00 01 40 3F
320  00 01 44 20 4C 1F 00 01 4C 3F 00 01 4C 3F 00 01
330  5B 2F 00 01 5B 0F 00 01 60 3F 00 01 79 3F 00 01
340  66 2F 00 01 79 10 72 3F 00 01 4C 1F 00 01 51 1F
350  00 01 79 3F 00 01 66 3F 00 01 5B 1F 00 01 4C 1F
360  00 01 51 1F 00 01 40 1F 00 01 44 40 51 30 00 10
370  5B 1F 00 01 4C 1F 00 01 51 1F 00 01 40 1F 00 01
380  3C 2F 00 01 3C 07 00 01 33 07 00 01 2D 0F 00 01
390  39 0F 00 01 48 0F 00 01 44 0F 00 81 D5 02
```

Love is Blue

```
200  36 1F 00 01 3C 17 00 01 44 07 00 01 48 0F 00 01
210  51 0F 00 01 5B 1F 00 01 51 0F 00 01 44 07 00 01
220  48 07 00 01 44 17 00 01 51 07 00 01 5B 07 00 01
230  51 07 00 01 5B 0F 00 01 6C 1F 00 01 36 1F 00 01
240  3C 17 00 01 44 07 00 01 48 0F 00 01 51 0F 00 01
250  5B 1F 00 01 51 0F 00 01 44 07 00 01 48 07 00 01
260  44 17 00 01 51 07 00 01 56 07 00 01 60 07 00 01
270  56 0F 00 01 51 20 00 40 36 1F 00 01 3C 17 00 01
280  44 07 00 01 48 0F 00 01 51 0F 00 01 5B 1F 00 01
290  51 0F 00 01 44 07 00 01 48 07 00 01 44 17 00 01
2A0  51 07 00 01 5B 07 00 01 51 07 00 01 5B 0F 00 01
2B0  6C 1F 00 01 36 1F 00 01 3C 17 00 01 44 07 00 01
2C0  48 0F 00 01 51 0F 00 01 5B 1F 00 01 51 0F 00 01
2D0  44 07 00 01 48 07 00 01 44 17 00 01 51 07 00 01
2E0  56 07 00 01 60 07 00 01 56 0F 00 01 51 20 00 40
2F0  40 1F 00 01 3C 1F 00 01 36 2F 00 01 40 07 00 01

300  51 07 00 01 51 1F 00 01 30 1F 00 01 36 3F 00 01
310  40 20 36 1F 00 01 48 2F 00 01 40 07 00 01 3C 07
320  00 01 3C 2F 00 01 48 07 00 01 56 07 00 01 51 30
330  00 10 36 1F 00 01 3C 17 00 01 44 07 00 01 48 0F
340  00 01 51 0F 00 01 5B 1F 00 01 51 0F 00 01 44 07
350  00 01 48 07 00 01 44 17 00 01 51 07 00 01 5B 07
360  00 01 51 07 00 01 5B 0F 00 01 6C 1F 00 01 36 1F
370  00 01 3C 17 00 01 44 07 00 01 48 0F 00 01 51 0F
380  00 01 5B 1F 00 01 51 0F 00 01 44 07 00 01 48 07
390  00 01 44 17 00 01 51 07 00 01 56 07 00 01 60 07
3A0  00 01 56 0F 00 01 51 20 51 20 00 A0 FF 01 00 10
3B0  36 0F 00 01 30 1F 00 01 2D 20 2D 1F 00 01 28 20
3C0  28 1F 00 01 2D 20 2D 1F 00 01 00 10 36 0F 00 01
3D0  30 1F 00 01 2D 20 2D 1F 00 01 28 20 28 1F 00 01
3E0  36 20 36 17 00 01 28 07 00 01 2B 07 00 01 30 07
3F0  00 01 2B 0F 00 01 28 10 00 10 00 10 36 0F 00 01

400  30 1F 00 01 2D 20 2D 1F 00 01 28 20 28 1F 00 01
410  2D 20 2D 20 00 10 36 0F 00 01 30 1F 00 01 2D 20
420  2D 1F 00 01 28 20 28 1F 00 01 36 20 36 17 00 01
430  28 07 00 01 2B 07 00 01 30 07 00 01 2B 0F 00 01
440  28 1F 00 01 51 1F 00 01 48 1F 00 01 40 2F 00 01
450  51 10 00 20 3C 1F 00 01 40 40 00 20 40 20 00 10
460  3C 10 30 10 24 10 00 10 28 0F 00 01 2B 20 00 10
470  40 10 36 10 28 10 00 10 36 0F 00 01 30 1F 00 01
480  2D 20 2D 1F 00 01 28 20 28 1F 00 01 2D 20 2D 1F
490  00 01 00 10 36 0F 00 01 30 1F 00 01 2D 20 2D 1F
4A0  00 01 28 20 28 1F 00 01 36 20 36 1F 00 01 40 0F
4B0  00 01 36 0F 00 01 36 1F 00 81 AD 03 00 10 44 0F
4C0  00 01 3C 20 3C 1F 00 01 36 1F 00 01 36 20 33 1F
4D0  00 01 3C 1F 00 01 36 20 00 10 44 0F 00 01 3C 20
4E0  3C 1F 00 01 36 1F 00 01 36 20 33 1F 00 01 3C 1F
4F0  00 01 44 17 00 01 00 08 3C 1F 00 01 44 10 00 10

500  00 10 44 0F 00 01 3C 20 3C 1F 00 01 36 1F 00 01
510  36 20 33 1F 00 01 3C 20 36 1F 00 01 00 10 44 0F
520  00 01 3C 20 3C 1F 00 01 36 1F 00 01 36 20 33 1F
530  00 01 3C 20 44 17 00 01 00 08 3C 1F 00 01 44 0F
540  00 01 36 0F 00 01 00 10 28 10 00 10 28 10 00 10
550  40 10 36 10 36 10 00 10 3C 10 3C 10 30 10 00 10
560  40 10 40 10 36 10 00 10 40 10 40 10 36 10 00 10
570  3C 10 3C 10 30 10 00 10 3C 10 3C 20 00 10 00 10
580  40 10 36 10 00 10 44 0F 00 01 3C 20 3C 1F 00 01
590  36 1F 00 01 36 20 33 1F 00 01 3C 1F 00 01 36 1F
5A0  00 01 00 10 44 0F 00 01 3C 20 3C 1F 00 01 36 1F
5B0  00 01 36 20 33 1F 00 01 3C 20 40 1F 00 01 51 1F
5C0  00 01 48 1F 00 81 BB 04 FF FF 51 1F 00 01 79 1F
5D0  00 01 5B 1F 00 01 44 1F 00 01 51 17 00 01 5B 07
5E0  00 01 66 1F 00 01 5B 1F 00 01 44 1F 00 01 51 1F
5F0  00 01 79 1F 00 01 5B 1F 00 01 44 1F 00 01 51 17

600  00 01 5B 07 00 01 66 1F 00 01 36 1F 00 01 51 1F
610  00 01 36 1F 00 01 51 1F 00 01 51 1F 00 01 79 1F
620  00 01 5B 1F 00 01 44 1F 00 01 51 17 00 01 5B 07
630  00 01 66 1F 00 01 5B 1F 00 01 44 1F 00 01 51 1F
640  00 01 79 1F 00 01 5B 1F 00 01 44 1F 00 01 51 17
650  00 01 5B 07 00 01 66 1F 00 01 36 1F 00 01 51 1F
660  00 01 36 1F 00 01 51 1F 00 01 51 1F 00 01 48 1F
670  00 01 40 2F 00 01 51 0F 00 01 3C 3F 00 01 51 3F
680  00 01 40 3F 00 01 3C 3F 00 01 36 2F 00 01 36 0F
690  00 01 51 3F 00 01 51 1F 00 01 79 1F 00 01 5B 1F
6A0  00 01 44 1F 00 01 51 17 00 01 5B 07 00 01 66 1F
6B0  00 01 5B 1F 00 01 44 1F 00 01 51 1F 00 01 79 1F
6C0  00 01 5B 1F 00 01 44 1F 00 01 51 17 00 01 5B 07
6D0  00 01 66 1F 00 01 36 1F 00 01 51 1F 00 01 51 1F
6E0  00 01 48 0F 00 01 36 0F 00 81 C9 05
```

Deck the Halls

```
200  33 17 00 01 39 07 00 01 3C 0F 00 01 44 0F 00 01
210  4C 0F 00 01 44 0F 00 01 3C 0F 00 01 4C 0F 00 01
220  44 07 00 01 3C 07 00 01 39 07 00 01 44 07 00 01
230  3C 17 00 01 44 07 00 01 4C 0F 00 01 51 0F 00 01
240  4C 1F 00 01 33 17 00 01 39 07 00 01 3C 0F 00 01
250  44 0F 00 01 4C 0F 00 01 44 0F 00 01 3C 0F 00 01
260  4C 0F 00 01 44 07 00 01 3C 07 00 01 39 07 00 01
270  44 07 00 01 3C 17 00 01 44 07 00 01 4C 0F 00 01
280  51 0F 00 01 4C 1F 00 01 44 17 00 01 3C 07 00 01
290  39 0F 00 01 44 0F 00 01 3C 17 00 01 39 07 00 01
2A0  33 0F 00 01 44 0F 00 01 3C 07 00 01 36 07 00 01
2B0  33 0F 00 01 2D 07 00 01 28 07 00 01 26 0F 00 01
2C0  28 0F 00 01 2D 0F 00 01 33 1F 00 01 33 17 00 01
2D0  39 07 00 01 3C 0F 00 01 44 0F 00 01 4C 0F 00 01
2E0  44 0F 00 01 3C 0F 00 01 4C 0F 00 01 2D 07 00 01
2F0  2D 07 00 01 2D 07 00 01 2D 07 00 01 33 17 00 01

300  39 07 00 01 3C 0F 00 01 44 0F 00 01 4C 20 00 40
310  00 40 00 C0 FF 01 3C 17 00 01 44 07 00 01 4C 0F
320  00 01 51 0F 00 01 5B 0F 00 01 51 0F 00 01 4C 0F
330  00 01 66 0F 00 01 51 07 00 01 4C 07 00 01 44 07
340  00 01 51 07 00 01 4C 17 00 01 5B 07 00 01 66 0F
350  00 01 66 0F 00 01 66 1F 00 01 3C 17 00 01 44 07
360  00 01 4C 0F 00 01 51 0F 00 01 5B 0F 00 01 51 0F
370  00 01 4C 0F 00 01 66 0F 00 01 51 07 00 01 4C 07
380  00 01 44 07 00 01 51 07 00 01 4C 17 00 01 5B 07
390  00 01 66 0F 00 01 66 0F 00 01 66 1F 00 01 51 17
3A0  00 01 4C 07 00 01 44 0F 00 01 51 0F 00 01 4C 17
3B0  00 01 4C 07 00 01 4C 0F 00 01 44 0F 00 01 4C 0F
3C0  00 01 4C 0F 00 01 3C 0F 00 01 3C 0F 00 01 44 0F
3D0  00 01 4C 0F 00 01 51 1F 00 01 3C 17 00 01 44 07
3E0  00 01 4C 0F 00 01 51 0F 00 01 5B 0F 00 01 51 0F
3F0  00 01 4C 0F 00 01 66 0F 00 01 4C 07 00 01 4C 07
400  00 01 4C 07 00 01 4C 07 00 01 4C 17 00 01 44 07
410  00 01 4C 0F 00 01 51 0F 00 01 4C 20 00 40 00 40
420  00 C0 15 03 33 17 00 01 33 07 00 01 33 0F 00 01
430  39 0F 00 01 3C 0F 00 01 33 0F 00 01 33 0F 00 01
440  3C 0F 00 01 33 07 00 01 33 07 00 01 33 07 00 01
450  33 07 00 01 33 17 00 01 39 07 00 01 3C 0F 00 01
460  44 0F 00 01 3C 1F 00 01 33 17 00 01 33 07 00 01
470  33 0F 00 01 39 0F 00 01 3C 0F 00 01 33 0F 00 01
480  33 0F 00 01 3C 0F 00 01 33 07 00 01 33 07 00 01
490  33 07 00 01 33 07 00 01 33 17 00 01 39 07 00 01
4A0  3C 0F 00 01 44 0F 00 01 3C 1F 00 01 33 17 00 01
4B0  33 07 00 01 33 0F 00 01 33 0F 00 01 33 17 00 01
4C0  44 07 00 01 3C 0F 00 01 33 0F 00 01 33 0F 00 01
4D0  33 0F 00 01 33 0F 00 01 33 0F 00 01 33 0F 00 01
4E0  36 0F 00 01 33 1F 00 01 33 17 00 01 33 07 00 01
4F0  33 0F 00 01 39 0F 00 01 3C 0F 00 01 33 0F 00 01

500  33 0F 00 01 3C 0F 00 01 39 07 00 01 39 07 00 01
510  39 07 00 01 39 07 00 01 33 17 00 01 2D 07 00 01
520  33 0F 00 01 39 0F 00 01 3C 20 00 40 00 40 00 C0
530  23 04 4C 17 00 01 51 07 00 01 4C 0F 00 01 66 0F
540  00 01 5B 0F 00 01 66 0F 00 01 4C 0F 00 01 4C 0F
550  00 01 66 07 00 01 66 07 00 01 66 07 00 01 66 07
560  00 01 4C 17 00 01 72 07 00 01 66 0F 00 01 66 0F
570  00 01 4C 1F 00 01 4C 17 00 01 51 07 00 01 4C 0F
580  00 01 66 0F 00 01 5B 0F 00 01 66 0F 00 01 4C 0F
590  00 01 4C 0F 00 01 66 07 00 01 66 07 00 01 66 07
5A0  00 01 66 07 00 01 4C 17 00 01 72 07 00 01 66 0F
5B0  00 01 66 0F 00 01 4C 1F 00 01 66 17 00 01 66 07
5C0  00 01 66 0F 00 01 66 0F 00 01 4C 17 00 01 4C 07
5D0  00 01 4C 0F 00 01 51 0F 00 01 4C 0F 00 01 3C 0F
5E0  00 01 4C 0F 00 01 5B 0F 00 01 44 0F 00 01 44 0F
5F0  00 01 66 1F 00 01 4C 17 00 01 51 07 00 01 4C 0F

600  00 01 66 07 00 01 60 07 00 01 5B 0F 00 01 66 0F
610  00 01 4C 0F 00 01 4C 0F 00 01 39 07 00 01 39 07
620  00 01 39 07 00 01 39 07 00 01 3C 17 00 01 39 07
630  00 01 33 0F 00 01 39 0F 00 01 4C 20 00 40 00 40
640  00 C0 31 05
```

REVIEW KIMEX-1

PROM, RAM and I/O Expansion for the KIM

Harvey B. Herman

Digital Engineering Associates $139.95
P. O. Box 207 Bethlehem, PA 18016

Those of us who have cut our computer baby teeth on the KIM have longed to have some of the capabilities of SYM (a newer, single-board computer) without, heaven forbid, having to throw out our first love. Digital Engineering Associates has come to our rescue with their product KIMEX-1. They are marketing a single-board add-on module which plugs into the KIM expansion interface and requires 6 wires to be soldered to the KIM application connector. The following features are standard:

1. Sockets for 4K of RAM (4118) contiguous with KIM's 1K RAM.
2. A 6522 VIA with I/O lines brought out to a
3. Sockets for four 2716 5VEPROMs which can be selectively vectored to on power up.

The last item is really neat as this should greatly simplify operation of applications programs in EPROM by users unfamiliar with KIM.

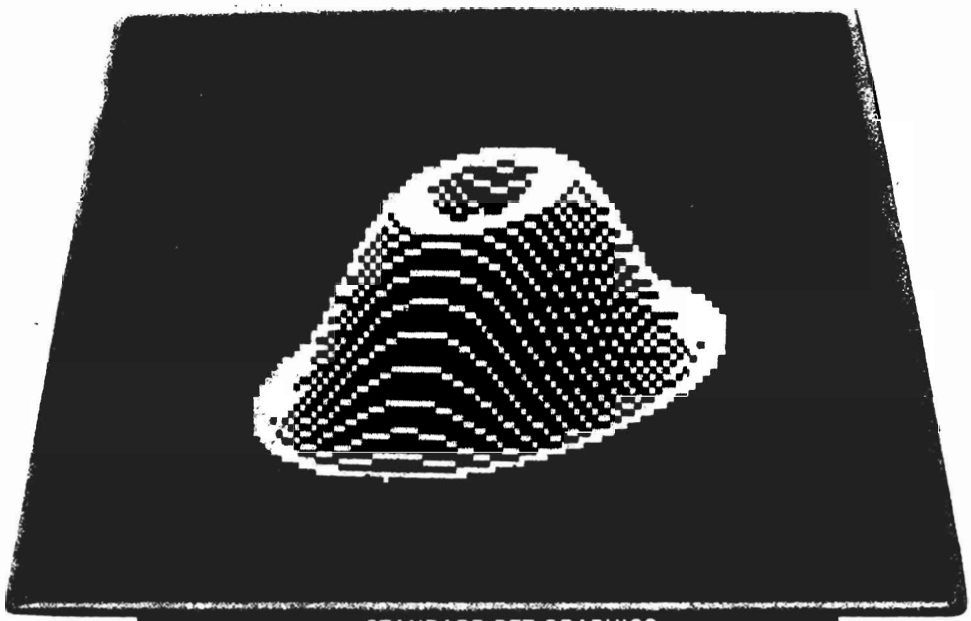
The module appears to my eye very well designed and professionally constructed. It was trivial to connect to a basic KIM (15 minutes or less). For review purposes only, the company provided a clock program on EPROM which is described as an example in their 19-page manual. I turned on power (an extra 300 mamp from the 5V supply is necessary) and I was into the clock program and counting. Their program makes use of the 6522 VIA on board (a data sheet on the 6522 is also included with the manual). I am only just beginning to appreciate the "versatility" of the VIA chip and missed having one on the original KIM. Now's my chance.

The negative points are minor. I believe it may be more difficult and/or expensive to obtain a MOSTEK 4118 (1K x 8) than a 2114 (1K x 4), for example. Furthermore, it might have been helpful in some systems to address the 4K of RAM anywhere in memory. Other than that, I think the module is a pretty good deal for KIM owners who need its features, and I recommend it to them. ©

Editor's Note: If this review seems familiar to you, you may have read it in Issue 3 of COMPUTE. The blank half page in that issue was supposed to be the company's ad. Hopefully it's in this issue. We're reprinting the review as a service to you and them. My apologies to Edward H. Carlson, author of ***Fast Tape Read/Write Programs for your OSI*** *(Issue 3, COMPUTE, p. 115). Here, in full, is Listing 3. Oh well... RCL*

```
10:       FAST KC TAPE READ
20;
30 LEADER =$0F    LEADER CHARACTER, $0F
40 SCREEN =$D100  LOCATION ON MONITOR SCREEN
50 ACIA   =$FC00  6850 ACIA TAPE PORT
60 START  =$00    HOLDS ADDRESS OF 1ST BYTE OF TEXT
70 END    =$02    HOLDS ADDRESS OF LAST BYTE OF TEXT
80 EXECUT =$04    CONTAINS ADDRESS OF PROGRAM START
90 CURENT =$06    HOLDS ADDRESS OF CURRENT TEXT BYTE
100 CHKSUM =$08   CHECK SUM FROM TAPE STORED HERE
110 COUNT  =$09   COMPUTED CHECK SUM AND OTHER STUFF
120       *=$C700
130       LDA #'N       READING NOISE BEFORE LEADER
140       STA SCREEN+2
150 MAIN  LDY #0        READ LEADER, $0F 0F 0F
160       STY COUNT
170 M1    JSR RT        READ TAPE BYTE
180       STA SCREEN
190       CMP #LEADER   IS IT A LEADER BYTE?
200       BNE MAIN      NO, READ ANOTHER BYTE
210       INC COUNT     YES, INCREMENT
220       LDA #'L       PRINT L FOR EVERY $0F READ
230       STA SCREEN+4,Y
240       INY
250       LDA #3        READ 3 OF THEM?
260       CMP COUNT
270       BNE M1        NOT YET, READ ANOTHER
280 ADDR  LDY #0        LEADER OVER.  READ START,
290       STY COUNT       END, EXECUTE ADDRESSES
300       LDA #'A
310       STA SCREEN+8
320 A1    JSR RT
330       STA START,Y
340       STA SCREEN
350       INY
360       CPY #6
370       BNE A1        BRANCH TO CONTINUE READING A
380       LDA START     SET INITIAL ADDRESS
390       STA CURENT+1
400       LDA START+1
410       STA CURENT
420 TEXT  LDY #0
430       STY COUNT     CLEAR FOR CALC. CHECK SUM
440       LDA #'T
450       STA SCREEN+10
460 RBT   JSR RT        READ A BYTE OF TEXT
470       STA (CURENT),Y
480       STA SCREEN
490       CLC
500       ADC COUNT     COUNT ACCUMULATES CHECK SUM
510       STA COUNT
520       LDA CURENT    TEST FOR END OF TEXT, LO
530       CMP END+1
540       BNE M3        NOT EQUAL, INC AND READ BYTE
550       LDA CURENT+1  LO EQUAL, TEST HI
560       CMP END
570       BEQ M6        BRANCH IF TEXT IS ALL READ
580 M3    INC CURENT    INCREMENT CURRENT ADDRESS
590       BNE M4
600       INC CURENT+1
610 M4    JMP RBT       GO READ NEXT BYTE
620 M6    JSR RT        READ CHECK SUM BYTE
630       STA CHKSUM
640       CMP COUNT     TEST CHECK SUM
650       BEQ GO        IF OK, BRANCH AND EXECUTE
660       LDA #'E       IF NOT, PRINT ERROR MESSAGE
670       STA SCREEN+12
680       JSR $CB48     BELL
690       BRK
700 GO    LDA EXECUT
710       STA CURENT+1
720       LDA EXECUT+1
730       STA CURENT
740       JSR $CB48     BELL
750       JMP (CURENT)  EXECUTE
760;
770;                    TAPE READ SUBROUTINE
780 RT    LDA ACIA      READ A BYTE FROM 6850
790       LSR A
800       BCC RT
810       LDA ACIA+1
820       RTS
```

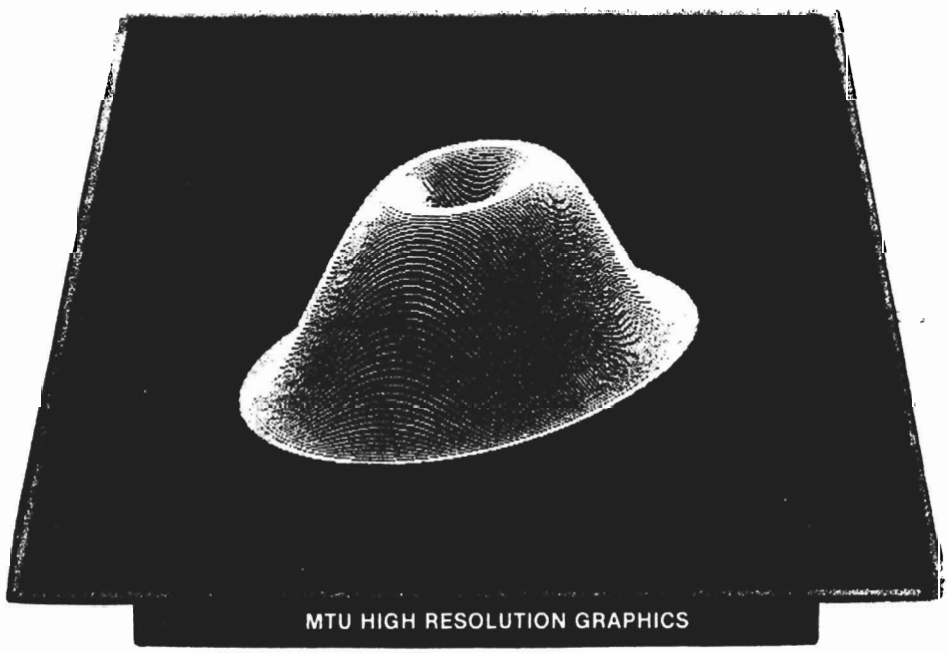
©



STANDARD PET GRAPHICS

HIGH RESOLUTION GRAPHICS

LOOK TO MTU. WE SUPPORT HIGH RESOLUTION GRAPHICS ON: PET — AIM — KIM — SYM



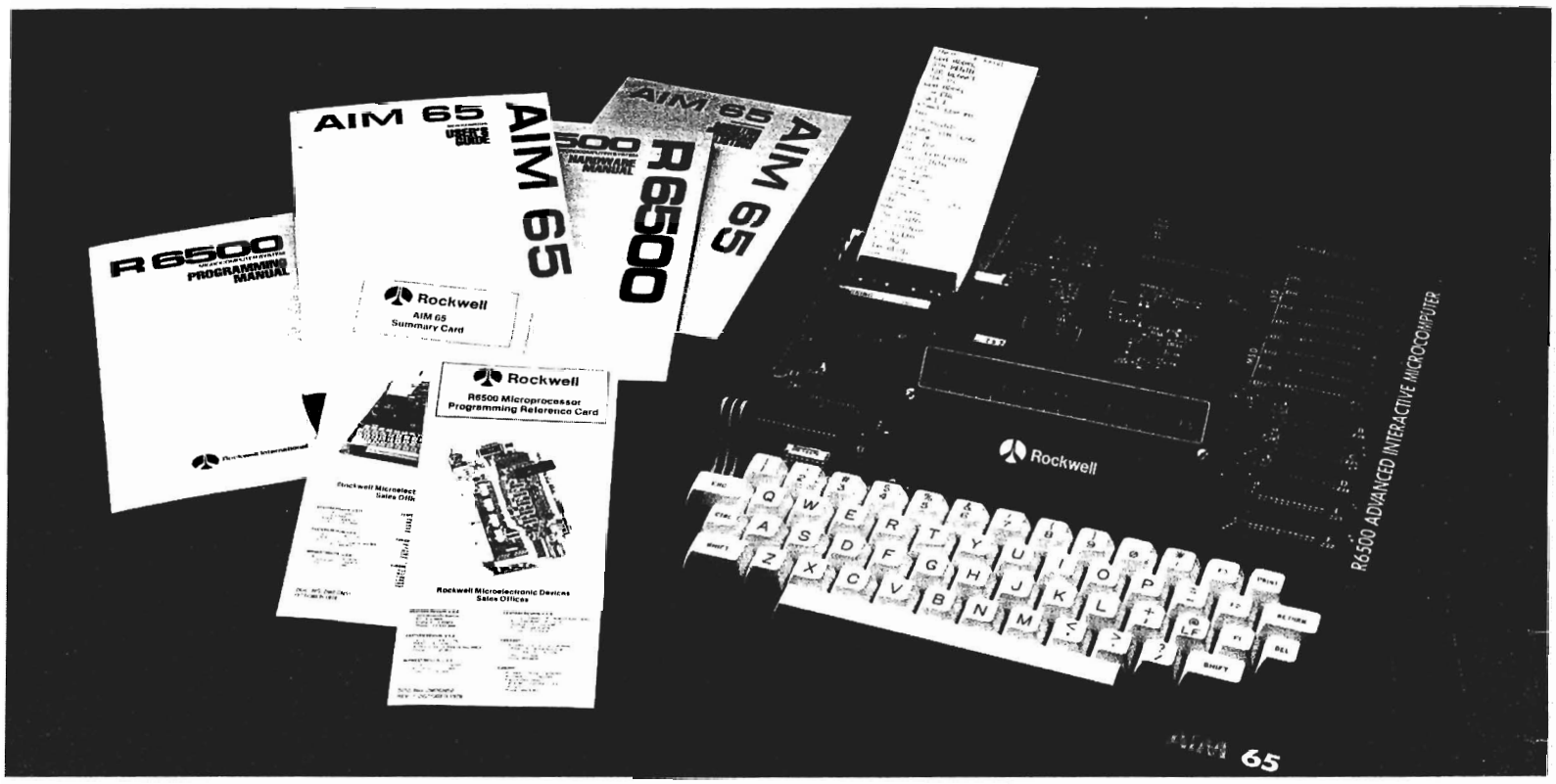
MTU HIGH RESOLUTION GRAPHICS

MTU

Micro Technology Unlimited
P.O. Box 4596, 841 Galaxy Way
Manchester, N.H. 03108
603-627-1464

Call Or Write For Our Full Line Catalog

AIM 65. The head start in educational microcomputers.



On-Board Printer, Advanced R6502 CPU, Versatile I/O — It's the Honors Candidate for Microprocessor Learning

Micro Power

It's tops in its class because it's expressly designed for microprocessor learning. Rockwell's AIM 65 is a fully-assembled microcomputer system with special educational features at a low price school budgets can afford.

AIM 65's on-board thermal printer — unique in its price range — produces hard copies of exercises for easy checking by student and instructor. On-board I/Os provide dual cassette, TTY and general purpose interfaces. Bus and system expansion is built in. Same for PROM, ROM and RAM expansion.

AIM 65's Interactive Monitor prompts students each step of the way in hands-on learning of microprocessor fundamentals. It includes a Text Editor, Interactive Mnemonic Assembler, Debugger (Trace, Breakpoints), and more!

An optional fully symbolic Assembler program makes AIM 65 a powerful hands-on learning system for microcomputer development and prototyping. Advanced students can explore high level languages with an optional ROM-resident BASIC Interpreter. There's even a college textbook available.

And you'll find AIM 65 is ideal for equipment control and other laboratory computer applications.

Discover how with one low investment you can combine several AIM 65s for hands-on, high-productivity microprocessor learning in classes where students don't have to wait in line. Check the high features and low prices of Rockwell's AIM 65 printing microcomputer.

Contact your local distributor or write or call AIM 65 Marketing, Electronic Devices Division, Rockwell International, P.O. Box 3669, D727, Anaheim, CA 92803, (714) 632-3824.

Rockwell International

...where science gets down to business