JUNE/JULY, 1980. ISSUE 2. $2.00

compute II.

The Single-Board COMPUTE™

AIM 65. The professional's microcomputer.

Printer, display, full keyboard. Under $500.00.

Micro power

For professional learning, designing and work, Rockwell's AIM 65 microcomputer gives you an easy, inexpensive head start. That's Rockwell Micropower!

- 20-column printer and display
- Dual cassette, TTY and general purpose I/Os
- R6502 NMOS microprocessor
- System expansion bus
- Read/write RAM memory
- Prom/ROM expansion sockets
- Self-prompt interactive monitor firmware
- Big terminal-style keyboard

For more on AIM 65 and how you can develop programs in assembly language or BASIC, write Rockwell International, Microsystems, RC 55, P.O. Box 3669, Anaheim, CA 92803 or contact your local Rockwell distributor. For application information call (714) 632-3729. For location of nearest dealer call 800-854-8099, in California 800-422-4230.

Rockwell International

...where science gets down to business

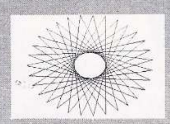
Low Cost Graphics Are Now Here

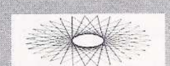
Add Printable High Resolution (320 x 200) Graphics To Your AIM-65 Microcomputer

Visible Memory Board It's a reality. The Visible Memory and graphic print software from MTU are now available for the Rockwell AIM-65. The Visible Memory gives a high resolution 320 wide by 200 high bit mapped pixel display matrix. Each dot is individually addressable for maximum utilization and speed. Thus characters, image shapes, and graphs can all be displayed separately or simultaneously if needed — maximum flexibility for you.

The Visible Memory is just that, an 8K byte RAM board that contains 2 access ports to the memory matrix. The microprocessor bus uses one port and the display refresh circuitry uses the second port. The contents of the memory bit-for-bit is precisely what is displayed. If you need 8K of RAM for a non-display application, use it! It makes no difference to the board what its contents are; program (seen in its binary pattern form) or a human recognizable display pattern. The display refresh occurs at times when the processor never goes to memory. Therefore there is no snow on the display and no wait states for the processor.

Hardcopy Too In addition MTU has engineered a software package to drive the AIM-65 printer in new ways. Three new forms of printing are possible. QUICKPRINT gives a matrix 200 across by 320 up the 2¼" wide paper. QUALITYPRINT gives two prints, each 100 x 320 which gives a higher quality (4½" wide) printed area when placed side by side. TEXTPRINT allows you to print the AIM text buffer area of memory as 10 rows of characters printed "up" the paper strip. You may specify up to 127 characters per row for the row length. The QUICK and QUALITY print modes are designed to give you fast, easy hardcopy of the Visible Memory contents. Thus you now have a graphic computer with hardcopy capability.





Graphic/Text Software Drivers To allow you to easily use this graphic display and print power, MTU has also designed the K-1008-5C software package which gives you point plotting, line drawing, character generation and a host of other subroutines. Written in assembly language, these routines may be executed from BASIC or assembly language — your choice. Text output from BASIC or the AIM monitor may also be shown on the Visible Memory display as up to 22 lines by 53 characters per line significantly enhacing the use of the AIM-65 as a computer with a CRT display.

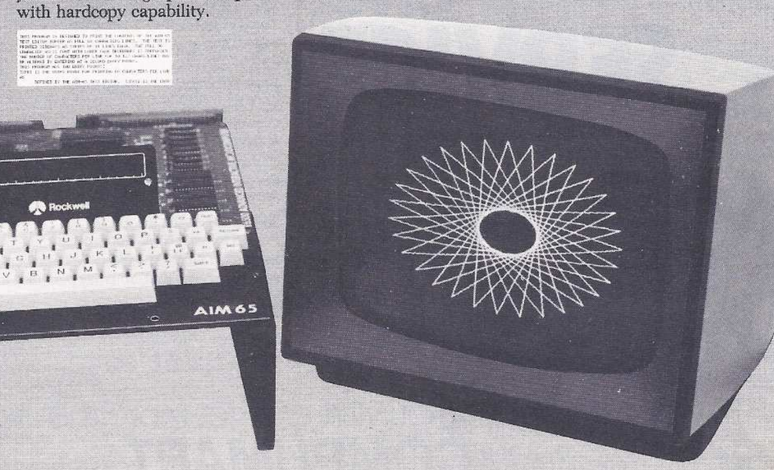
Call Us Now Many educators have been waiting for this type of price/performance to set up courses. MTU will be pleased to quote quantity purchases — call us direct — now. Demand is high and Fall is just around the corner.

K-1008 Visible Memory	$240
K-1008-5C Graphic/text software	$25
K-1009-1C AIM Print software	$25
K-1000-5 AIM-65 Power Supply	$65
K-1005-A AIM-65 Card File	$85
Many others not listed	

Call or write for our catalog listing all our AIM-65 (also our PET, KIM-1, and SYM-1) products.

As of June 1, 1980 place orders at: Micro Technology Unlimited, P.O. Box 12106, 2806 Hillsborough Street, Raleigh, North Carolina 27605

MTU
Micro Technology Unlimited
P.O. Box 12106
2806 Hillsborough Street
Raleigh, N.C. 27605
(919) 833-1458



GET AIM

The Rockwell AIM-65

For professional learning, designing and work, Rockwell's AIM-65 microcomputer gives you an easy, inexpensive headstart! The AIM's full-size keyboard, true alpha-numeric display and on-board printer give you all of the peripherals associated with a large development system, yet AIM is priced lower than the first circuit board would cost for such a system.

4K AIM-65 $515.00

AIM Enclosure $43.50
Protects the AIM 65 from damage while maintaining convenient access to on-board switches and printer.

Power Supply for the AIM 65 by MTU $79.95
This power supply from Micro Technology Unlimited provides the power and cooling capacity needed to run your AIM at peak efficiency.

AIM BASIC ROM $100.00
Written by microsoft, Rockwell AIM-65 8K BASIC is contained in two easy-to-install ROM circuits.

AIM-65 Assembler ROM $85.00
Just plug it into the AIM 65 board and your AIM is ready to do all the busy work of creating 6502 Machine-Language programs.

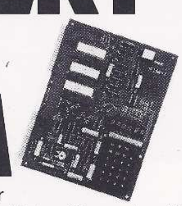
CompuMart's Complete AIM-65 System. $850.00
Save $28
Includes a 4K AIM 65 with BASIC and Assembler, an MTU power supply, Sanyo tape recorder and enclosure.

NEW! PASCAL for your AIM $125.00
Remove the BASIC interpreter chips, plug in the two PASCAL chips and your AIM is running in PASCAL.

Rockwell Motherboard for the AIM $195.00
Want to expand your AIM and keep Rockwell quality? Purchase the Motherboard from CompuMart.

and SERVICE and TOLL FREE ORDERING

FREE 10 DAY RETURN and IMMEDIATE DELIVERY and KIM



KIM-1 Microcomputer $179.00
From MOS Technology, a division of Commodore. This famous single board based computer comes to you fully assembled and tested – simply attach a power supply and you're ready to start computing. Includes three informative manuals FREE.

Micro Technology Power Supply $35.00
The perfect mate for a small KIM system. Comes fully assembled, tested, and enclosed in a Bakelite Box.

KIM 1 with power supply $204.00
Save $10 when purchased together!

KIM Enclosure $23.50
Give your KIM-1 a professional appearance and protect it from shorts and physical damage.

Current Loop/IRS-232 Adaptor $24.50
For general interfacing applications. (KIM-1 to printer, KIM-1 to modem, etc.)

Sanyo Cassette Recorder $55.00
Perfect for your KIM. Works great.

Books on KIM. (The following three books are included FREE when you buy a KIM. You may purchase them separately however).

6502 Programming Manual	$9.95
6502 Hardware Manual	$9.95
KIM User Manual	$9.95
First Book of KIM	$8.95

Buy direct from CompuMart and get more than you pay for.

270 THIRD STREET, **COMPUMART** CAMBRIDGE, MA. 02142

Toll Free 1-800-343-5504 [Ext. 313] In Mass. 617-491-2700

Table of Contents

Advertiser's Index

Subscription Information:
U.S. $9.00/6 issue year
CANADA $12.00/6 issue year (U.S. funds)
Europe: Air Subscription $22.50 U.S./6 issue year
Other areas: Inquire for air rates.

In the **United Kingdom** and **Europe, new subscribers and dealers** should contact:

L.P. Enterprises
8-11 Cambridge House
Cambridge Road
Barking, Essex
England IG1 18NT
(United Kingdom and Northern Europe)

Ing. W. Hofacker GMBH
8 Munchen 75
Postfach 437
West Germany
(Germany, Switzerland, and Austria)

Dealers in Canada should contact:
Micron Distributing
409 Queen Street West
Toronto, Ontario M5V 2A5
(416) 363-6058

Application to mail at controlled circulation postage rate is pending at Greensboro, North Carolina. Postmaster: Send change of address to compute II, Post Office Box 5406, Greensboro, NC 27403.
compute II. is published by Small System Services, Inc., 900-902 Spring Garden Street, Greensboro, North Carolina 27403. Telephone: (919) 272-4867. compute II. is published six times each year on a bimonthly schedule. Subscription cost for one year is $9.00. compute II. is available by subscription or through retail dealer sales. Subscription prices higher outside the US. (See below)

Address all manuscripts and correspondence to compute II. Post Office Box 5406, Greensboro, N.C. 27403. Materials (advertising art work, hardware, etc.) should be addressed to compute II., 900 Spring Garden Street, Greensboro, N.C. 27403.

Authors of manuscripts warrant that all materials submitted to compute II. are original materials with full ownership rights resident in said authors. By submitting articles to compute II. authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of Small System Services, Inc. Unsolicited manuscripts not accepted for publication by compute II. will be returned if author provides a self-addressed, stamped envelope. Program listings should be provided in printed form as well as machine readable form. Articles should be furnished as typed copy with double spacing. Each page of your article should bear the title of the article, date and name of the author.

Address all articles, circulation questions and other inquiries to:
compute II.
P.O. Box 5406
Greensboro, NC 27403
(919) 272-4867

The Editor's Notes

Robert Lock, Publisher/Editor

Well, the NCC is off and rolling and here I sit, plugging away at **compute II**. Oh well, I hope those of you who make it, enjoy it. As of this writing, I hear Apple will be showing the APPLE 3, and Commodore will be showing off several new goodies, among them their new touch screen, and rumor of rumors, a PASCAL oriented bank switching system for their new 80 column machine.

Speaking of Rumors

Synertek does not have a 6516 project. It was abandoned, as best I can tell, many months ago. They seemed a bit upset at the recent rumors to the contrary.

A Better Rumor

Word is drifting up from several KIM owners that Synertek Systems Marketing head, George Gregoire, has been making interesting advances regarding a new trade-in offer for KIM owners looking at moving over to SYM. Ah competition. It will be interesting to see if, and how, Commodore/MOS reacts. If you want to see what Synertek is up to, you can reach George at Synertek Systems Corporation, 150 South Wolfe Road, Sunnyvale, CA 94086.

A Non-Rumor

compute II grows on! I have been honestly suprised at the response to **compute II**. The feedback, overwhelmingly positive, has been tremendous. Included in this issue is an "Editor's Feedback" card. Please take a moment to fill it out, add a 10¢ stamp and return it to me. Your input helps give direction to the magazine(s), and is invaluable to me.

A Ballot

One of the first questions you'll encounter on the feedback card may qualify me for flakiest publisher of the year award.

I'd like your input. The idea, way back (two months ago) when we split **compute II** into its own magazine, was that everybody would get more information. **COMPUTE** was going to shrink from its Issue 3 size of 120 pages (it didn't) to around 96 pages. **compute II** is pushing now to grow past 64 pages. I'm getting a certain amount of reader/advertiser pressure to go monthly with **COMPUTE** and **compute II**. Reasons given are variations of timeliness, and gentle beefs about waiting so long for information between issues. Frankly, there's only one way I could go monthly. That would be by merging the two magazines back together, and proceeding with a monthly publication (one big happy family again) of about 120 pages. Among the disadvantages would be sharing the magazine; among the advantages would be one every month, and greater access to information. I took a look at overlap between the two magazines, and was surprised at how many of you get both. I would assume a monthly would be fine with you readers, but am equally concerned about you others. To assist you in the decision, I've printed the table of contents from Issue 4 of **COMPUTE** in this issue. Take a look and let me know what you think. Remember, this is simply a survey. I'll keep you posted on the results.

Writing for compute II

I've really been pleased with the wealth of information that started coming in after issue #1 hit the readers. Keep it up! As I've said before, I'm as interested in short application notes as in long feature length articles. I think a good mix of both helps keep the magazine useful, readable, and handy. Articles should be submitted to my attention at **compute II, Post Office Box 5406, Greensboro, NC 27403.** If you want to hear from me very promptly, enclose a self-addressed post card, and we'll acknowledge receipt as soon as we get it.

Here are a few guidelines on submitting articles/listings:

1. Text should be typed, with double-spacing, and upper/lower case letters. That last one may seem strange, but I've had several articles that were all upper case. Before they can be typeset, I have to go through and underline every letter in the entire article that's really supposed to be upper case. If you have to submit something in upper case only, please take the time to go through and underline each letter that really should be upper case.
2. Be as clear as possible with your drawings and figures. While we redraw many of them, we don't have to if you're neat and precise the first time through.
3. Use a new ribbon for your program listings, and attempt to make your comments and data statements consistent in width with the rest of the program. Here's an example. We photographically reproduce (and generally reduce) most of the listings that appear in the magazine. The extent of the reduction is governed by the longer lines in your program. Generally these end up being comments and/or data statements. If you attend to these small details at the time you're generating the listing for us, it makes the job easier on this end, and helps maintain the quality and readability of the magazine. ©

Some A/D And D/A Conversion Techniques

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

INTRODUCTION

The purpose of this paper is to describe some A/D conversion circuits and programs that can be used with 6502 based microcomputer systems. A digital-to-analog (D/A) converter is also described. Our motivation for this project was an NSF Short Course on the Science of Sound. The complete project was to be a circuit that would sample some waveform, from an electric guitar for example, and a program that would perform a Fast Fourier Transform (FFT). The Fast Fourier Transform program has not yet been completed, but the necessary A/D circuit and driver programs have been completed and are herein described. A digital-to-analog converter allows the sampled waveform to be displayed on an oscilloscope, producing a much improved storage oscilloscope over our original "storage scope" described in THE BEST OF MICRO, Volume 1, page 30, and Volume 2, page 61. Some results of our experiments are also included.

The analog-to-digital converter is based on the AD570, an 8-bit A/D converter sold by Analog Devices, Route 1 Industrial Park, P.O. Box 280, Norwood, MA 02062. Its nominal conversion time is 25 microseconds, allowing a maximum sampling rate of 40,000 kHz. (The time necessary to read the converter and store the data will reduce this rate.) The AD570 requires *no external components,* and can be operated either in a bipolar or a unipolar mode. We chose it because it is inexpensive, relatively fast, and easy to interface.

The D/A converter we used is a Signetics NE5018. It is also easy to interface because it has input latches. It can be operated with few external components, but it is not an exceptionally fast converter. A better choice would have been the Analog Devices 565, but this would have required an 8-bit latch.

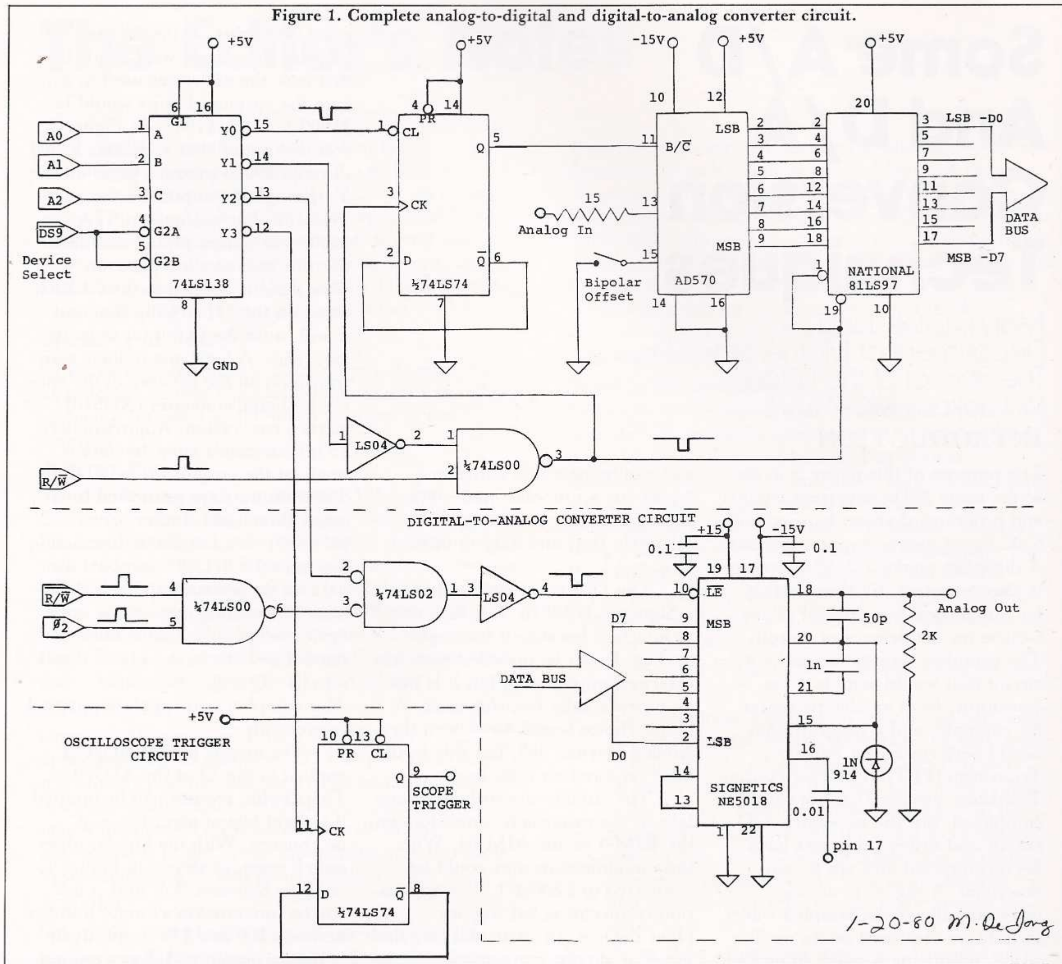
The circuits shown here interface to the expansion connectors on the KIM-1 or the AIM 65. With little modification they could be connected to a SYM-1. The application connector is left free for other devices. In particular, we had hoped to do our mathematics for the FFT program with an AM9511 Arithmetic Processor Unit interfaced to the I/0 ports on the application connector. In any case, Appendix A suggests a circuit for interfacing the converters to a 6522 Versatile Interface Adapter.

Description Of The Circuit

The complete A/D, D/A, and oscilloscope trigger circuitry is shown in Figure 1. This circuit was used to interface the converters to an AIM 65 microcomputer, and all the necessary connections are available at the expansion connector, including the DS9 device select pulse. The same circuit could be used with a SYM-1 if the DS18 device select pulse, available on the SYM-1 expansion connector, were used. In that case the addresses used to activate the various circuits would be $1800 through $1803. In Figure 1 you will notice that addresses $9000 through $9003 produce pulse on the Y_0 through Y_3 outputs on the 74LS138. For example, a STA $9000 instruction produces a negative one microsecond pulse on Y_0. This pulse is applied to the CLEAR input on the 74LS74 flip-flop and it will cause the Q output to go to logic zero. A logic one to logic zero transition on the B/C pin of the analog-to-digital converter (AD570) starts a conversion. Approximately 25 microseconds later the data is ready at the outputs of the AD570. These outputs are connected to an octal, three-state, buffer-driver (81LS97). A LDA $9001 instruction activates the 81LS97 and puts the data on the microcomputer's data bus. The trailing edge of the same device select pulse that enables the 81LS97 clocks the 74LS74 back into its "Q high" logic state. This completes one analog-to-digital conversion.

The analog input voltage is applied to pin 13 of the AD570. The 15 ohm resistor can be omitted if a slight loss of precision is of no concern. With the bipolar offset switch open, as shown in Figure 1, voltages between -5 V and +5 V will be converted to a binary number between $00 and $FF respectively. A binary output of $80 corresponds to pin 13 being at zero volts. If the bipolar output switch is closed, the AD570 will read voltages between 0 V and +10 V. The AD 570 will also work with a negative supply voltage of -12 V rather than the -15 V shown in Figure 1. Although a "data ready" signal is available on the AD570, we chose to use software to wait for the conversion to be completed. One final note on the AD570: the input impedance for the analog input is only about 5 k ohm. Consequently it makes a very poor voltmeter unless a high impedance (a voltage follower circuit, for example) amplifier is placed between the analog input

Figure 1. Complete analog-to-digital and digital-to-analog converter circuit.



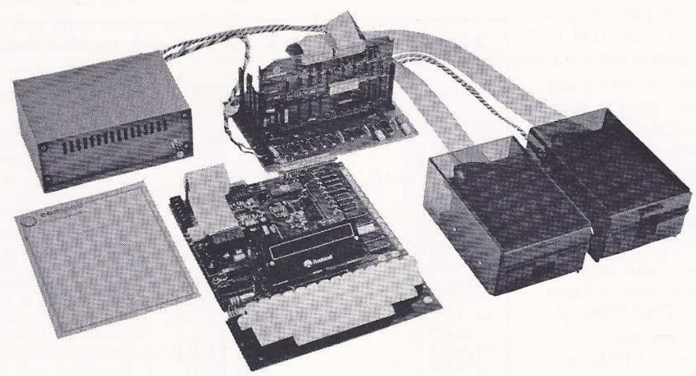
and the source of the analog voltage. We used the instrumentation amplifier (AD521) circuit given in Figure 3.

The digital-to-analog circuit is activated by a STA $9002 instruction. That is, to write to the NE5018, use a STA $9002 instruction and new data will be latched into the NE5018. The voltage at pin 18 is proportional to the 8-bit number written to the NE5018, shown operating in its bipolar mode in Figure 1.

Finally, if you wish to use the A/D and D/A converters to implement a storage scope, then you may wish to use the other half of the 74LS74 flip-flop to produce a trigger signal. Two STA $9003 instructions produce a pulse on the Q output of the flip-flop that can be used as a trigger pulse to start the oscilloscope sweep.

For all you purists who think that if it can't be done with a KIM-1 then it isn't worth doing, the circuit in Figure 1 will also work with a KIM-1. We simply replaced the DS9 device select with the K1 device select produced by the on-board 74145. A 3.3k pullup resistor is required. The entire circuit can be simplified somewhat with the modifications shown in Figure 2. The various 04s, 02s, and 00s are replaced with two gates on a 74LS32, the K1 device select goes to the 74LS138, and the remainder of the circuit is the same as in Figure 1. The RAM-RW signal on the KIM-1, which does not have the same meaning on the AIM 65, can be used to simplify the logic, as shown in Figure 2. The same simplification could probably be used with the SYM-1.

The preamplifier circuit we used to study several waveforms is shown in Figure 3. We used it as a gain of 100, and as a gain of 1000, amplifier, but it can also be operated at unity gain by adjusting R1 to be 100k. In this latter



mode it simply provides a high impedance buffer for the AD570. The AD521 is a differential amplifier with a differential input impedance of about 3 X 10^9 ohms. Pin 3, the - input, need not be grounded but can be connected directly to the input voltage source.

The circuits of Figures 1 - 3 provide a complete A/D and D/A system that can be used for a large variety of applications including voltage measurements, temperature measurements, and the storage scope application described here.

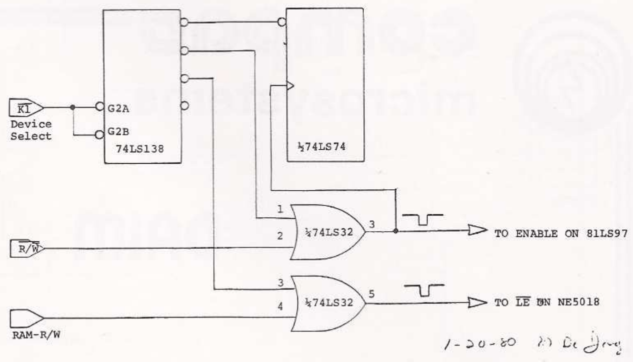
A/D and D/A Converter Software

The program in Example 1 was designed to work with the AIM 65 or any other system that has a 6522 VIA available for timing purposes. The addresses used to start the conversion, read the A/D converter, load the D/A converter, and trigger the oscilloscope are $9000 through $9003, respectively. If a device select other than the DS9 is used to enable the 74LS138 decoder, then these addresses must be changed accordingly. For example, if the DS18 select on the SYM-1 is used, then these addresses become $1800 through $1803, respectively. Since the KIM-1 does not have a 6522, we wrote another program that will work for it, and this program is listed in Example 2.

The program in Example 1 has a maximum sampling rate of one sample every 32 microseconds, or 31,250 Hz. It allows the AD570 exactly 28 microseconds to make a conversion if the T1 timer is loaded with $0000. If you have an AD570 that is slightly faster, try taking out the NOP instruction at $0F3A. If your AD570 is slightly slower, insert some extra NOP instructions after $0F3A. Change the various branch offsets accordingly. You can tell if you are giving the AD570 enough time by examining the data it returns.

The program in Example 1 has the following features. It continuously samples the analog voltage at the input of the A/D

Figure 2. Modifications for the KIM-1.



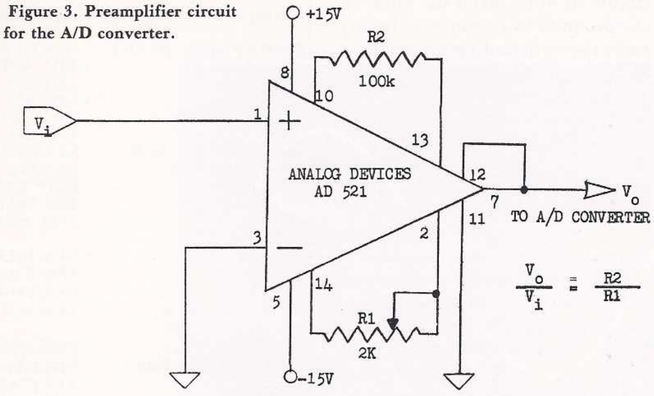
Example 1. A/D and D/A driver program for 6522 based timing.

```
$0F00 8D 00 90  START   STA CVNST   Pulse 74LS74 flip-flop to be in a
                                    known
 0F03 AD 01 90          LDA A/D     condition with Q at logic one.
 0F06 A2 00             LDX $00     Initialize X register to zero
 0F08 A9 40             LDA $40     Initialize ACR of 6522 to put T1 in
 0F0A 8D 0B A0          STA ACR     its free-running mode.
 0F0D A9 00             LDA $00     Clear accumulator.
 0F0F F0 03             BEQ TEST    Branch to start the first conversion.
 0F11 AD 01 90  AGN     LDA A/D     Read A/D converter
 0F14 8D 00 90  TEST    STA CVNST   Start a conversion.
 0F17 8D 02 90          STA D/A     Output A/D to D/A converter.
 0F1A C5 00             CMP TRIG    Compare conversion result with trigger
 0F1C B0 0E             BCS SAMPLE  level. Branch to sample an additional
 0F1E A5 01             LDA TIMLO   255 points if A/D exceeds trigger level.
 0F20 8D 04 A0          STA T1LL    Load 6522 with the number of micro-
 0F23 A5 02             LDA TIMHI   seconds between conversions.
 0F25 8D 05 A0          STA T1LH    Start timer.
 0F28 90 37             BCC AGN     Branch to read A/D.
 0F2A AD 01 90  MORE    LDA A/D     Read A/D
 0F2D 8D 00 90  SAMPLE  STA CVNST   Start sampling waveform.
 0F30 9D 00 02          STA TAB,X   Previous result into table.
 0F33 E8                INX         X keeps track of the number of
                                    conversions.
 0F34 F0 0C             BEQ OUT     When X = 00, 256 conversions are
                                    complete.
 0F36 AD 04 A0          LDA T1CL    Clear T1 interrupt flag.
 0F39 EA                NOP         Fill in the 25 microsecond conversion
 0F3A EA                NOP         time with no operation instructions.
 0F3B 2C 0D A0  LOAF    BIT IFR     Has T1 timed-out?
 0F3E 70 E9             BVS MORE    Yes, get another conversion.
 0F40 50 F9             BVC LOAF    No, wait for timer.
 0F42 8D 03 90  OUT     STA SCPTRG  Trigger scope.
 0F45 EA                NOP         Use an eight microsecond pulse to
 0F46 EA                NOP         trigger scope.
 0F47 8D 03 90          STA SCPTRG
 0F4A BD 00 02  NEXPT   LDA TAB,X   Get some data from the table.
 0F4D 8D 02 90          STA D/A     Output it to the D/A and the scope.
 0F50 E8                INX
 0F51 D0 F7             BNE NEXPT   Branch to get more data.
 0F53 F0 ED             BEQ OUT
$0000 = TRIG; load with desired triggering level but not $00.
$0001 = TIMLO; low-order byte of time interval between samples (microseconds).
$0002 = TIMHI; high-order byte of time interval between samples.
$0200 = TAB: base address of table to store 256 samples.
$9000 = CNVST; a STA CVNST instruction will start an A/D conversion.
$9001 = A/D; the analog-to-digital converter is read at this location.
$9002 = D/A; write to this location to perform a digital-to-analog conversion.
$9003 = SCPTRG; write to this location to trigger the oscilloscope.
```

converter. When the conversion result exceeds a preassigned level stored in TRIG (location $0000), the program proceeds to sample another 255 points on the waveform at a rate determined by the numbers stored in TIMLO (location $0001) and TIMHI (location $0002). The 256 data points are stored in page two of memory. Once the data have been obtained, the program proceeds to read the data out, one point at a time, to the D/A converter for the purpose of displaying the values on an oscilloscope. Each time the 256 points are output to the D/A converter, a trigger pulse is also supplied. Since the conversion time is 32 microseconds with this program, there is no use loading the T1 timer with a number less than 32 unless you wish to sample at the maximum rate. In that case, put $00 in location $0001. In the program in Example 1, T1 is in its free running mode, so its interrupt flag, IFR6, will be set every N + 1 microseconds, where N is the 16-bit number loaded into T1 from locations $0001 and $0002. Be sure to load the locations TRIG, TIMLO, and TIMHI before running the program. The program comments should explain how the program works. The first two instructions may produce a dummy conversion, but their real function is to put the 74LS74 flip-flop in a condition with Q at logic one. The program consists of three main loops. The AGN loop continuously samples the incoming data, and the program branches out of this loop to the MORE loop when the incoming voltage exceeds the trigger level. In the MORE loop another 255 points are produced. When this data has been gathered, the program branches to the OUT loop to output the 256 points to the D/A converter.

The program in Example 2 works in about the same way with the same purpose in mind, but it was used on the KIM-1. The sampling rate with this program is once every 39 microseconds, or 25641 Hz. Its speed could be

Figure 3. Preamplifier circuit for the A/D converter.



Example 2. A/D and D/A driver program for a KIM-1 interface.

0300 8D 00 04	START	STA CVNST	Pulse 74LS74 flip-flop to be in a known
0303 AD 01 04		LDA A/D	condition with Q at logic one.
0306 A2 00		LDX $00	Initialize X register to zero.
0308 A9 00		LDA $00	Initialize accumulator to zero.
030A 8D 00 04	TEST	STA CVNST	Start A/D conversion.
030D 8D 02 04		STA D/A	Previous result into D/A converter.
0310 C5 00		CMP TRIG	Compare conversion result with trigger
0312 B0 16		BCS SAMPLE	level. Branch to sample 256 points if
0314 EA		NOP	voltage exceeds trigger level.
0315 EA		NOP	Delay with no-operation instructions
0316 EA		NOP	until the 25 microsecond conversion
0317 EA		NOP	time is completed.
0318 EA		NOP	
0319 EA		NOP	
031A EA		NOP	
031B AD 01 04		LDA A/D	Read A/D converter.
031E 90 EA		BCC TEST	Branch to start another conversion.
0320 8D 00 04	MORE	STA CVNST	Start an A/D conversion.
0323 9D 00 02		STA TAB,X	Previous result into table.
0326 E8		INX	X keeps track of number of conversions.
0327 F0 13		BEQ OUT	When X = 00, 256 conversions are complete.
0329 A5 01	SAMPLE	LDA TIME	Get time in microseconds from $0001.
032B 8D 04 17		STA TIMER	Store in divide-by-one timer.
032E EA		NOP	Fill in time to make 25 microseconds
032F EA		NOP	before reading A/D converter.
0330 EA		NOP	
0331 EA		NOP	
0332 AD 01 04		LDA A/D	Read converter.
0335 2C 07 17	LOAF	BIT TIMER	Has timer timed out?
0338 30 E6		BMI MORE	Yes, then start another conversion and
033A 10 F9		BPL LOAF	store the result of the last. Otherwise
033C 8D 03 04	OUT	STA SCPTRG	wait. Trigger the oscilloscope.
033F A2 00		LDX $00	
0341 8D 03 04		STA SCPTRG	
0344 BD 00 02	NEXPT	LDA TAB,X	Get some data from the table.
0347 8D 02 04		STA D/A	Output it to the D/A and the oscillo-
034A E8		INX	scope.
034B D0 F7		BNE NEXPT	Branch to get more data.
034D F0 ED		BEQ OUT	Return to output table again.

$0000 = TRIG; load with desired triggering level.
$0001 = TIME: load with time (in microseconds) between samples.
$0400 = CVNST; a STA CVNST instruction will start an A/D conversion.
$0401 = A/D; the analog-to-digital converter is read at this location.
$0402 = D/A; write to this location to perform a digital-to-analog conversion.
$0403 = SCPTRG; write to this location to trigger the oscilloscope.

improved to be about the same as the program in Example 1. In any case, the on-board timers on the KIM-1 were used to produce the necessary timing. Again, the trigger level is stored in $0000, and the time is stored in $0001. The divide-by-one timer at $1704 on the KIM-1 was used, but the other timers may also be used.

The last program listing for the circuit in Figure 1 is a program that can be used to sample a waveform at as many points as your R/W memory will allow. Rather than use just one page of R/W memory for storing the waveform, it will use as many pages as you have available. The maximum sampling rate for this program is one sample every 43 microseconds or 23256 Hz. The program in Example 3 uses several of the same locations as the program in Example 1. The trigger level is stored in TRIG at $0000. The 16-bit number giving the number of microseconds between samples is stored in TIMLO at $0001 and TIMHI at $0002. The low-order byte of the base address of the table to store the conversion data is in location TAB at $0003. Normally this location initialized to $00. The high-order byte of the base address (page number) of the table is stored in TAB + 1 at $0004. For our experiments with the AIM 65 we used pages $02 through $0E. The page number of the last page you wish to fill with data is incremented by one and stored in location END at $0005. Thus if page $0E is the last page to be used to store data, then $0F is stored in END. Load location $0006, STARTP with the same value you put in $0004 if you wish to output all the results to the D/A for display on the oscilloscope.

The program in Example 3 samples an incoming waveform at N*256 points where N is the number of pages used to store the data. It then outputs all of these points to the D/A converter *at the same rate* that it sampled the waveform. If you want to output the results faster, replace the BIT IFR and BVC WAIT instructions at $0f5D with NOPs.

Example 3. N-Page A/D Conversion and Storage Program

```
$0F00 8D 00 90  START    STA CVNST    Pulse 7474 to be in a known condition,
 0F03 AD 01 90           LDA A/D      with Q at logic one.
 0F06 A0 00              LDY $00      Initialize Y to zero for indirect indexed
 0F08 A9 40              LDA $40      addressing that follows.
 0F0A 8D 0B A0           STA ACR      Put 6522 T1 in free-running mode.
 0F0D A9 00              LDA $00      Clear A.
 0F0F 8D 00 90  AGN      STA CVNST    Start a conversion.
 0F12 8D 02 90           STA D/A      Output result to D/A converter
 0F15 C5 00              CMP TRIG     Compare conversion result with trigger
 0F17 B0 21              BCS SAMPLE   level.
 0F19 A5 01              LDA TIMLO    Get low-order byte of time between
                                      conversions.
 0F1B 8D 04 A0           STA T1LL     Result into T1.
 0F1E A5 02              LDA TIMHI    Get high-order byte of time between
 0F20 8D 05 A0           STA T1LH     conversions
 0F23 AD 01 90           LDA A/D      Read A/D converter to get conversion
                                      level.
 0F26 90 E7              BCC AGN      Return to compare with trigger level.
 0F28 8D 00 90  DATA     STA CVNST    Start an A/D conversion.
 0F2B 91 03              STA (TAB),Y  Result of previous conversion into table.
 0F2D C8                 INY
 0F2E D0 0A              BNE EQUAL    Branch around the page number incre-
                                      ment routine.
 0F30 E6 04              INC TAB + 1  Increment page number
 0F32 A5 04              LDA TAB + 1  Now compare it with the ending page
 0F34 C5 05              CMP END      number.
 0F36 90 09              BCC MORE     Fill another page.
 0F38 B0 14              BCS NOMORE   Table is filled, branch to output the table.
 0F3A EA        SAMPLE   NOP          These NOPs equalize the time between
 0F3B EA                 NOP          loading the table when no page boundary
 0F3C EA                 NOP          is crossed and when a page boundary is
 0F3D EA                 NOP          crossed.
 0F3E EA                 NOP
 0F3F A5 05              LDA TAB + 2  This is also a dummy instruction.
 0F41 AD 04 A0  MORE     LDA T1CL     Clear the T1 interrupt flag.
 0F44 AD 01 90           LDA A/D      Read the A/D converter.
 0F47 2C 0D A0  LOAF     BIT IFR      Has the timer timed-out?
 0F4A 70 DB              BVS DATA     Start another conversion.
 0F4C 50 F9              BVC LOAF     Wait for timer.
 0F4E 8D 03 90  NOMORE   STA SCPTRG   Trigger scope.
 0F51 A5 06              LDA STARTP   Reload TAB with starting page number.
 0F53 85 04              STA TAB + 1
 0F55 AD 04 A0  RPT      LDA T1CL     Clear T1 interrupt flag.
 0F58 B1 03              LDA (TAB),Y  Get data from the table.
 0F5A 8D 02 90           STA D/A      Output it to D/A.
 0F5D 2C 0D A0  WAIT     BIT IFR      Test T1 flag.
 0F60 50 FB              BVC WAIT
 0F62 C8                 INY
 0F63 D0 F0              BNE RPT      Get some more data for the D/A converter.
 0F65 E6 04              INC TAB +1
 0F69 C5 05              CMP END
 0F6B 90 EA              BCC RPT      Get more data from a new page.
 0F6D B0 E1              BCS NOMORE   Output the table again.
```

We used this program to see how the waveform from a plucked guitar string varied with time, but we couldn't help connecting a microphone to the AD521 and using the program to output this speech sound to an audio amplifier. The results are quite good, even though we made no attempt to include low-pass filters in either the input or output circuits. The word spoken into the microphone and output to an audio amplifier is intelligible and one can easily identify the person who made the sound. We had enough storage capability on the AIM 65 to store one three-syllable word. If you want a project, you might try improving the circuit and program to do a better job with speech.

Results

In Figure 4 we show a photograph of the results of sampling a 1000 Hz sine wave at a rate of about 25,000 Hz. The photograph shows 256 points on the sine wave. Since we did not have a camera for our oscilloscope, the pictures were taken through a Celestron 5'' telescope, placed about 25 ft. from the oscilloscope. Figure 5 shows the scope trace expanded to show just one cycle of the same waveform in Figure 4. Figure 6 shows 256 points of a 100 Hz sine wave sampled about once every 40 microseconds, while Figure 7 shows 256 points on a 10 Hz sine wave sampled once every 2000 microseconds. Figure 8 is the waveform of the A string of an electric guitar just after being plucked. The sampling rate in this case was about one sample every 85 microseconds. Finally, in Figure 9 we show a sampled version of a 2500 Hz sine wave. Clearly the system still does a pretty good job of reconstructing a 2500 Hz sine wave, but the information in frequencies much above 5000 Hz will be lost. Hopefully these pictures are worth a thousand words.

Figure 4. 256 points on a 1000 HZ Sine wave.

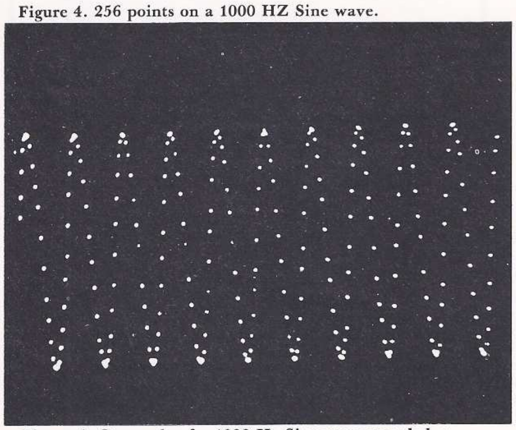


Figure 5. One cycle of a 1000 Hz Sine wave sampled at about 24,000 Hz.

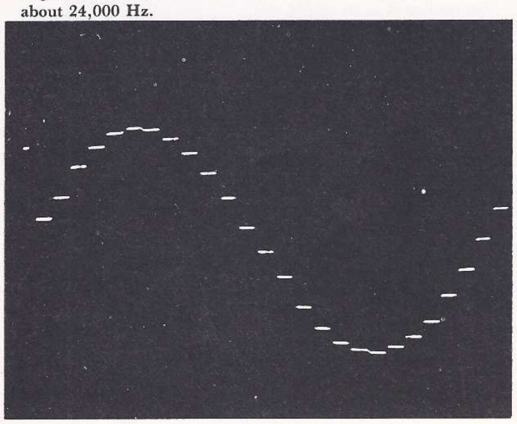


Figure 6. 256 points on a 100 Hz Sine wave.

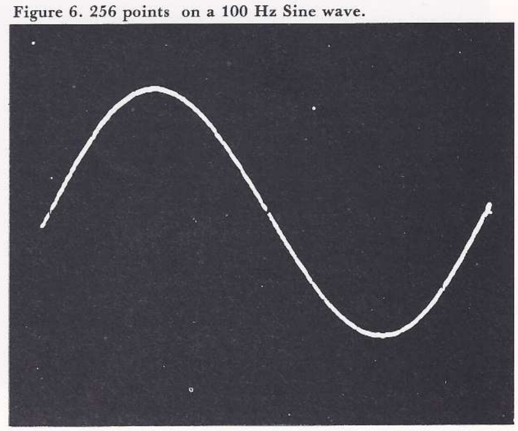


Figure 7. 256 points on a 10Hz Sine wave.

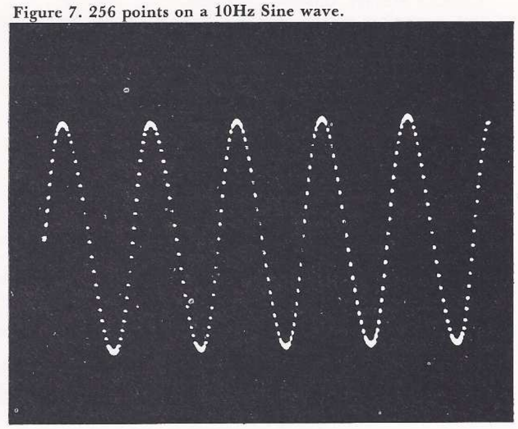


Figure 8. Plucked A string on an electric guitar.

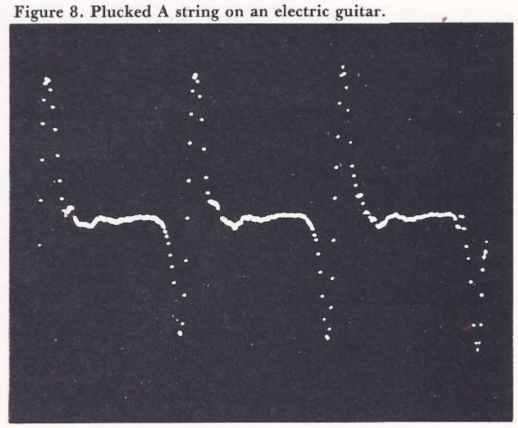


Figure 9. Several cycles of a 2500 Hz Sine wave.

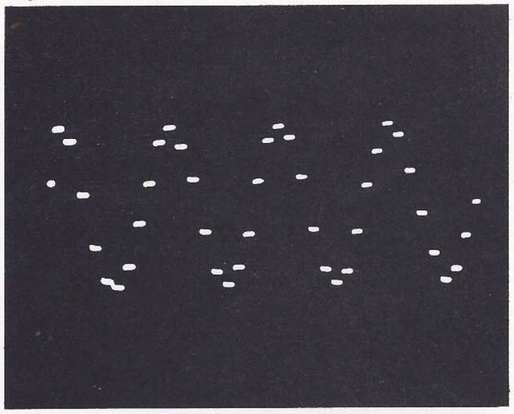
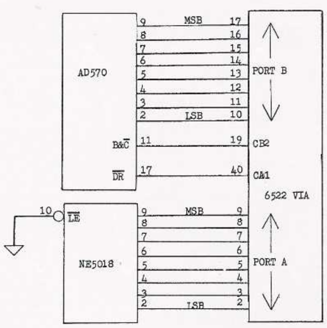


Figure 10. Interfacing the AD570 and the NE5018 to a 6522 Versatile Interface Adapter. Only data and control connections are shown in this figure. Refer to Figure 1 for the other details.



Appendix A. Interfacing The Converters To A 6522 VIA

The AD570 analog-to-digital converter and the NE5018 converter can easily be interfaced to a 6522, eliminating the need for most of the control logic shown in Figure 1. AIM 65 and SYM-1 users may wish to use the 6522 accessed at the application connector and the circuit shown in Figure 10. Note that only the data and control connections are shown in Figure 10. The other circuitry, mainly a few resistors and capacitors, can be found in Figure 1, as are the necessary power connections. This circuit eliminates the 74LS138, the 74LS74, the 81LS97, and the various NAND, NOR, and INVERTER chips. The CA2 pin on the 6522 could be used as an output to trigger the oscilloscope. Below find a short assembly language program that will collect 256 conversions and store them. This program has not been tried.

```
        LDA $FF     Set up Port A as an output port.
        STA DDRA
        LDA $18     Set up the ACR so the shift register shifts out (on CB2)
        ORA ACR     at the clock rate.
        STA ACR
        LDA $FE     Set up the PCR so a negative transition on CA1 sets
        AND PCR     its interrupt flag.
        STA PCR
HERE    LDA $03     The shift register is used to supply a 4 microsecond
        STA SR      pulse to the A/D converter to start a conversion.
        LDA $02     Test to see if conversion is complete by reading IFR1.
TEST    AND IFR
        BEQ TEST
        LDA IRB     Read the A/D converter.
        STA TAB,Y   Store the result in a one page table.
        INY
        BNE HERE    When Y = 0, 256 conversions are complete.
                    Otherwise get another conversion.
OUT
```

©

Some Routines From Microsoft Basic

Jim Butterfield, Toronto

Routines were identified by examining specific machines. There may well be other versions of Basic on these machines; the user is urged to exercise caution.

OSI is from a C2-4 machine. KIM is a cassette tape version. SYM and AIM are the ROM versions.

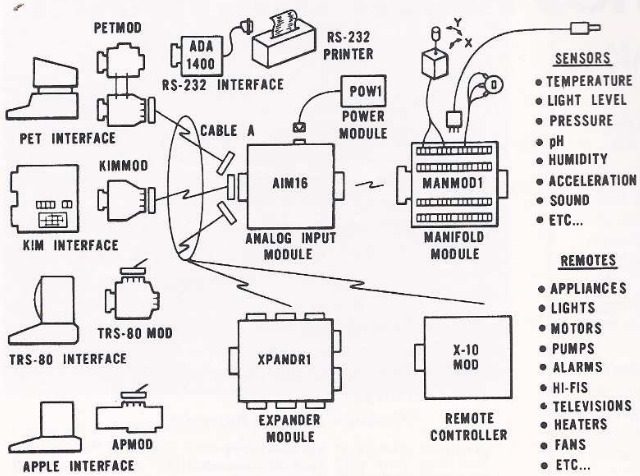
The addresses given identify the start of the area in which the described routine lies. This may not be the proper program entry point or calling address.

KIM	SYM	AIM	OSI	Description
2000	C003	B00A	A000	Action addresses for primary keywords
203A	C03D	B044	A038	Action addresses for functions
2068	C06B	B072	A066	Hierarchy and action addresses for operators
2086	C089	B090	A084	Table of Basic keywords
2169	C16E	B175	A164	Basic messages, mostly error messages
2274	C1AB	B1AC	A1A1	Search the stack for FOR or GOSUB activity
22A2	C1D9	B1DA	A1CF	Open up space in memory
22E5	C21C	B21D	A212	Test: stack too deep?
22F2	C229	B22A	A21F	Check available memory
231F	C256	B257	A24C	Send canned error message, then:
2348	C27E	B27F	A274	Warm start; wait for Basic command
236A	C2A0	B29D	A295	Handle new Basic line input
23F1	C32C	B329	A32E	Rebuild chaining of Basic lines
2420	C359	B356	A34B	Receive line from keyboard
2466	C39F	B3AE	A3A6	Crunch keywords into Basic tokens
24F2	C427	B436	A432	Search Basic for given line number
2521	C456	B465	A461	Perform NEW
253C	C472	B481	A68C	Perform CLEAR
256B	C49F	B4AE	A4A7	Reset Basic execution to start
2579	C4AC	B4BC	A4B5	Perform LIST
2608	C535	B55C	A556	Perform FOR
26AA	C5DA	B601	A5FF	Execute Basic statement
26CB	C60A	B631	A61A	Perform RESTORE
26DA	C619	B640	A62C	Check stop key
26E8	C622	B65C	A638	Perform STOP or END
2711	C64B	B685	A661	Perform CONT
272B	C665		A67B	Perform NULL
273C	C676	B69F	FFF7	Perform SAVE
278C	C6B7		FFF4	Perform LOAD
		B6AB		Special AIM input routines
27CA	C707	B6EC	A691	Perform RUN
27D5	C712	B6F7	A69C	Perform GOSUB
27F2	C72F	B714	A6B9	Perform GOTO
281F	C75C	B741	A6E6	Perform RETURN, then:
2845	C782	B767	A70C	Perform DATA: skip statement
2853	C790	B775	A71A	Scan for next Basic statement
2857	C793	B778	A71D	Scan for next Basic line
2875	C7B2	B797	A73C	Perform IF, and perhaps:
2888	C7C5	B7AA	A74F	Perform REM: skip line
2898	C7D5	B7BA	A75F	Perform ON
28B8	C7F5	B7DA	A77F	Input fixed-point number
28F2	C82F	B814	A7B9	Perform LET
		B89D		Enable printer
297B	C8B8	B8A9	A829	Perform PRINT
2A13	C94F	B94A	A8C3	Print string from memory
2A35	C971	B967	A8E0	Print single format character
2A59	C991	B988	A904	Handle bad input data
2A7E		B9AD		Perform GET
2A8D	C9B0	B9BC	A923	Perform INPUT
2AB0	C9DC	B9E7	A946	Prompt and receive input
2AB9	C9E5	B9F0	A94F	Perform READ
2BA2	CAB4	BADC	AA1C	Canned Input error messages
2BC6	CAD8	BB00	AA40	Perform NEXT
2C34	CB43	BB59	AAAD	Check type mismatch
2C48	CB57	BB7F	AAC1	Evaluate expression
2D82	CC9F	BCB9	ABF5	Evaluate expression within parentheses
2D88	CCA5	BCBF	ABFB	Check parenthesis, comma
2D99	CCB6	BCD0	AC0C	Syntax error exit
2D9E	CCBB	BCD5	AC11	Setup for functions
2DA5	CCC2	BCDC	AC18	Variable name setup
2DC5	CCE6	BD00	AC27	Set up function references
2E04	CD25	BD3F	AC66	Perform OR, AND
2E34	CD55	BD6F	AC96	Perform comparisons
2E9F	CE11	BDDA	AD01	Perform DIM
2EA9	CE5F	BDE4	AD0B	Search for variable
2F3D	CEF3	BE78	AD8B	Create new variable
2FA3	CF57	BEDC	ADE6	Setup array pointer
2FB4	CF68	BEED	ADF7	Evaluate integer expression
2FD4	CF8B	BF10	AE17	Find or make array
3181	D138	C0BD	AFAD	Perform FRE, and:
3195	D14C	C0D1	AFC1	Convert fixed-to-floating
31A2	D159	C0DE	AFCE	Perform POS
31A8	D15F	C0E4	AFD4	Check not Direct
31B2	D16C	C0F1	AFDE	Perform DEF
31E0	D19A	C11F	B00B	Check FNx syntax
31F3	D1AD	C132	B01E	Evaluate FNx
3266	D21E	C1A3	B08C	Perform STR$
3276	D22E	C1B3	B09C	Do string vector
3288	D240	C1C5	B0AE	Scan, set up string
32EF	D2A9	C232	B115	Build descriptor
3321	D2DB	C264	B147	Garbage collection
3434	D3F2	C37B	B24D	Concatenate
3471	D42F	C3B8	B28A	Store string
349A	D458	C3E1	B2B3	Discard unwanted string
34D2	D490	C419	B2EB	Clean descriptor stack
34E3	D4A1	C42A	B2FC	Perform CHR$
34F7	D4B5	C43E	B310	Perform LEFT$
3523	D4E1	C46A	B33C	Perform RIGHT$
352E	D4EC	C475	B347	Perform MID$
3556	D516	C49F	B36F	Pull string data
3573	D531	C4BA	B38C	Perform LEN
3579	D537	C4C0	B392	Switch string to numeric
3582	D540	C4C9	B39B	Perform ASC
3592	D550	C4D9	B3AB	Get byte parameter
35A4	D562	C4EB	B3BD	Perform VAL
35E3	D5A1	C52A	B3FC	Get two parameters for POKE or WAIT
35EF	D5AD	C536	B408	Convert floating-to-fixed
3605	D5C3	C54C	B41E	Perform PEEK
3610	D5DA	C563	B429	Perform POKE
3619	D5E3	C56C	B432	Perform WAIT
3635	D5FF	C588	B44E	Add 0.5
363C	D606	C58F	B455	Perform subtraction
364E	D618	C5A6	B467	Perform addition
3765	D6FD	C686	B537	Complement accum#1
379C	D734	C6BD	B564	Overflow exit
37A1	D739	C6C2	B569	Multiply-a-byte
3802	D772	C6FB	B59C	Constants
3830	D7A0	C729	B5BD	Perform LOG
386E	D7DE	C76A	B5FB	Perform multiplication
3904	D842	C7CB	B64D	Unpack memory into accum#2
392F	D86D	C7F6	B673	Test & adjust accumulators
394C	D88A	C813	B690	Handle overflow and underflow
395A	D898	C821	B69E	Multiply by 10
3971	D8AF	C838	B6B5	10 in floating binary
3976	D8B4	C83D	B6B9	Divide by 10
3987	D8C5	C846	B6CA	Perform divide-by
398C	D8CA	C851	B6CF	Perform divide-into
3A1A	D958	C8E1	B74B	Unpack memory into accum#1
3A3F	D97D	C906	B76B	Pack accum#1 into memory
3A74	D9B2	C93B	B79B	Move accum#2 to #1
3A84	D9C2	C94B	B7AB	Move accum#1 to #2
3A93	D9D1	C95A	B7BA	Round accum#1
3AA3	D9E1	C96A	B7CA	Get accum#1 sign
3AB1	D9EF	C978	B7D8	Perform SGN
3AD0	DA0E	C997	B7F5	Perform ABS
3AD3	DA11	C99A	B7F8	Compare accum#1 to memory
3B13	DA51	C9DA	B831	Floating-to-fixed
3B44	DA82	CA0B	B862	Perform INT
3B6B	DAA9	CA32	B887	Convert string to floating-point
3C0A	DB3B	CABD	B912	Get new ASCII digit
3C3F	DB70	CAF2	B947	Constants
3C4E	DB7F	CB01	B953	Print IN, then:
3C55	DB86	CB0C	B95A	Print Basic line #
3C69	DB9A	CB1C	B96E	Convert floating-point to ASCII
3D99	DCCA	CC4C	BA96	Constants
3DC2	DCF3	CC75	BAAC	Perform SQR
3DCC	DCFD	CC7F	BAB6	Perform power function
3E05	DD36	CCB8	BAEF	Perform negation
3E10	DD41	CCC3	BAFA	Constants
3E3E	DD6F	CCF1	BB1B	Perform EXP
3E91	DDC2	CD44	BB6E	Series evaluation
3EDB	DE0C	CD8E	BBB8	RND constants
3EE3	DE14	CD96	BBC0	Perform RND
3F1F		CDD2	BBFC	Perform COS
3F26		CDD9	BC03	Perform SIN
3F6F		CE22	BC4C	Perform TAN
3F9B		CE86	BC78	Constants
3FD3			BC99	Perform ATN
4003			BCC9	Constants
4041	DE50	CE86	BCEE	CHRGET sub for zero page

Remaining routines are Basic startup.

©

MICROCOMPUTER MEASUREMENT and



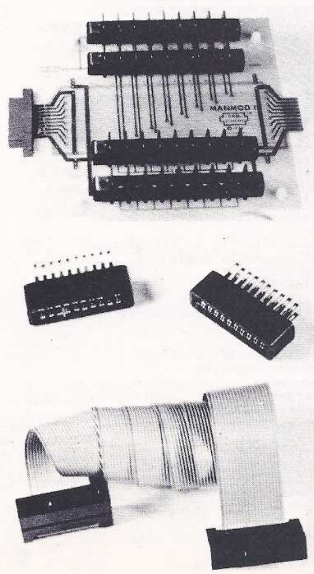
The world we live in is full of variables we want to measure. These include weight, temperature, pressure, humidity, speed and fluid level. These variables are continuous and their values may be represented by a voltage. This voltage is the analog of the physical variable. A device which converts a physical, mechanical or chemical quantity to a voltage is called a sensor.

Computers do not understand voltages: They understand bits. Bits are digital signals. A device which converts voltages to bits is an analog-to-digital converter.

Our AIM 16 (Analog Input Module) is a 16 input analog-to-digital converter.

The goal of Connecticut microComputer in designing the uMAC SYSTEMS is to produce easy to use, low cost data acquisition and control modules for small computers. These acquisition and control modules will include digital input sensing (e.g. switches), analog input sensing (e.g. temperature, humidity), digital output control (e.g. lamps, motors, alarms), and analog output control (e.g. X-Y plotters, or oscilloscopes).

Connectors



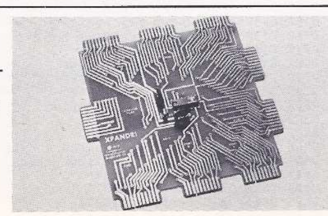
The AIM 16 requires connections to its input port (analog inputs) and its output port (computer interface). The ICON (Input CONnector) is a 20 pin, solder eyelet, edge connector for connecting inputs to each of the AIM16's 16 channels. The OCON (Output CONnector) is a 20 pin, solder eyelet edge connector for connecting the computer's input and output ports to the AIM16.

The MANMOD1 (MANifold MODule) replaces the ICON. It has screw terminals and barrier strips for all 16 inputs for connecting pots, joysticks, voltage sources, etc.

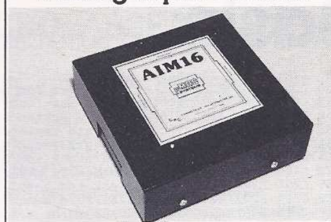
CABLE A24 (24 inch interconnect cable) has an interface connector on one end and an OCON equivalent on the other. This cable provides connections between the uMACSYSTEMS computer interfaces and the AIM 16 or XPANDR1 and between the XPANDR1 and up to eight AIM 16s.

XPANDR1

The XPANDR1 allows up to eight Input/Output modules to be connected to a computer at one time. The XPANDR1 is connected to the computer in place of the AIM16. Up to eight AIM16 modules are then connected to each of the eight ports provided using a CABLE A24 for each module. Power for the XPANDR1 is derived from the AIM16 connected to the first port.



Analog Input Module



The AIM 16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the uMAC SYSTEMS special interfaces.

The input voltage range is 0 to 5.12 volts. The input voltage is converted to a count between 0 and 255 (00 and FF hex). Resolution is 20 millivolts per count. Accuracy is 0.5% ± 1 bit. Conversion time is less than 100 microseconds per channel. All 16 channels can be scanned in less than 1.5 milliseconds.

Power requirements are 12 volts DC at 60 ma.

The POW1 is the power module for the AIM16. One POW1 supplies enough power for one AIM16, one MANMOD1, sixteen sensors, one XPANDR1 and one computer interface. The POW1 comes in an American version (POW1a) for 110 VAC and in a European version (POW1e) for 230 VAC.

TEMPSENS

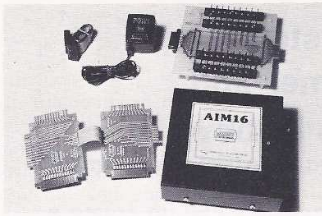
This module provides two temperature probes for use by the AIM16. This module should be used with the MANMOD1 for ease of hookup. The MANMOD1 will support up to 16 probes (eight TEMPSENS modules).

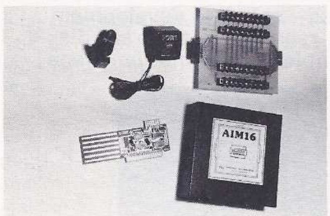
Resolution for each probe is 1°F.

CONTROL for PET, Apple, KIM, and AIM

μMAC SYSTEMS

Computer Interfaces and Sets





For your convenience the AIM16 comes as part of a number of sets. The minimum configuration for a usable system is the AIM16, one POW1, one ICON and one OCON. The AIM16 Starter Set 2 includes a MANMOD1 in place of the ICON. Both of these sets require that you have a hardware knowledge of your computer and of computer interfacing.

For simple plug compatible systems we also offer computer interfaces and sets for several home computers.

INTRODUCING
SUPER X-10 MODULE

Open a door or window and turn on a light, tape recorder, alarm!

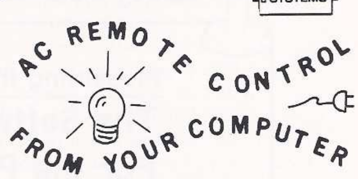
Control lab equipment. CLOSE THE LOOP on the real world.

AN INEXPENSIVE CONTROL SOLUTION FOR

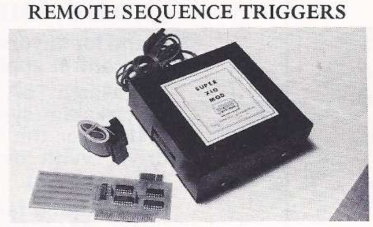
HOME SECURITY
ENERGY CONSERVATION
GREENHOUSES
ENVIRONMENTAL CONTROL
INDUSTRIAL CONTROL
LABORATORIES

SUPER X-10 MOD SPECS

1. Remote controller
   Controls up to 256 different remote devices by sending signals over the house wiring to remote modules. Uses BSR remote modules available all over the USA (Sears, Radio Shack, etc.). Does not require BSR control module. Does not use sonic link.
2. Clock/calendar
   Time of day - hours, minutes, seconds
   Date - month, day - automatically corrects for 28,29,30 and 31 day months.
   Day of the week.
3. Digital inputs/outputs
   8 inputs - TTL levels or switch closures. Can be used as triggers for stored sequences.
   8 outputs - TTL levels



PLUS: CLOCK, CALENDAR, REMOTE SEQUENCE TRIGGERS



4. Computer interfaces
   S-100: Requires one 8-bit input port and one 8-bit output port.
   Requires cable assembly.
   PET, APPLE, TRS-80, KIM, SYM, AIM65: Plug-in sets available - no cable assembly required.
   Other: same as S-100
5. Self-contained module in metal case with its own power supply. Physical size approximately 5X6X2.

Price (until April 30, 1980): $199.00 (S-100), $249.00 (other)

All prices and specifications subject to change without notice. Our 30-day money back guarantee applies.

AIM16 (16 channel-8 bit Analog Input Module)	179.00
POW1a (POWer module-110 VAC)	14.95
POW1e (POWer module-230 VAC)	24.95
ICON (Input CONnector)	9.95
OCON (Output CONnector)	9.95
MANMOD1 (MANifold MODule)	59.95
CABLE A24 (24 inch interconnect cable)	19.95
XPANDR1 (allows up to 8 Input or Output modules to be connected to a computer at one time)	59.95
TEMPSENS2P1 (two temperature probes, -10°F to 160°F)	49.95
LIGHTSENS1P1 (light level probe)	59.95

The following sets include one AIM16, one POW1, one OCON and one ICON.

AIM16 Starter Set 1a (110 VAC)	189.00
AIM16 Starter Set 1e (230 VAC)	199.00

The following sets include one AIM16, one POW1, one OCON and one MANMOD1.

AIM16 Starter Set 2a (110 VAC)	239.00
AIM16 Starter Set 2e (230 VAC)	249.00

The following modules plug into their respective computers and, when used with a CABLE A24, eliminate the need for custom wiring of the computer interface.

PETMOD (Commodore PET)	49.95
KIMMOD (KIM, SYM, AIM65)	39.95
APMOD (APPLE II)	59.95
TRS-80 MOD (Radio Shack TRS-80)	59.95

μMAC SYSTEMS

Order Form

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

The following sets include one AIM16, one POW1, one MANMOD1, one CABLE A24 and one computer interface module

PETSET1a (Commodore PET - 110 VAC)	295.00
PETSET1e (Commodore PET - 230 VAC)	305.00
KIMSET1a (KIM, SYM, AIM65 - 110 VAC)	285.00
KIMSET1e (KIM, SYM, AIM65 - 230 VAC)	295.00
APSET1a (APPLE II - 110 VAC)	295.00
APSET1e (APPLE II - 230 VAC)	305.00
TRS-80 SET1a (Radio Shack TRS-80 - 110 VAC)	295.00
TRS-80 SET1e (Radio Shack TRS-80 - 230 VAC)	305.00

QUANTITY	DESCRIPTION	PRICE	TOTAL
SUBTOTAL			
Handling and shipping — add per order			$3.00
Foreign orders add 10% for AIR postage			
add 7% sales tax			
TOTAL ENCLOSED			

NAME ____

COMPANY ____

ADDRESS ____

CITY ____

STATE ____ ZIP ____

VISA ○ M/C ○ Expiration date ____

Card number ____

Nuts and Volts

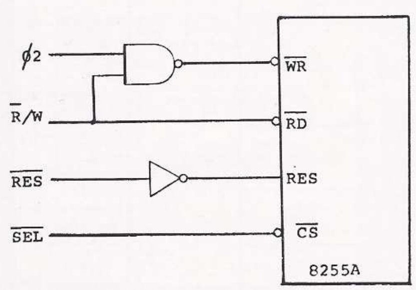
Gene Zumchak
1700 Niagara Street
Buffalo, N.Y. 14207

In the first N & V discussion, I talked about read/write timing in general, and 6502 timing in particular. Fast TTL chips can be used with the 6502, but so can most of the I/O chips of other processor families, provided all the timing requirements are resolved. Even chips with apparently incompatible timing requirements can usually be accomodated by using tricks like latching the write data, or shortening the write strobe, as discussed in the first column. Let's consider what is required to interface a popular port chip of the 8080 family.

The 8255A port chip has 24 I/0 pins, programmable in groups of four or eight bits as inputs and outputs. The ports can be used as simple ports, ports with handshaking (and interrupts) and even as bidirectional buses. The reader might want to dig up a spec sheet to study this versatile chip. The "A" suffix of the part number is important. The original 8255 (without the "A") had long set-up and hold time requirements. The 8255A, like newer Intel family chips, has improved timing specs with a 100 ns. set-up time and 30 ns. hold time, completely 6502 compatible.

The low-true read gate of the 8255A, RD, can be the inverted R/W signal which need not (but can be) gated with Ø2. The low-true write strobe, WR, is met by the normal 6502 write strobe, which we saw earlier is Ø2 NANDed with the inverted R/W line. A high true Reset signal must be provided. Like most peripherals, it has a low-true chip select. Figure 1. shows the connections which satisfy the 8255A's timing requirements.

Figure 1.



Interfacing 8255A Port

If you have an I/0 application requiring more than 16 pins, or you covet some other 8255A feature, there's no electronic reason why you cannot use this chip with your 6502 system. The same can be said of I/0 chips from other families. Clearly, all families are designed to be both voltage level and drive compatible with TTL and hence with each other. As we can see, accommodating the read/write timing need not be difficult.

Using Port Chips

The most commonly used family accessory chips are the I/0 port chips. However, when simple I/0 is required, port chips may not be the best choice. Family chips, including port chips, are not inexpensive. Port chips typically sell in the $8 to $15 range. Since they are MOS devices, their drive capability is usually just one TTL load. They are also vulnerable to static. Since their data bus lines also can only drive a single TTL load, their use is limited to the internal unbuffed data bus around the processor. One could interface them to a buffered bus with bidirectional buffers, but these buffers are expensive and power hungry. MOS port chips, therefore, are most attractive for use in small dedicated controllers, especially where power and parts count are important considerations.

In applications using a buffered bus, where simple I/0 is adequate, and where ruggedness and drive capability are important, TTL I/0 is more attractive, and usually much cheaper.

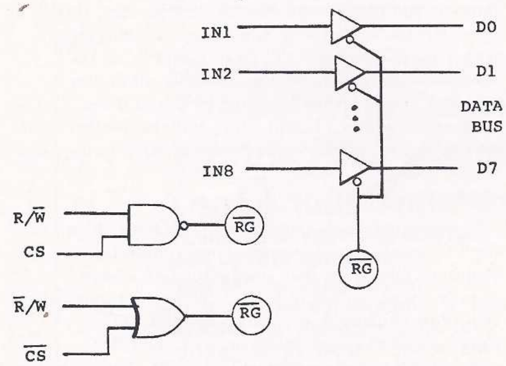
TTL Input

To make an input port, we need a set of tri-state® (® *trademark National Semiconductor)* gates which are gated in unison. A tri-state gate is an electronic switch. When enabled, the output reflects the input (sometimes with inversion). When disabled, the output is high impedance or floating. Thus a large number of tri-state outputs can be bussed together, provided that only one set or device is enabled at one time. RAM chips, ROM chips, and any other devices designed to attach directly to a bidirectional data bus have built in tri-state outputs.

TTL tri-state gates come in quad, hex, and octal configurations. Quad types like the 74LS125 have individual enables for each gate. Hex types like the 8T97, 74LS367, 8097 etc. have four gates with one enable, and two gates with one enable. Although octal gates are the most attractive for eight bit processors, the supply has not kept up with the demand, and hex types are a little easier to come by. Octal types 81LS97 (Nat.) and 74LS244 are not pin compatible.

All that's required to use some tri-state gates as an input port is a low-true read gate. This is obtained by ANDing of the R/W line in the read state, and a chip select decoded from the address lines. Figure 2. shows a couple of possibilities, depending upon the polarity of the chip select.

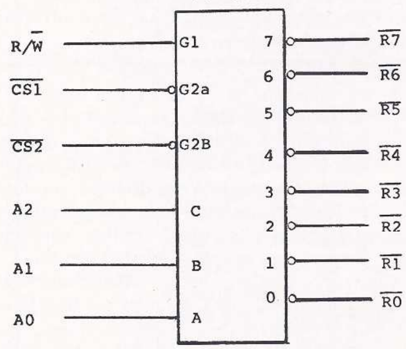
Figure 2.



TTL Input Port With Gating

If read gate signals are required for several ports, a single three to eight decoder chip can be used to get eight read gates from a coarser select. The R/W line is used as an enable and is internally gated with all the outputs, as shown in figure 3.

Figure 3
74LS138



Input Port Read Selects

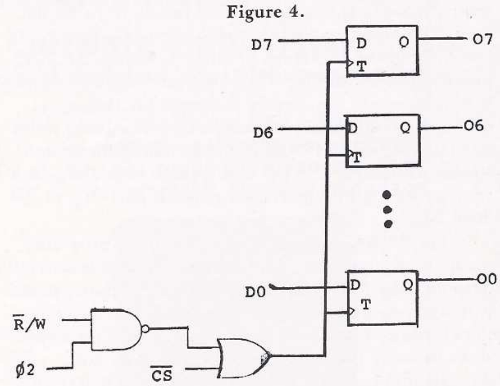
One nice feature of TTL tri-state gates is that they are always buffers and are meant to drive busses. Low power Schottky devices are more desireable and usually adequate for most applications.

TTL Output

An output bit is a flip-flop which can be written to and from the data bus and whose output is connected to the world. Output bits are usually "D" type flip-flops or latches. In TTL there are several configurations, duals, 74LS74, 74LS109; quads, 74LS75, 74LS175; hex, 74LS174; and octal, 74LS273, 74LS373, 74LS374 and others. Again octal types are sometimes a bit hard to come by.

Output ports need a write strobe generated by ANDing the general purpose write strobe with a select decoded from addresses. Figure 4. shows an output port and the necessary write strobe.

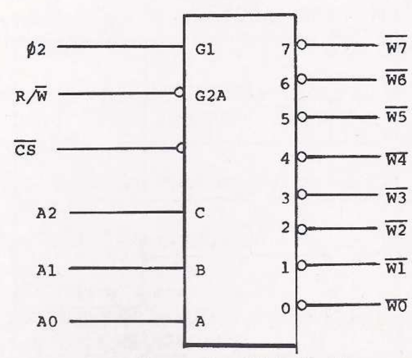
Figure 4.



Output Port With Write Strobe

Since TTL devices are very fast, they have set-up time requirements of only a few nanoseconds. Therefore the locking edge of the write strobe *better* come before the data goes away. That is, the Ø2 closest to the processor must be used, and not any delayed versions. With a little care, we can use a single decoder chip to generate write strobes for several ports as in Figure 5.

Figure 5.
74LS138



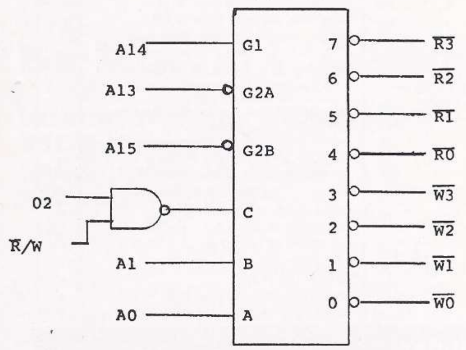
Output Port Write Strobes

From the data sheet for the 74LS138, we see that the delay from the high true enable input (G1) to any output is a maximum of 26 ns. (typically 17 ns.). This is quite acceptable, provided that we are not using a delayed Ø2.

Now if you are building a small dedicated controller, you certainly may not need eight input ports or eight output ports. There's no reason why you cannot use a single 74LS138 to give you gates and strobes for four of each.

Figure 6.

74LS138



Write Strobes and Read Gates

Figure 6 shows a 74LS138 wired to give four read gates and four write strobes. In a dedicated controller, you usually have memory space to burn so that you can afford to waste some. In figure 6. we apply address lines directly to the enables. This puts the ports in an 8K block of memory starting with $4000. The Nand gate generates the general purpose write strobe. It is applied to the "C" input of the decoder. When it is low, a write strobe is generated, when high, a read gate. The maximum delay through the NAND gate is 15 ns, through the decoder, a maximum of 26 ns. Thus Ø2 experiences a worst case delay of 41 ns. to the trailing edge of the write strobe. This would be acceptable even if there was no data bus buffer delay to compensate it.

Summary

Interfacing I/0 to an existing system or a do-it-yourself prototype is not difficult as long as you understand and consider read/write timing. Family chips from any family are useable. Some applications may favor family chips. Others may suggest TTL. The gates and strobes required by TTL I/0 are easy to generate.

In the next column I will talk about address decoding and generating selects. Please feel free to write and suggest hardware topics that you would like me to write about.

©

Programming & Interfacing the 6502, with Experiments,

by Marvin L. Dejong.

Howard W. Sams & Co., Inc.
4300 West 62nd St.
Indianapolis, IN 46268
414 pages, $13.95

Review by Jim Butterfield

This book might have been subtitled, ''A hands-on guide to the 6502.'' That's what it really is: it invites the owner of a KIM, SYM or AIM to learn the 6502 by working through example after example on his machine. Most of us learn by doing, rather than just by reading; and this book contains eighty carefully graded ''experiments'' that encourage you to get your hands on the machine and prove to yourself that it works the way the book says.

This is good stuff: the text and experiments are carefully graded and go at a gentle pace. You won't get very many advanced programming concepts here: the book covers only the basics. But it does a careful and thorough job. Early concepts are developed with care at a pace the beginner can cope with.

As the title suggests, the book comes in two parts. Part I deals with programming the 6502, Part II with interfacing. Each chapter begins with a statement of objectives, identifying what you may expect to learn there. Each chapter ends with a series of experiments designed to reinforce what you have learned. An experiment often takes the form: ''load this program .. now do this .. what do you see? .. can you explain why?''. Emphasis is on gaining understanding as to how a simple program operates; the last experiment or two in a chapter often suggest small projects for the reader.

Machine language is developed a few op-codes at a time. Loads, Stores, and Transfers are introduced first, and subsequent chapters progressively bring in more commands. Branches, for example, don't arrive until chapter six - I would rather have seen them a little earlier because I believe loops are so important - and the op codes aren't completely covered until chapter 9 has been completed. Advanced addressing modes, such as indexing and indirect addressing, are held back until chapter 8. It's all carefully graded, and the going is about as easy as it can be for machine language.

The pace changes in Part II, Interfacing the 6502. We're thrown quite abruptly into the hardware field: logic diagrams, truth tables, timing charts and oscilloscope traces start to appear with great rapidity. The author seems to assume that the reader will have some understanding of hardware, which is likely true for a sizable fraction of KIM/SYM/AIM owners. A beginner who isn't sure about the different shapes of AND and NOR logic symbols will have to work hard.

In keeping with the accelerated pace of the material, Part II takes on a number of more ambitious projects, some of which might prove to be of special interest to readers. Music synthesis, an ASCII input port, data logging, a morse keyer, and a lunar occultation program are included; most are adapted from other sources but are accompanied by extra explanations.

The book contains a quite extensive appendix section, with emphasis on hardware. Many of the data sheets are printed in very fine type and may be hard to read. An index is included.

Is it possible to write a book which deals with three different machines--the KIM, SYM, and AIM? The experiments jump around from one machine to the other without always specifying which machine is intended. Even so, most users will be able to sort it out without too much trouble. Two key tables point out vital addresses on the respective machines; readers may find themselves repeatedly scrambling for page 44 or 55 - I wish that these had been printed on fold-out sheets so that they could be visible during the experiments.

The author deals carefully with difficult subjects; he doesn't gloss over the tricky parts but treats them with precision. One thing, however, bothers me: his notation for immediate-mode addressing. If you want to load the A register with the value 12 decimal, any of the following may be used on most assemblers:

```
LDA #12
LDA #$0C
LDA #%00001100
```

.. you may code the number in binary, decimal, hexadecimal or whatever, but you must include that pounds sign (#) to indicate Immediate mode. The author codes LDA $0C; most assemblers would take this to mean, ''load the contents of address 12'' - not the value 12. Readers will have to re-adapt when they start using an assembler.

The book is a good, gentle introduction to programming the 6502. It's a little harder going for inter-facing, especially for hardware beginners.

The ''hands-on'' nature of the experiments tend to drive the lessons home. It's a good way to come to grips with your computer.

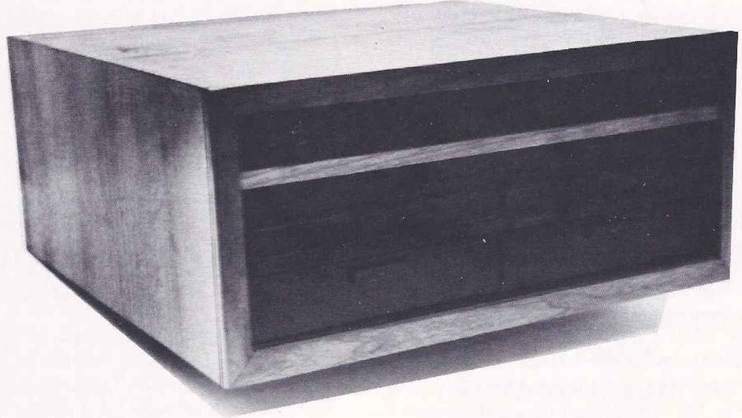
©

HDE inc.

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

HUDSON DIGITAL ELECTRONICS INC.

ANNOUNCING THE HDE OMNIDISK 65/8



Now, you can "plug in" the latest in a successful series of flexible disk systems developed by HDE for the KIM, SYM and AIM microcomputers. The OMNIDISK 65/8 is a complete system, using 8 inch soft sectored diskettes with a formatted (IBM Standard) capacity of 256K. Of course, a disk formatting function is included as are system supporting utilities for file renaming, disk packing, copy (dual systems) and others.

TED, a full featured, line oriented editor is standard in KIM and SYM based versions to get you up and running on your project in a hurry. The AIM version uses the on-board editor. With the OMNIDISK 65/8 you can concentrate on your problem, the disk supports you all the way.

OMNIDISK 65/8 is available in an attractive walnut wood cabinet, or unpackaged for OEM applications in dual and single drive configurations. The HDE disk controller is a state-of-the-art 4½" by 6½" card electronically compatible with the 44-pin KIM-4 bus structure. The controller and disk-driver are designed to operate with the popular Shugart 801-R and compatible devices.

The OEM single drive is $1195, the dual, $1895 and the dual in the walnut cabinet, $2200. Price is another reason to step up to the proven quality of an HDE system.

HDE PRODUCTS – BUILT TO BE USED WITH CONFIDENCE
AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

JOHNSON COMPUTER
Box 523
Medina, Ohio 44256
216-725-4560

ARESCO
P.O. Box 43
Audubon, Pa. 19407
215-631-9052

PLAINSMAN MICROSYSTEMS
Box 1712
Auburn, Ala. 36830
800-633-8724

LONE STAR ELECTRONICS
Box 488
Manchaca, Texas 78652
612-282-3570

PERRY PERIPHERALS
P.O. Box 924
Miller Place, N.Y. 11764
516-744-6462

Fast Random Numbers For The 6502

Edward B. Beach
5112 Williamsburg Blvd.
Arlington, VA 22207

If you should ever have need for a short (two instructions), fast (10 us) routine to produce an 8-bit random number, read on! Notice we said *random* number, not *pseudorandom* number.

Most random number generators (pseudorandom, that is) depend upon complex mathematical theorems to produce algorithms which depend upon 16, 24, 32, or perhaps even 40-bit numbers which act as a seed to produce a sequence of numbers that will repeat after some astronomically large number of iterations. The net result is a routine that takes up lots of room in memory (both for the random number generator routine and for the multibyte seed number), and which takes a fair amount of computer time to execute. In addition, many of these algorithms have seed values which are not allowed as they will cause the generator to lock up, causing the generator to produce some constant rather than a random number. The only way to restart the generator is to insert a new seed number.

The one real virtue of this type of random number generator is that a given starting value will always produce the same sequence of random numbers. This makes these routines useful in generating hash addresses and the like, so you can search a hash table over and over by properly priming the seed number.

I'm not into all that sort of exotic stuff yet. My needs for random numbers are almost trivial. I need them mostly for games, for rolling dice, dealing cards, producing tones for computer "music", and for getting random Morse code characters for code practice. Most times an 8-bit number is enough. For these needs my Fast Random Number Sequencer (FRNS) is ideally suited.

Here's how it works.

First of all we need two locations in page zero for an address pointer. I call these RND and RND + 1. Only one of these locations needs to be initialized. This location (RND + 1) will be set to point to one specific page in your computer memory. In my KIM-1 I usually use page $18, which is the beginning of the monitor ROM. Other times I set RND + 1 to page zero. Whenever I need an 8-bit random number (in the accumulator) I insert the following two instructions:

```
INC RND
SBC (RND),Y
```

This is the entire FRNS!

1) The value in A as FRNS is entered.
2) The value in Y as FRNS is entered.
3) The value in RND as FRNS is entered.
4) The condition of the carry bit as FRNS is entered.

The FRNS does not alter or use any registers other than the accumulator. The carry may or may not be altered as a result of the SBC instruction. If it is important to preserve the condition of the carry, you can precede FRNS with PHP (3 uS) and follow it with PLP (4 uS). This adds two bytes and 7 uS to the routine.

As you can see, FRNS requires four bytes of code for the whole thing - one byte more than a JSR instruction alone. This means that in most cases it will probably be most efficient to use FRNS as a macro, coding it in-line every time you need a random number. You would have to call it as a subroutine from at least six different places in one program before it would become more economical (memory-wise) to code it as a subroutine. In addition, the time overhead of each JSR/RTS is 12 uS - more than the execution time of the two-instruction FRNS itself!

In selecting a page number to use in RND + 1, keep in mind that almost two full pages will be addressed using indirect indexed addressing. Using page $18 as an example, you will be selecting locations which could range from $1800 to $18FF + $FF = $19FE. Both pages should contain SOMETHING. That's why I frequently use $00 in RND + 1 since page zero usually has lots of temporary data and page $01 is the stack which is also quite active, adding another degree of randomness to the FRNS.

If you like to experiment, there are variations you can use for FRNS like:

```
DEC RND
SBC (RND),Y
```

or:

```
INC RND
ADC (RND),Y
```

or perhaps you might like:

```
SBC (RND),Y
INC RND
```

or:

```
ADC (RND),Y
DEC RND
```

or some such. The results are equally unpredictable. After all, that's what random number generators should be.

Reprinted by permission of the Washington Area KIM Enthusiasts (W.A.K.E.), Ted Beach, President.

SPEEDING UP FOCAL

Jim Knopf
12021 S.E. 51 Street
Bellevue, Wa 98006

Focal-65E is the best thing I ever did for my Kim. It's a tremendously powerful language packed into 5 1/2 K of code and I've yet to find a Basic program that I can't convert to Focal.

Most programs run plenty fast, but some of the "number crunchers", such as 3-D Tic Tac Toe, Connect-4, Mastermind, and my Maze program were embarassingly slow.

Speeding up the Focal interpreter by about 20% (see issue no. 15 of the 6502 User Notes) helped a little, but not enough. I've discovered that with a few simple changes in the programs themselves, most programs will run 2 to 3 times faster, and the changes are often trivial.

Hint #1: Pre-allocate the frequently used variables. This is easier than it sounds. Take a guess at which few variables in the program are used most frequently. These are the variables used inside of programming loops and in frequently used subroutines. When you have settled on at least five or six of them, insert a statement which will be executed very early in the program, right after the initial "erase" statement, to initialize these variables. For example:

```
1.01 E;S N=M=X=I=S=0
```

Focal builds its list of variables sequentially, front to back, in sequence by "when referenced". Thus, the above example will cause N,M,X,I, and S to be placed first in the list. Each time a variable is referenced, Focal starts at the front of the list and scans sequentially until it finds the required variable. If it doesn't find it, it creates it at the end of the list. You guarantee that Focal will find this data quickly by placing it at the front of the variable list.

Hint #2: Repackage the program for fast branching. This one takes more work, but can also bring big benefits. When Focal needs to "Go" to or "Do" a statement number, it uses the following logic:

A. If the "branched to" statement number is *larger* than the "current" line number, it starts at the "current" line number and scans through the program, byte by byte, moving down the program until it finds the new line.

B. If the "branched to" statement number is *smaller* than the "current" line number, it starts scanning at the front of the program and goes until it finds the required line.

You will save Focal a lot of time if you will place frequently used routines where Focal can find them quickly, and move seldom used code out of the way so that Focal doesn't have to keep scanning past it.

1. Move the "startup" code to the tail end of the program.
2. Move the small, highly used subroutines to the front of the program, or to a point right after the spots they are referenced.

I try to structure my programs, for speed, like the example below:

```
1.01 G 99.1
2.10 (highly used subroutines)
10.10 (mainline logic)
80.10 (less frequently used subroutines)
99.10 E;S N = M = X = I = S = 0
    (startup and initialization code)
99.90 G 10.1
```

©

Part 2: RS232 COMMUNICATIONS

Michael E. Day
2590 DeBok Road
West Linn, OR 97068

In part 1, Michael described the full RS-232C standard with associated communications details. Here he provides expanded descriptions of RS-232C signals. RCL

The following is a list of the definitions of the RS232-C signals which are listed in order of pin number. To simplify the definitions, the transmitter of the message will be identified as the "transmitting terminal" and the receiver as the "receiving terminal."

PROTECTIVE GROUND PIN 1: This ground is electrically connected to the equipment frame. It may be connected to external grounds, as required.

TRANSMITTED DATA PIN 2: This signal is generated by the transmitting terminal and is transferred to the local transmitting data set for transmission of data to the receiving terminal. The transmitting terminal will hold "Transmitted Data" in marking condition during the intervals between characters or words, and at all times when no data are being transmitted.

In all systems, the transmitting terminal will not transmit data unless an ON condition is present on all of the following four signals:

1. Request To Send
2. Clear to Send
3. Data Set Ready
4. Data Terminal Ready

RECEIVED DATA PIN 3: This signal is generated by the receiving data set in response to data signals received from transmitting terminal via the transmitting data set. "Received Data" will be held in the binary one (marking) condition at all times when "Received Line Signal Detector" is in the OFF condition. This is called clamping the line.

On a half-duplex channel, "Received Data" signal will be held in the binary one (marking) condition when "Request To Send" is in the ON condition and for a brief interval following the ON to OFF transition of "Request To Send" signal to allow for the completion of transmission and the decay of line reflections. This is called squelch.

REQUEST TO SEND PIN 4: This signal is used to condition the data set for data transmission. On simplex channels or duplex channels, the ON condition maintains the data set in the transmit mode. The OFF condition maintains the data set in a non-transmit mode.

On a half-duplex channel, the ON condition maintains the data set in the transmit mode and inhibits the received mode. The OFF condition maintains the data set in the receive mode.

A transition from OFF to On instructs the data set to enter the transmit state which turns on the carrier. The data set responds by taking such action as may be necessary and indicates completion of such actions by turning ON "Clear To Send," thereby indicating to the terminal that data may be transferred on the interchange signal "Transmitted Data."

A transition from ON to OFF instructs the data set to complete the transmission of all data which was previously transferred on the interchange signal "Transmitted Data" and then assumes a non-transit mode or a receive mode, as appropriate. The data set responds to this instruction by turning OFF "Clear To Send" when it is prepared to again respond to a subsequent ON condition "Request To Send."

When "Request To Send" is turned OFF, it will not be turned ON again until circuit "Clear To Send" has been turned OFF by the data set.

An ON condition is required on "Request To Send" as well as on "Clear To Send," "Data Set Ready" and, where implemented, "Data Terminal Ready" whenever the transmitting terminal transfer data on the interchange signal "Transmitted Data."

It is permissible to turn "Request To Send" ON at any time when "Clear To Send" is OFF, regardless of the condition of any other interchange circuit.

CLEAR TO SEND PIN 5: A signal generated by the data set to indicate whether or not the data set is ready to transmit data.

The "Clear To Send" ON condition together with the ON condition of interchange signals "Request To Send," "Data Set Ready" and, where implemented, "Data Terminal Ready" will be transmitted to the communication.

The OFF condition is an indication to the transmitting terminal that it should not transfer data across the interface on interchange "Transmitted Data."

The ON condition of "Clear To Send" is a response to the occurrence of a simultaneous ON condition on "Data Set Ready" and "Request To Send" delayed as may be appropriate to the data set for establishing a data communication channel to a remote terminal (including the removal of the MARK HOLD clamps from the received data interchange circuit of the remote data set).

Where "Request To Send" is not implemented in the data set with transmitting capability, "Request To Send" shall be assumed to be in the ON condition at all times and "Clear To Send" will respond accordingly.

DATA SET READY PIN 6: This signal is used to indicate the status of the local data set. The ON condition of this signal is presented to indicate that the local data set is not in test (local or remote), talk (alternate voice), or dial mode (optional).

This circuit shall be used only to indicate the status of the local data set. The ON condition will not be interpreted as either an indication that a communication channel has been established to a remote terminal or the status of any remote terminal.

The OFF condition will appear at all other times and shall be an indication that the local is to disregard signals appearing on any other interchange signal with the exception of "Ring Indicator." The OFF condition will not impair the operation of "Ring Indicator" or "Data Terminal Ready."

SIGNAL GROUND PIN 7: This conductor establishes the common ground reference potential for all interchange signals except "Protective Ground." Within the data set, this circuit is brought to one point, and it is possible to connect this point to "Protected Ground" by means of a wire strap inside the equipment. The wire strap can be connected or removed at installation, as may be required to meet applicable regulations or to minimize the introduction of noise into electronic circuitry.

RECEIVED LINE SIGNAL DETECTOR PIN 8: The ON condition of this signal is presented when the data set is receiving a signal which meets its suitability criteria. These criteria are established by the data set manufacturer.

The OFF condition indicates that no signal is being received or that the received signal is unsuitable for demodulation.

The OFF condition of "Received Line Signal Detector" will cause "Received Data" to be clamped to the binary one (marking) condition.

The indications on this circuit follow the actual onset or loss of signal by appropriate guard delays.

DATA SET TESTING PINS 9 and 10: These pins are reserved for testing the data set and will not be used as a normal part of the interface.

PIN 11-UNASSIGNED: This pin may be used by the manufacturer for any purpose desired.

SECONDARY RECEIVED LINE SIGNAL DETECTOR PIN 12: This signal is equivalent to "Received Line Signal Detector" except that it indicates the proper reception of the secondary channel line signal instead of indicating the proper reception of a primary channel received line signal.

Where the secondary channel is useable only as a circuit assurance or an interrupt channel (see Secondary Request To Send) "Secondary Received Line Detector" will be used to indicate the circuit assurance status or to signal the interrupt. The ON condition will indicate circuit assurance or a non-interrupt condition. The OFF condition will indicate circuit failure (no assurance) or the interrupt condition.

SECONDARY CLEAR TO SEND PIN 13: This signal is equivalent to "Clear To Send," except that it indicates the availability of the secondary channel instead of indicating the availability of the primary channel. This circuit is not provided where the secondary channel is useable only as a circuit assurance or an interrupt channel.

SECONDARY TRANSMITTED DATA PIN 14: This signal is equivalent to "Transmitted Data" except that it is used to transmit data via the secondary channel.

Signals on this circuit are generated by the data terminal equipment and are connected to the local secondary channel transmitting data set for transmission of data to remote terminal.

The transmitting terminal shall hold "Secondary Transmitted Data" in marking condition during intervals between characters or words and at all times when no data is being transmitted.

In all systems, the data set shall not transmit data on the secondary channel unless an ON condition is present on all of the following four circuits, where implemented:

1. Secondary Request to Send
2. Secondary Clear to Send
3. Data Set Ready
4. Data Terminal Ready

All Data signals that are transmitted across the interface on interchange "Secondary Transmitted Data" during the time when the above conditions are satisfied will be transmitted to the communication channel.

When the secondary channel is useable only for circuit assurance or to interrupt the flow of data in the primary channel, "Secondary Transmitted Data" is normally not provided, and the channel carrier is turned ON or OFF by means of "Secondary Request To Send." Carrier. OFF is interrupted as an interrupt condition.

TRANSMISSION SIGNAL ELEMENT TIMING PIN 15: Signals on this circuit are used to provide the data terminal equipment with signal element timing information.

The data terminal equipment will provide a data signal on "Transmitted Data" in which the transitions between signal elements nominally occur at the time of the transitions from OFF to ON condition of the signal on "Transmission Signal Element Timing." When "Transmission Signal Element Timing" is implemented in the terminal, the terminal shall normally provide timing information on this circuit whenever the terminal is in a POWER ON position. It is permissible for the terminal to withhold timing information on this circuit for short periods provided "Data Set Ready" is in the OFF condition. (For example, the withholding of timing information may be necessary in performing maintenance tests within the terminal).

SECONDARY RECEIVED DATA PIN 16: This circuit is equivalent to "Received Data" except that it is used to receive data on the secondary channel.

When the secondary channel is useable only for circuit assurance or to interrupt the flow of data in the primary channel, "Secondary Received Data" is normally not provided. See interchange "Secondary Received Line Signal Detector."

RECEIVER SIGNAL ELEMENT TIMING PIN 17: Signals on this circuit are used to provide the terminal with received signal element timing information. The transition from ON to OFF condition shall normally indicate the center of each signal element "Received Data." Timing information on "Receiver Signal Element Timing" shall be provided at all times when circuit "Received Line Signal Detector," is in the ON condition. It may, but need not, be present following the ON to OFF transition of "Received Line Signal Detector."

UNASSIGNED PIN 18: This pin may be used by the manufacturer for any purpose desired.

SECONDARY REQUEST TO SEND PIN 19: This signal is equivalent to "Request To Send" except that it requests the establishment of the secondary channel instead of requesting the establishment of the primary data channel.

Where the secondary channel is used as a backward channel, the ON condition of "Request To Send" will disable "Secondary Request To Send" and it will not be possible to condition the secondary channel transmitting data set to transmit during any time interval when the primary channel transmitting data set is so conditioned. Where system considerations dictate that one or the other of the two channels be in transmit mode at all times but never simultaneously, this can be accomplished by permanently applying an ON condition to "Secondary Request To Send" and controlling both the primary and secondary channels, in complementary fashion, by means of "Request To Send." Alternatively, in this case, "Secondary Clear To Send" need not be implemented in the interface.

When the secondary channel is useable only for circuit assurance or to interrupt the flow of data in the primary data channel, "Secondary Request To Send" will serve to turn ON the secondary channel carrier. The OFF condition of "Secondary Request To Send" will turn OFF the secondary channel carrier and thereby signal an interrupt condition at the remote end of the communication channel.

DATA TERMINAL READY PIN 20: This signal is used to control switching of the data set to the communication channel. The ON condition prepares the data set to be connected to the communication channel.

SIGNAL QUALITY DETECTOR PIN 21: Signals on this circuit are used to indicate whether or not there is a high probability of an error in the received data.

As ON condition is maintained whenever there is no reason to believe that an error has occurred.

An OFF condition indicates that there is a high probability of an error. It may, in some instances, be used to call automatically for the retransmission of the previously transmitted data signal. Preferably the response of this circuit shall be such as to permit identification of individual questionable signal elements on "Received Data."

RING INDICATOR (CE) PIN 22: The ON condition of this signal indicates that a ringing signal is being received on the communication channel.

DATA SIGNAL RATE SELECTOR PIN 23: Signals on this circuit are used to select between the two data signaling rates in the case of dual rate synchronous data sets or the two ranges of data signaling rates in the case of dual range non-synchronous data sets.

An ON condition shall select the higher data signaling rate or range of rates.

The rate of timing signals, if included in the interface, shall be controlled by this circuit as may be appropriate.

TRANSMIT SIGNAL ELEMENT PIN 24: Signals on this circuit are used to provide the transmitting data set with signal element timing information.

The ON to OFF transition shall nominally indicate the center of each signal element on "Transmitted Data." When "Transmit Signal Element Timing" is implemented in the data set, the data set shall normally provide timing information on "Transmit Signal Element Timing" whenever the data set is in a power on condition. It is permissible for the data set to withhold timing information on this signal for short periods provided "Request To Send" is in the OFF condition.

UNASSIGNED PIN 25: This pin may be used by the manufacturer for any purpose desired.

Although the "EIA" publishes an interface standard, some data set manufacturers do not conform to the standard in all cases. *CHECK* the specifications on each data set to determine which signals are on each pin. ©

A 6502 Multiprocessor System

Michael E. Day
2590 DeBok Road
West Linn, OR 97068

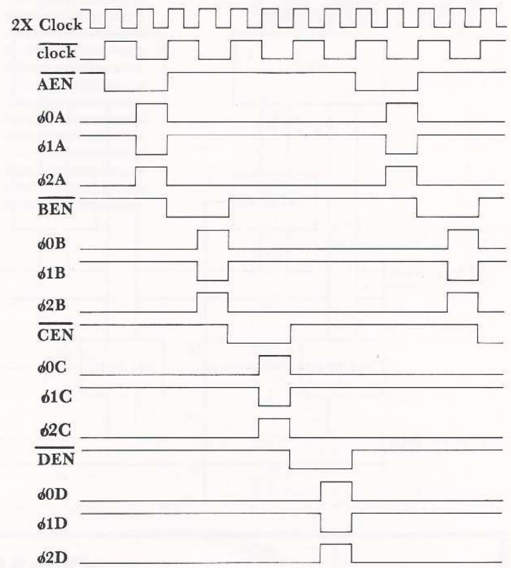
One of the problems encountered when attempting to use the 6502 in a multiprocessor environment is that a large amount of system control must be added when using the standard methods. An optional method, which allows the system integrity to be retained, is to syncronously switch the clocks to the processors. This is a better way than manipulation of the ready line in a system that requires the full CPU cycle to occur intact, as is sometimes the case with dynamic memory. Also (for the same reason) half cycle stealing can be a problem.

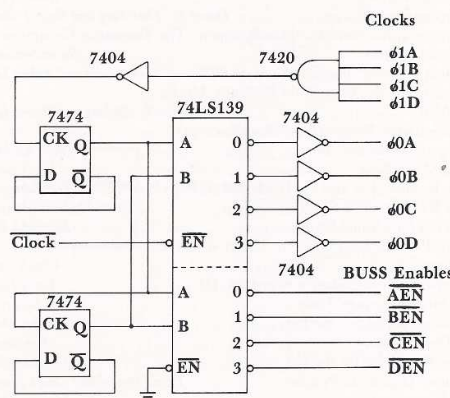
Cycle switching can be done with a minimum of components. Since clock shut down occurs during phase one clock the data buss of the 6502 is in the high impedance state which allows the data busses of the processors to be tied together. Since the address busses don't tri-state however, they must be isolated with tri-state buffers or multiplexers, as must the control lines. Although the number of processors that may be parallelled is theoretically unlimited, the physical and operational limit is ten. The method used to switch between the various processors is dependent upon the number of processors and requirements of the system.

The system shown allows four 6502's to share the buss. By using the leading edge of the phase one clock which signifies the end of one cycle, and the beginning of the next, syncronization with each processor can be maintained in selecting processors.

It should be noted that the complete address buss does not need to be multiplexed, and in many systems it is not desirable. In the average system the *top* 16K of address is used by each processor for its system control ROM. The *first* 16K of address is used by each processor for its work space RAM. This leaves the *second* and *third* 16K blocks available for us by all processors. All systems require some kind of IO operation to talk to the outside world; in the 6502 IO is memory mapped. In the 6502, memory mapped IO normally is placed in the third 16K address block. This leaves the second 16K address block open for interprocessor communications from the IO processor. The above example is a generalization only, and accual address allocation can be manipulated to fit each systems needs.

6502 Multiprocessor Timing





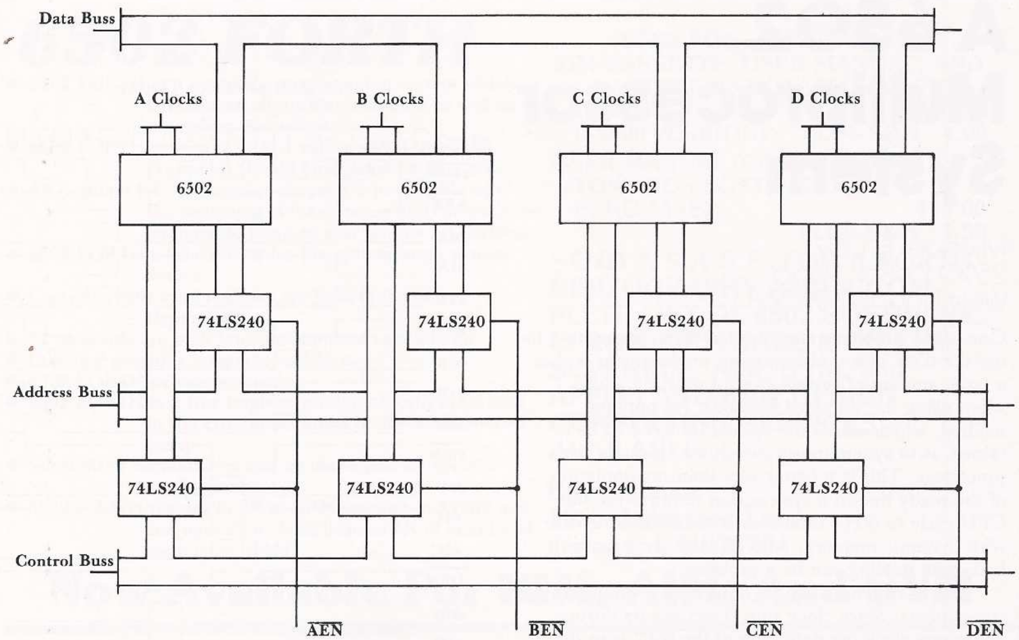


Table of Contents

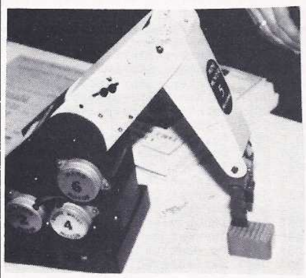
Basic Memory Map (Page O)

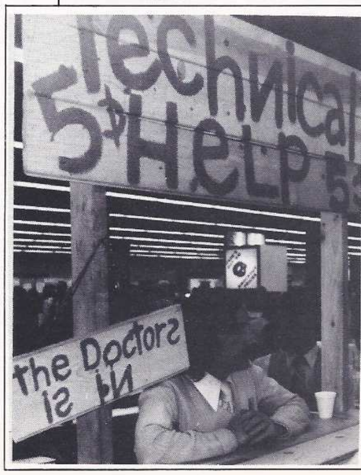
Compiled by Jim Butterfield

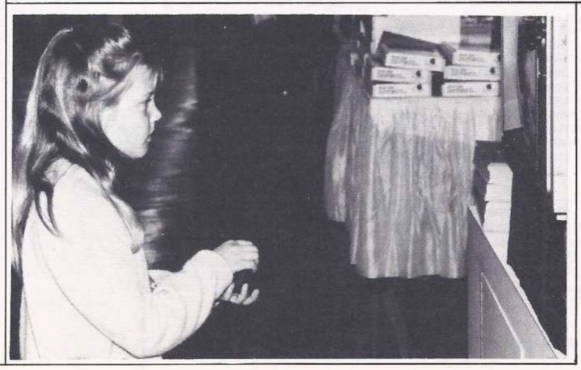
OSI is C2-4P. There may be differences in particular implementations of Basic.

KIM	SYM	AIM	OSI	Description
0000		0000		New-line jump
0003	000A	0003		USR jump
0006	0008	B006		Vector to 'fixed-to-floating' subroutine
0008	0006	B008		Vector to 'floating-to-fixed' subroutine
000A	000D	0006	005B	Search character
000B	000E	0007	005C	Scan-between-quotes flag
000C	000F	0008	005D	Input buffer pointer; # of subscripts
000D	0010	0009	005E	Defaault DIM flag
000E	0011	000A	005F	Type: FF=string, 00=numeric
000F	0012	000B		Type: 80=integer, 00=floating point
0010	0013	000C	0060	Flag: DATA scan; LIST quote; memory
0011	0014	000D	0061	Subscript flag; FNX flag
0012	0015	000E	0062	0=INPUT; $40=GET; $98=READ
0013	0016	000F	0063	Comparison Evaluation flag
0014	0017	0010	0064	Input flag (suppress output)
0016	0019	0011	000E	Position on print line
0017	001A	0012	000F	Maximum print line width
0018	001B	0013	0010	Input column limit
0019	001C	0014	0011	Integer value (for GOTO etc)
001B	001E	0016	0013	Start of input buffer
0062	0065	005D	005A	End of input buffer
0063	0066	005E	0065	Pointers for descriptor stack
0066	0069	0061	0068	Descriptor stack(temp strings)
006E	0071	0069	0070	End of Descriptor stack
006F	0072	0072	0071	Utility pointeer area
0073	0076	006E	0075	Product area for multiplication
0078	007B	0073	0079	Pointer: Start-of-Basic
007A	007D	0075	007B	Pointer: Start-of-Variables
007C	007E	0077	007D	Pointer: Start-of-Arrays
007E	0081	0079	007F	Pointer: End-of-Arrays
0080	0083	007B	0081	Pointer: String-storage(moving down)
0082	0085	007D	0083	Utility string pointer
0084	0087	007F	0085	Pointer: Limit-of-memory
0086	0089	0081	0087	Current Basic line number
0088	008B	0083	0089	Previous Basic line number
008A	008D	0085	008B	Pointer: Basic statement for CONT
008C	008F	0087	008D	Current DATA line number
008E	0091	0089	008F	Current DATA address
0090	0093	008B	0091	Input vector
0092	0095	008D	0093	Current variable name
0094	0097	008F	0095	Current variable address
0096	0099	0091	0097	Variable pointer for FOR/NEXT
0098	009B	0093	0099	Start of work area, pointers, etc
00A1	00A4	009C	00A1	Jump vector for functions
00A4	00A7	009F	00A4	Misc numeric work area
00AE	00B1	00A9	00AC	Accum#1: Exponent
00AF	00B2	00AA	00AD	Accum#1: Mantissa
00B3	00B6	00AE	00B0	Accum#1: Sign
00B4	00B7	00AF	00B1	Series evaluation constant pointer
00B5	00B8	00B0	00B2	Accum#1 hi-ordeer (overflow)
00B6	00B9	00B1	00B3	Accum#2: Exponent, etc.
00BC	00BF	00B7	00B8	Sign comparison, Acc#1 vs #2
00BD	00C0	00B8	00B9	Accum#1 lo-order (rounding)
00BE	00C1	00B9	00BA	Series pointer
	00C3	00BB		Error jump
	00C6			SAVE jump
	00C9			LOAD jump
00C0	00CC	00BF	00BC	CHRGET subroutine; get Basic char
00C6	00D2	00C5	00C2	Sub entry: get prev character
00C7	00D3	00C6	00C3	Basic pointer (within subrtn)
00D8	00E4	00D7	00D4	Random number seed.

©







Photos by Eric Rehnke. COMPUTE persons at the show (L-R): Larry Isaacs, Brenda Rehnke (helping staff the booth), Joretta Klepfer, Robert Lock. Please note that Eric did a much better job of photographing his wife than me. RCL

The Single-Board 6502

Eric Rehnke

The 5th West Coast Computer Faire was FANTASTIC!!! Besides having the chance to meet a number of you, I got a real good look at the latest developments in the small computer industry. I am very excited with what's happening.

Everything is becoming increasingly sophisticated. Music, graphics, interface devices, software, applications. . .and on and on.

Graphics seemed to be one dominating theme of the show. Everywhere you looked was evidence on the fact. New and lower cost graphics peripherals were introduced. Two drum plotters for under $700, a graphics input device for $200, sophisticated 3-D software for the Atari machines, graphics animation on the Apple, the list goes on.

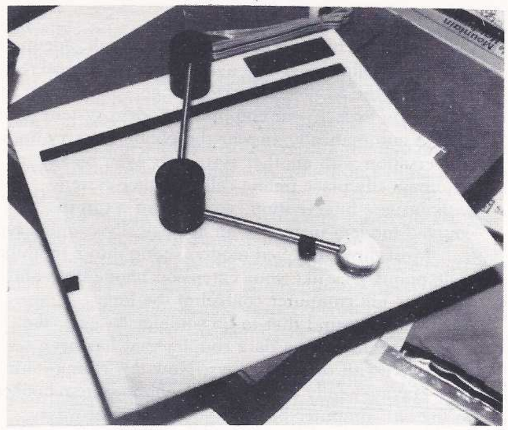
Telecommunications is another area of the industry that is expanding greatly. This is an area which I am particularly interested in because of the fact that as a society, we will be facing an increasing need to replace fossil-fuel burning transportation with energy and time efficient communication. The office of the future will more than likely be in the home for people who can interact with their jobs through a low-cost computer terminal and a modem.

We as computer hobbyists will have much to do with the future of tele-computing. We're the pioneers . . . .

Basically, there are two broad types of information systems accessible today with low-cost equipment. The decentralized type of system includes PCNET (Personal Computer Network), CBBS (Community Bulletin Board Systems) and the like. These systems are fairly casual, since they're more than likely run by hobbyists, have no access charges, and are, at the very least, excellent ways to become familiarized with computer "networking".

The other, more centralized, approach is that taken by The Source and Micronet (to name two). These outfits have large computers with access to very large data bases and many other services available. You can write programs in many of your favorite languages (BASIC, COBOL, FORTRAN, APL, RPG), have access to such things as the UPI General wire service, stock exchange quotes, backgammon, bridge, travel club, a buying service, file generators, editors, Star Trek and Football. On one service you can even download complete programs to your Apple, Pet or TRS-80 (how'd that one get in this column?). Anyhow, all kinds of stuff.

All you need to access this myriad of service is a 300 baud terminal and modem. But, to get the full benefit of all the services, you should also have a microcomputer on your end of the phone line.



For low-cost digital input (about $200), how about this? Your Apple (or whatever) simply reads the position of the two pots which are mounted in the pivot points to compute the position of the arm. Clever, huh???

Of course, with these large centralized information systems, you have access charges, passwords and the need of a plastic bank credit card to get into the system in the first place. Small price to pay for a little piece of the future, though. Beats the hell outa' the BOOBTUBE!

Getting Hooked Up

Presumably, you already have a computer and a terminal (or a computer with a built-in CRT) and are looking for a modem. The minimum modem necessary will be an originate only, acoustically coupled style capable of handling the BELL 103 standard modem protocol (300 baud). This will permit access to the centralized information system and the hobbyist bulletin board service but will not allow communication with other hobbyists that have orginate-only modems.

You see, for modem systems to communicate with each other, certain conventions must be adhered to. The most important of these states that the system that originates the phone call has to be in the "orginate" mode while the system answering the call should be in the "answer" mode. This originate/answer mode business has to do with the set of frequencies that's used to send the data and need not concern us here except to realize that to be able to receive calls as well as place them, you need both modes (orginate and answer) in your modem system.

Now modems can couple up to the telephone line in two ways: acoustically and directly.

With an acoustically coupled modem, you must usually place the telephone call manually and put the telephone handset into rubber cups on the modem when the telephone call is connected.

This type of modem is easiest to install, adequate for most applications, and available from several sources in the $150--$200 price range.

If you expect your computer/modem system to be able to automatically answer the phone to carry on a conversation with another system or even be able to automatically place phone calls to other systems without user intervention, you'll want a direct-coupled modem instead of an acoustically-coupled type.

Most direct-coupled modems plug into a modular style phone jack like your extension phone does and allow for full computer control of the line.

Keep in mind that to be completely legal, the modem MUST use a data coupler that has been registered with the FCC guys. Now *that's* important.

Having a fully automatic telephone system hooked to the old computer benefits you in several ways. First, you can take messages from other systems all day long while you're at work or out playing golf (of course, this presumes you have enough friends to make it all worthwhile). And secondly, your computer can place calls to your friendly local (or long distance) data base very late at night to take advantage of low activity and/or cheaper phone rates. You could even download the complete UPI news service to your disk so you can enjoy the up-to-the-minute news with your coffee in the morning. Since the data stream is happening at 300 baud, your computer could sit and scan for key words-picking out only what you're interested in reading about. Quite a bit more efficient than the newspaper. Wouldn't you say?

Anyhow, there are three modem manufacturers which seem interested in supporting the hobby/personal computer market. They are

U. S. Robotics Inc.
1035 W. Lake St.
Chicago, IL 60607
(312) 733-0497

NOVATION Inc.
18664 Oxnard St.
Tarzana, Ca 91356
(213) 996-5060

TNW Corp.
5924 Quiet Slope Dr.
San Diego, Ca 92129
(714) 225-1040

(TNW modem useable only with PET or other IEEE Bus computer)

There are other companies making modems for this market, such as D. C. Hayes but most of these are useable only with certain bus configurations such as Apple or S-100. If you have one of these machines, this part of this column won't prove very useful to you.

I placed a call to U.S. Robotics to get more data on their 300 baud, direct coupled modem and was treated very well. They expressed a willingness to help me with my application and even sent me all their technical literature on the promise that I'd sent them a $5.00 check. No, I didn't tell them that I wrote a column for COMPUTE II. As far as they knew, I was just another hobbyist.

I also had some contact with TNW Corporation. They manufacture stuff for the PET (or other IEEE Bus equipped computers) so their direct-couple modem didn't turn out to be as useful for my particular application. But, if you're looking to turn your PET into an electronic mail system, TNW has the software and hardware to do just that. I believe they are working very closely with the PCNet people so they should have some good software coming out.

As it turns out, the PCNet software protocol is a bit on the complicated side for those of us not well versed in the esoterics of network theory and the like, so having a software package alredy prepared looks mightly appealing.

My personal application for a modem includes use on the PCNet as well as checking into one of the large time sharing systems like the Source or Micro Net (or both). Since I may want to automatically access a data base late at night, the modem/telephone interface needs to be fully automated.

I'll be checking out modems for a while and will report my findings.

Barcodes Come Of Age!!!

Back in 1976 (November to be exact) BYTE magazine introduced an interesting concept regarding program entry from magazine pages (or other printed media).

Using a code very similar to the Universal Product Code, which can be found on just about anything you purchase anymore, programs (and data) can be reproduced on paper in a form that can be fed directly into your computer. This, of course, eliminates, the laborious typing in of magazine software. Just think about the amount of wasted energy when 10,000 computerists across the country have to type in the same program? Now **THAT** amounts to a lot of effort!!! Well, this new scheme could put an end to all that.

I'll bet you're wondering if it's so great, why aren't all the magazines offering software in barcode format. Well, that's a fair question----and the answer is that up until now, bar code reading wands have cost from $300 up.

But that's all changed since Hewlett-Packard introduced the HEDS-3000 bar-code data entry wand for around $100 in single quantities. Now, for a little more than the price of a good audio cassette deck, you can have a truly revolutionary peripheral device for your computer!

Think of all the neat things that can be done with such a device. You computer music users now have the ability to load musical scores directly into your "instrument" (providing of course, music publishing companies print music in some sort of bar code format). Industrial controllers could have the control program or several programs printed right on the face plate for ease of operator input. You could easily input trip data to your car computer or phone numbers to your communication computer. The applications are numerous.

The April 1980 issue of BYTE has an article on the new HP bar code reader and the bibliography of past BYTE articles written on the subject, so I'd suggest you start there if you want more information.

HP can be contacted directly at: 640 Page Mill Rd., Palo Alto, CA 94394 Attention: John Sien.

I'm very tempted to spring for one of these devices but will probably have to put it under the modem on my priority purchase list.

If you'd like to see COMPUTE (or COMPUTE II) publish software in bar code format contact Robert Lock and make yourselves known.

MTU Graphics

I received the Micro Technology Unlimited Visible Memory board a short time ago and have been working on application ideas for this rather unique board.

For those of you not familiar with it: Visible Memory is both an 8K dynamic RAM board with invisible refresh AND a 320x200 bit-mapped video graphics board.

This clever design makes use of the fact that the video circuitry must read the entire 8K block at specified intervals and allows it to serve the double purpose of also refreshing the dynamic RAM. You're wondering why you didn't think of it, right?

"Bit-mapped" means that every bit in the 320x 200 screen matrix is represented by one bit in the screen memory. With this board, one has total control over every pixel. It's very similar to the Apple hi-resolution graphics in that respect, with the exception that the MTU board is slightly denser (320x200 vs. 280x193).

MTU also has some software available for this board that could, assuming you owned an AIM-65, turn your computer into a low cost version of the HP-85. One software package works together with AIM Basic to allow such things as mathematical functions to be graphed out on the display while another software package allows the built-in AIM printer to record whatever pattern is on the screen. How does that sound? That same software also allows text lines up to 80 characters in length to be printed SIDEWAYS on the AIM printer for increased readability.

My appreciation for AIM increased considerably when I saw it performing in this fashion.

Without any further software work, the AIM 65 coupled with some MTU hardware would seem ideally suited for duty in the laboratory, the classroom or most anywhere that a relatively low-cost graphics system can be justified. Assembling such a system turns out to be very easy. It can be performed by someone with moderate electronic skills and with totally "off-the-shelf" components.

But don't let your imagination stop here. Many other things can be done with such a display. How 'bout a 16-channel digital logic analyzer? Very possible with a bit-mapped graphics display.

Want to make your KIM, AIM, or SYM look like a PET? Simple.

PET's screen is organized as 25 lines of 40 characters each. Each of these characters is composed of an 8x8 dot matrix. Multiply 40 characters times 8 bits (character width) and what do you get? Why 320, of course. Then do the same with 25 lines times 8 bits and you get 200.

So, when you break down PET's display to the dot level, the MTU and PET display are precisely the same. It is possible to generate all PET's graphic characters in software or design your own special purpose characters for that matter.

Get the picture?

The Apple and Atari can be simulated in precisely the same fashion. Foreign language fonts are also possible.

Normal X Y plotting subroutines are also ineluded in the MTU graphics software.

You can get more information on these and other products from

Micro Technology Unlimited
P.O. Box 4596
Manchester, NH 03108
(603) 627-1464

Sound Chip Update

I finally got hold of some General Instruments Programmable Sound Generator chips (AY3-8910). One of them is residing on a prototype card along with a 6522, which interfaces the sound chip to my computer.

After some initial problems (with me, not the chip) I was able to get the sound generator to start generating some sound. I haven't yet even scratched the surface of what's possible with the PSG-maybe you'll also hook one to your computer and see what sounds you can get out of it.

In my next column, I'll write up the driver software to save you the trouble.

Lately, my mind has also been racing with some of the possibilities for ways to input music into the system as well as output it.

Hope For The OSI Users

There may be hope for you OSI users yet. No, not from OSI but from a company called AARDVARK TECHNICAL SERVICES (1690 Bolton, Walled Lake, MI 48088 tel (313) 624-6316).

They seem to have a really good attitude and sure have lots of low-cost game and utility software for C1 and C2 system users.

Their catalog says it all though with several BASIC program listings (including LIFE), at least 4 pages of useful info on Microsoft BASIC and the OSI system besides the incredibly large catalog of program offerings. Well worth their asking price of $1.

Remember the friend of mine who was working on using his C2-4P as a terminal for his new found love (a KIM-1)? Well, that story had a happy ending when he loaded in the dumb terminal program from AARDVARK and it worked perfectly the first time.

Love those happy endings. ©

The 1802?

Dann McCreary

Introducing the 1802

COSMAC? What's that, a new treat from Mac-Donaldland? Wrong! COSMAC is the trade name for the 1802 microprocessor developed by RCA. What's an 1802 got that your micro hasn't got? What do it's strongest supporters have to say about it? It's worst enemies? Before completely writing off the 1802 as not for you, why not tune in to this column a few times? The 1802 may turn out to be just what the doctor ordered for that special project you've been putting off. At the very least, seeing what makes one tick could be a mind expanding experience.

Since many of use are 6502 users taking our first real look at the 1802, let's be adventureous! A quick glance at our travel brochure promises a lot of intriguing territory to explore. So where do we begin? Today we'll take a reconnaisance flight over the 1802 map and get a broad view of COSMAC architecture. I'll point out some of the more re-nowned applications for a CMOS micro and suggest a few unusual possibilities as well. The point of this brief sightseeing tour is to stimulate your interest in a more detailed exploration of one or more possible 1802 applications. Come on aboard and fasten your seatbelts! Please speak up when you see some-thing you like - or don't like - and we'll move in for a closer look, or give it a wide berth as the case may be.

While we taxi for takeoff, a flight attendant points out that the terrain we'll be flying over consists of CMOS (Complementary Metal Oxide Semiconductor). Its' very low power dissipation makes the 1802 a good candidate for a variety of battery - based applications. We lift off smoothly, and as we gain altitude some prominent COSMAC features come into view.

Our first impression of the 1802 is that it has a lot of registers. Row after row of general purpose 16 bit registers pass by beneath us - sixteen in all. What are they good for? Any one may be used as an up or down counter or even a program counter. Look, there's one being used as a stack pointer! A couple over here are holding scratchpad data. A passing stewardess pauses to tell us that the only way to access the memory beyond the chip's horizon is indirectly through one of these sixteen registers. But who or what determines which is which?

Coming into view on our left are the P and X registers. They are each four bits wide and thus able to designate any one of the sixteen general registers. We note that P contains a zero, so RO is the program counter at the moment. X has an A in it, making RA the current data pointer, accessible to a wider than usual range of program instructions. On our right we see I and N, also four bit regis-ters. We ignore I for the moment since its' contents are strictly incidental to the instruction being executed and move on to the N register. It has three lines extending from it out past the edge of the chip. These lines are used to decode one of seven input or output ports.

Beyond the N register we see a register labeled T. It appears to be eight bits wide. As we descend for a closer look, we are surprised by a sudden interrupt! The contents of X and P are abruptly stuffed into T! X now contains a 2, making R2 the data pointer and a likely candidate for stack pointer. A one has been put into P and R1 is now the program counter! Fortunately, R1 contains the address of an interrupt routine. As we leave T behind, we note that the interrupt routine is methodically saving T's contents via R2 into a RAM stack. IE has been cleared, disabling further in-terrupts until the 1802 is done handling this one.

A little ways ahead of us we can see the D register. D is an eight bit register that looks remarkably like a 6502 accumulator. It is the main data manipulating register of the COSMAC. Branch instructions are available to test D for both zero and non-zero states. All arithmetic and logical results are placed in D and in the one bit register right next to it, DF. DF represents the carry, borrow or overflow from add, subtract and shift instructions. Unfortunately, try as we might, we are unable to locate a status register of any type.

The aircraft begins a climbing turn now in preparation for a return to our point of origin. We notice some lines stretching past the edge of the 1802 below us. Signals coming in on these lines appear to be changing the path of the program in operation below us. Consulting our map, we discover that they are in fact EF1 to EF4. Defined as negative-true inputs, these external flag lines can be tested for either state by special program branch instructions.

Sudden turbulence sends those of us in the aisles scrambling for our seats. The captains' voice over the intercom reassures us that it is only the COSMAC Q line toggling a speaker far below. We are now high enough (thanks to the 1802's GHI (Get High) command) to see that the program in progress down there is a musical greeting card playing 'Happy Birthday'. Stretching toward the horizon in every direction we can see COSMAC systems both large and small. An OSCAR amateur radio satellite passes

CDP1802 block diagram.

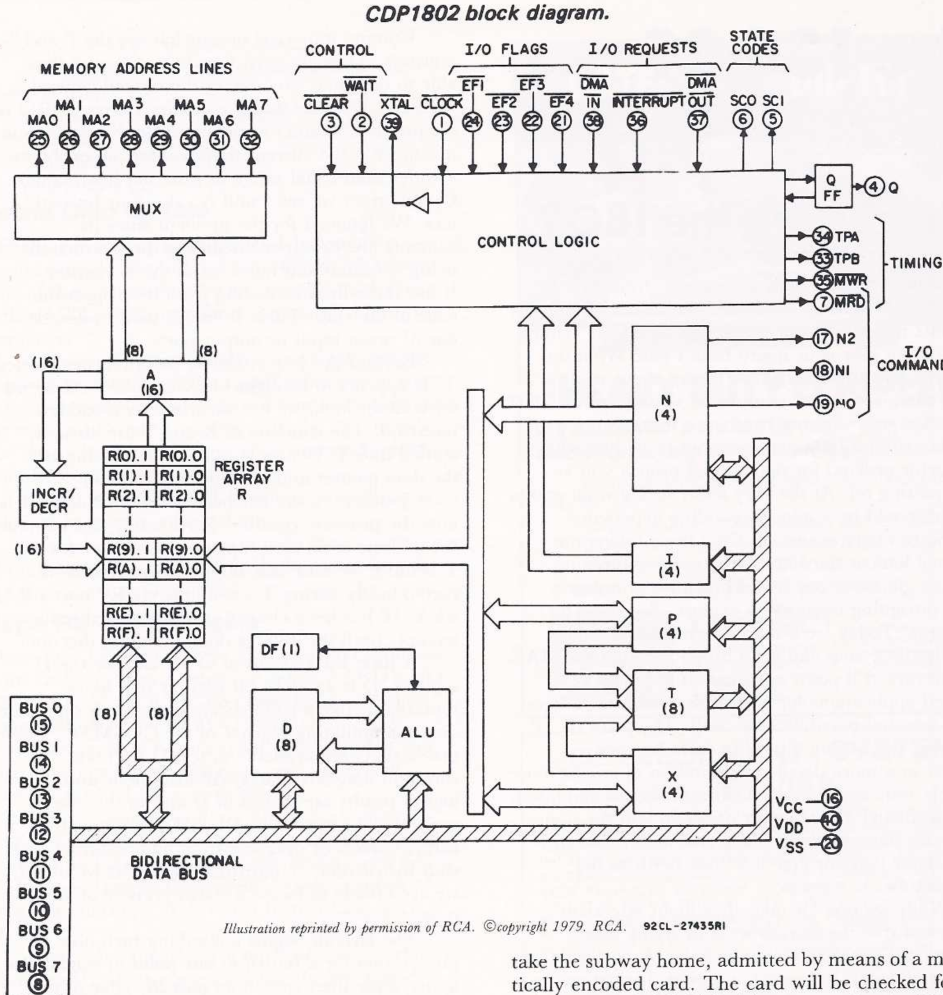


Illustration reprinted by permission of RCA. ©copyright 1979. RCA.

far overhead, relaying personal messages around the world under the control of an 1802. Off our left wing tip we see an intelligent model glider and a bewildered seagull riding a thermal in formation. Someone with a sharp eye spots a data logging weather station on a distant ridge, patiently accumulating wind velocity and direction information. The information it gathers will be used to determine the feasibility of installing a wind generator there.

Throughout the flight, I have been taking notes for this column with the aid of my secretary - alas, not a cute blonde, but an 1802 with a one hander keyboard dumping blocks of data into a mini-cassette recorder. When I get home I'll polish and expand the notes with a full scale text editor on my APPLE II. We'll be landing soon. Some of us will take the subway home, admitted by means of a magnetically encoded card. The card will be checked for validity by a COSMAC driven control unit. Others will drive out of a parking lot serviced by 1802 based ticket dispensing equipment. But before we do come down, I'd like to thank you for joining me on this flight of fancy. Many of these applications are real, and others can be realized by you and me if we show a little interest and effort. I hope you'll take a few moments right now to drop me a line and share with me and other COMPUTE II readers some of your ideas for the 1802. Tell me what you are doing or would like to do with a CMOS micro. Shall we look at real-time control? Security systems? Medical applications? Portable instrumentation? Smart 6502 peripherals?

We've just rolled to a stop at the terminal. Don't forget your personal effects. Hope you've enjoyed your trip. ©

Creating Data Files on Tape With OSI Computers

Rodger Olsen
1690 Bolton
Walled Lake, Michigan 48088

It's a dream come true. $349.00 for a complete ready-to-go computer with a high level language in ROM, a cassette interface keyboard, and the works. My GOD, it's more powerful than the million dollar IBM goodie that I learned on fifteen years ago, and I've been around long enough to be impressed with 8K of memory. I can remember running a meat-packing plant with 4K and doing the complete inventory, pricing and billing for a lumber yard with less than 8K.

Every owner has visions of checkbook balancing with the flick of a digit, payrolls done with the wave of an index finger and semi-magical appointment books. Unfortunately, he soon realizes that the blasted thing has a memory just slightly worse than a teenager told to clean his room. Everytime you turn the switch off, the bloody thing forgets *everything* it has learned.

That is not too bad in static programming - that is, programs that do the same thing every time they are run - such as games - or programs that require unpredictable data to be entered every-time - such as statistics programs. The problem starts to gnaw when the system must generate data *this* time to use *next* time the program is run. OSI does does mention in its manual that it is possible to do limited files on tape, but then neglects to mention how this feat is accomplished.

In fact, the system has an excellent tape facility once it is properly understood. The secret of its versatility is two fold. First, the tape system stores all information in ASCII. Unlike some of its competitors, it does not tokenize or encode the data it outputs. That's what allows us to run printers off the cassette interface.

The second thing that makes the system useable is the simplicity of its concept. When the system receives a SAVE command for any source - including a program line, it simply puts out to the ACIA any information that it puts to the screen. That means that any LIST or PRINT statement will then put on tape anything it puts on the screen. In a similar manner, a LOAD command tells the system to treat the ACIA as if it were the keyboard. That's nice, because it means that any information and any command that can be keyboard input, can be input from tape. I imagine that the original intention from OSI was to save on software overhead, but the final result is a tape interface of exceptional simplicity and versatility.

It turns out that there is even a trick that will allow you to turn the ACIA on and off without inputing LOAD or SAVE commands. The problem is that normally the SAVE command is terminated by inputing the LOAD command and the LOAD command disables the keyboard until it is terminated with an input of a space from the space bar. That is a rather unhandy set of things to do in the middle of the program. Fortunately, you can manipulate the SAVE and LOAD flags directly with POKE statements. POKEing a 1 into location 517 will enable the SAVE which can be disabled by POKEing a 0 into the same location. Similarly, LOAD can be disabled with a 1 POKEd into location 515. That simplifies program control of the functions.

The first and simplest method of setting up a tape data file is to simply read the information onto tape in raw form and then read it off the tape with a LOAD command followed by an INPUT statement. It works this way:

```
5 REM PROGRAM 1
10 DIM A(10)
15 REM RECORDER REMINDER-TURN ON CASSETTE
   INTERFACE
20 INPUT"IS RECORDER READY";A$:POKE517,1
25 REM READ OUT DUMMY DATA
30 FORX=1TO10
40 PRINTX
50 NEXT
55 REM TURN OFF SAVE FUNCTION
60 POKE517,0
70 END
80 .
90 .
```

Now we have ten numbers - or the latest payroll data - on tape in raw ASCII form. Reading the DATA is equally simple. We do it this way -

```
190 REM PROGRAM II
200 DIMA(10)
205 REM WE DON'T CHECK FOR NOT READY AS NO
207 REM ONE EVER INPUT UNTIL HE WAS READY
210 INPUT"IS RECORDER RUNNING";A$
215 REM TURN ON ACIA
220 LOAD
225 REM SYSTEM THINKS THE ACIA IS A
    KEYBOARD
230 FORX=1TO10:INPUTA(X):NEXT
```

```
235 REM TURN OFF ACIA-CHECK TO SEE IF
    DATA RIGHT
240 POKE515,1
250 FORX=1TO10:PRINTA(X):NEXT
260 END
270 .
280 .
290 .
300 .
```

That is a beautifully simple system that is easy to use and memory efficient. We use it on one of the graphics plotting programs we sell. We do not, however, use it for serious applications as it does have a couple of limitations. There are two basic problems with this method of data storage. First, you must be certain that the tape is on a tone filled section before you start the system listening. Remember that the system will treat the ACIA exactly as if it were a keyboard. Also remember what happens when you hit a carriage return with no input (a NULL input). The blasted thing breaks out of the program! If you have an OSI, you have seen the garbage that the system picks up from random noise on the leader and the unrecorded parts of the tape. If any of those random noises is a carriage return, you just left your program. Fortunately, the system puts out a steady tone even when it is not being used. If you have recorded some of the tone of the tape before you start storing data, you can go up to the recorded section and start there in a relatively noise free environment.

The second problem is that this simple system has no indexing or error checking. If you get the first DATA byte out of place (by, say, winding the tape up too far before you read it), all of the other data bytes are also out of place. For instance, our graphics program puts out two numbers to describe each point. The first is the address on the screen and the second number is the value to POKE into that point. The program reads the tape and POKEs the second number into the first one. If you miss the first number, the system may confuse which numbers are locations and which are POKE values. It could end up trying to POKE 53467 into location 161. IT has no way of knowing what comes first.

There is a strange fix that we have used to get around those problems in applications where memory is short and we couldn't afford the room for DATA statements. We need a couple of assumptions to discuss this one. Assume that we will need DATA in line 1010 to perform some function. We also need to assume that the DATA will have been generated at the end of a previous run of this or another program. It was, for instance, handy in an inventory program where we had to store the counts at the end of one day and input them at the beginning of the next.

To make sure that we do not jump accidently out of the program, we diliberately leave the program with a STOP and have the generated data tape cause a jump to the next program line after the information is input. To avoid the problem of misread DATA - at least in the sense of mis-indexing and therefore misplaced reads, we will name each variable as it is input. Let's set up a little inventory program for a hot dog stand.

The first section we have to put up is the section that stores DATA. Assume that at the end of the day, we jump to line 2000 to store the DATA that we have accumulted during the day. (We'll output dummy DATA in the example). Remember that you would have come up with real values for A(1) to A(10).

```
2000 REM PROGRAM III
2010 DIMA(10):REM SET UP SOME DUMMY DATA
2020 FORX=1TO5:A(X)=24
2030 NEXT
2040 A(6)=8:A(7)=9:A(8)=1
2050 INPUT"IS RECORDER READY";A$:POKE517,1
2060 PRINT:PRINT:REM OUTPUT THE DATA-WITH
     NAMES OF VARIABLES
2070 FORX=1TO10:PRINT"A("X")="A(X):NEXT
2080 REM OUTPUT A DIRECT COMMAND FOR
     JUMP AT END OF LOAD
2090 PRINT"GOTO1020
2100 REM TURN OFF SAVE
2110 POKE517,0:END
```

Reading it requires a slightly different technique than the last example. We will need to turn on the LOAD function and stop the program. To avoid accidental jumping out of a program, we will deliberately jump out and arrange for continuance. The line we jump back to will have to turn off the LOAD function and restore normal keyboard operation.

```
1000 REM PROGRAM IV
1005 REM HALT PROGRAM-WAIT FOR INPUTS
1010 INPUT"IS RECORDER READY";A$:LOAD:STOP
1015 REM GOT INPUTS-TURN OFF LOAD AND
     RESTART PROGRAM
1020 POKE515,1:FORX=1TO10:READA$(X):NEXT
1025 REM CHECK OUT THE DATA
1030 FORX=1TO10
1040 PRINT" GOT "A(X);A$(X):NEXT
1050 DATA"BUNS","HOT DOGS","HAMBURGERS",
     "CHEESE PATTIES","COKE CUPS
1060 DATA"COFFEE CUPS","PLATES","DRINKS",
     "PROFITS","FUTURE"
1070 END
```

Now when the program reaches line 1000, it will stop and wait for keyboard inputs. As the LOAD command is activated, it will treat the ACIA as the keyboard. It will input each variable (along with its name), receive the direct mode command to jump to 1010 where it will turn off the ACIA and continue with the rest of the program.

It is almost, but not quite, foolproof. The only thing that can cause problems is noise on the tape that looks like a line being input or erased. (i.e. 100xK1die9). These will cause a CONT error as the system thinks you have modified the program.

It is also highly efficient in memory usage. You need to remember that inputing a variable as a program line or a DATA statement charges twice for the same variable. The line 100 A(4) = 19 requires that you use at least 7 bytes of memory for the program line material that names and assigns the value to variable *plus* you have the overhead of storing the variable value in an array in the work area. The method we just used uses only the variable storage space in the work area and saves the program line bytes.

However, the most foolproof and probably most satisfactory overall system - WHEN YOU HAVE MEMORY SPACE - is to input the information in computer generated DATA statements. The information is then input as actual program lines and no more prone to error than standard program input - and with OSI that reliability is high.

The only neat trick you have to pull is to get the computer to write the DATA lines. Assume that you want to keep track of the checks from one session to the next, preferably as automatically as possible. We won't go through the entire program - you need *some* fun yourself - but here is the section that would store checks as you enter them. You'll notice that there is no separate read section - aside from line 540. The program reads the entries it stored from last time when it fires up. When you store the DATA, it ends by listing the main body of the program on the same tape - handy for the next time. Just load and run. The self-listing format is, of course, only handy for those programs where you store DATA once and where the program is not a lot longer than the DATA. It would be rather unhandy to have to store an 8K program each time you wanted to store 4 bytes of information, but the data writing lines are the same whether or not you also store the program.

Before leaving this topic, we should mention that it is much more efficient to store several DATA items per line. We neglected to do so here only to insure the clarity of the example.

Now we come to the last and most exotic of the file possibilities. Yes, Martha, it *is* possible to have names filed on tape with your OSI system. There are several methods by which this can be done, but this one is my favorite. It's the simplest.

The first thing we need is a way to read the ACIA and therefore scan the tape without entering anything into the system. The WAIT command, CHR$ functions and PEEK will let us do it. To cover all the OSI systems, we'll have to have two programs as the ACIA in the C1 (SUPERBOARD) is in a different location than it is on the rest of the systems. We'll use the WAIT command to halt the system until the low order bit on the ACIA status register is high and, when it is, PEEK the character from the main register (one location higher on an OSI system), and, if we want to see it, print the CHR$ofthe PEEKed number to the screen. You got all that? O.K.

```
500 REM PROGRAM V
510 REM DIM AS LARGE AS MEMORY PERMITS
520 DIMCH(50),AM(50)
530 REM READ NUMBER OF CHECKS ON FILE-THEN THAT MANY CHECKS&AMOUNTS
540 READNC:FORX=1TONC:READCH(X),AM(X):NEXT:NC=NC+1
550 REM REPORT WHATS ON FILE
560 PRINT"ON FILE":FORX=1TONC:PRINT"NUMBER"CH(X)" AMOUNT$"AM(X):NEXT
570 REM INPUT NEW CHECKS -9999 CAUSES JUMP TO END
580 INPUT"CHECK NUMBER";CH:IFCH=9999THENNC=NC-1:GOTO620
590 REM IF NOT 9999 THE GET AMOUNT-INCREMENT COUNTER-DO AGAIN
600 INPUT"AMOUNT";AM:CH(NC)=CH:AM(NC)=AM:NC=NC+1:GOTO580
610 REM FAMOUS DELAY AND CHECK LINE
620 INPUT"IS RECORDER READY";A$:POKE517,1
630 REM PRINT OUT NUMBER ON RECORD NOW
640 PRINT:PRINT:PRINT"700 DATA"NC
650 REM PRINT OUT CHECKK NUMBER AND AMOUNTS
660 FORX=1TONC:PRINT600+2*X"DATA"CH(X)","AM(X):NEXT
670 REM LIST OUT BODY OF PROGRAM
680 LIST 500-690
690 REM DUMMY DATA LINE TO PREVENT READ ERROR ON FIRST RUN
700 REM IT WILL BE REPLACED BY GENERATED LINES
710 DATA0,0,0
```

Here's how it works:

```
4 REM PEEK A PORT UTILITY
8 REM C2/4/8 VALUES
10 A = 64512:B = A + 1
20 WAITA,1:PRINTCHR$(PEEK(B));:G0T010
35 REM C1 VALUES
40 A  = 61440:B = A  +  1
50 WAITA,1:PRINTCHR$(PEEK(B));:G0T050
```

This is a **TWO** line program. C2/4/8 users enter lines 10 and 20. C1 users enter lines 40 and 50.

The previous little program is handy for other things as it will let you read what is on a tape without entering anything in the computer. If you have ever sat staring at a tape and wondering if this is the right program segment to enter next or the right data and been afraid to disturb the 8K program you have spent 15 minutes loading into the system, you're gonna love the PEEK A PORT.

Before we can use it, we'll have to make up some files, name them and fill them with DATA. For names, we'll use a trick from the TRS-80. You can enter any name as long or complicated as you want for each file. Unfortunately, we'll only store and search for the first character of the name.

In order to show that it is a file name and not another character, we'll store it as a number converted up out of the ASCII range. Normally, ASCII numbers range from 33 to 127 even if you include lower case letters. What we will do is take the first letter of the file name (using the ASC function) add 100 to it to take it out of the ASCII range and print it to tape using the CHR$ function. Whenever the system sees a number from 133 to 227, it will know that that is a file header as it cannot occur naturally. CUTE?

```
700 INPUT"FILE NAME";A$
705 REM KICK FILE NAME UP OUT OF ASCII
    NORMAL RANGE
710 J=ASC(A$)+100
712 REM USUALLY THE PROGRAM WILL KNOW-
    THIS ONE IS A DUMMY!
715 INPUT"HOW MANY ITEMS IN FILE";NI
720 INPUT"IS RECORDER READY";A$
730 SAVE:PRINT:PRINTCHR$(J)
735 PRINTNI
740 FORX=1TONI:PRINTX:NEXT
750 POKE517,0:GOTO700
```

Reading it turned out to be a little more complicated than I had anticipated. Of course, my main problem was caused by my learning disability - I cannot learn to keep my mouth shut. I casually mentioned to a customer that this would be a good system for the files he needed and assured him that it would be simple to implement. All we had to do was use PEEK-a-PORT, detect the character and jump to a load.

The first problem I ran into was that the bloody thing kept jumping out of the program everytime we tried to read a file. It would detect the header character correctly and then fail to read any data - in fact, it kept inputing the data as empty line numbers and erasing part of my program.

After much time spent on timing loops and fruitless tests, I remembered that everytime a character is printed to tape, the system also prints a carriage return unless specifically told not to do so. I added the second WAIT and halted the system until the carriage return passed.

The next problem was that the system would not restart properly after the second load. I still haven't looked into why it happens, but brute experimentation showed that the LOAD could only be turned off once with the POKE515 command. A carriage return was needed the second time even if the POKE had been executed. The problem had never shown up before as the typical program only read DATA once. I could not, however, see having named files and only reading one during a typical run.

Put down in print, the program looks simpler now than when I was struggling with it.

At this moment several astute programmers are going to realize that we could have eliminated the necessity of the second WAIT and testing for the carriage return after the file header by printing the file header followed by a ";" to eliminate the carriage return and the necessity for a test. Sorry Charlie, I'm going to save you several hours of frustration by mentioning that when I tried that, the system often left too little time between the file header and the first character to process the LOAD and INPUT commands and therefore, intermittently lost the first character of the file. I refuse to explain in detail how I know that it's an intermittent problem that can lead a programmer down a blind alley for several frustrating hours.

As it finally developed, it is a simple program that reads the PORT as before and now tests for the header character rather than printing it to the screen. When it finds the header, it jumps to a LOAD/INPUT command that inputs the file. It then prints out the first character of the file name (Header character-100) so that you can check which file it read.

I should point out that it still does not pay to put a lot of files on one tape. The system is limited to a sequential 300 BAUD file search. As you can use the fast forward and reverse on your tape recorder, and as OSI's excellent tape system allows you to read the file on the screen as it passes, it is usually faster to find it yourself rather than wait through a machine search. I rather suspect that one of the reasons other systems added named files to their BASIC was that they could not search a tape visually as it was tokenized and did not appear on the screen.

The one advantage the named file has, aside from impressing your friends and neighbors, is that it allows you to pack a lot of DATA closely on a tape.

You can run files right next to each other, wind the tape to somewhere near the spot where the file you want starts and then load only the file you named. Once you catch on to storing DATA, you'll find that that itty-bitty system of yours can function as well as the giants of a few years ago.

```
790 REM A=64512 FOR C2/4/8-THE ACIA IS IN A DIFFERENT LOCATION
800 A=61440:B=A+1:DIMA(15)
810 INPUT"FILE NAME TO LOAD";A$:J=ASC(A$)+100
815 REM LOOK FOR FILE NAME-ONE BYTE ONLY
820 WAITA,1:C=PEEK(B):IFC<>JTHEN820
825 REM CLEAR OFF CR AFTER NAME-WILL JUMP PROGRAM IF  YOU FORGET
830 WAITA,1:IFPEEK(B)<>13THEN830
833 REM SET UP LOAD AND GET NUMBER OF ITEMS IN FILE
835 LOAD
837 INPUTNI
840 FORX=1TONI:INPUTA(X):NEXT
845 REM MAKE SURE IT IS CORRECT FILE
850 PRINTCHR$(J-100)"FILE
855 REM POKE 515 ONLY WORKS ONCE-ASK FOR SPACE BAR PRESS TO ALLOW REPEAT
860 PRINT"HIT SPACE BAR":POKE515,1:GOTO810
```

©

Some OSI Tape Utilities

Edward H. Carlson
3872 Raleigh Dr.
Okemos, MI 48864

At best, tape is a rather slow medium for storage; at worst, it can be very aggravating. Here are some soft and hard devices I use to minimize the inconvenience. I usually use short cassettes, 5 minutes to a side, and put one program on each side. This eliminates all searching. Rewinding from one end to the other is not often necessary because the program on one side is usually much more "popular" than that on the other.

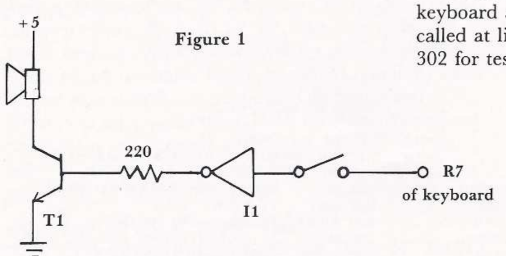
Occasionally I suspect that I have a bad cassette. I have written the two programs shown in Listing 1 to test the cassettes. One program (starting at line 315) fills the entire tape with the character $FF. The other (starting at 110) then reads the tape and "peeps" if it finds any character that is not $FF. (The peep comes from the BELL hard- and software described below.) I suspect that this test is not foolproof since a bad spot may fall between 2 characters or otherwise be invisible to the test. I have found a few bad cassettes this way. $FF seems to be a better test character than $00. Retesting the tape nearly always confirms the bad spot.

The other device I use is a bell or tone output. I call this tone at the end of my

```
  5 0000          ;
 10 0000          ;         READ FILLED TAPE
 20 0000          ;
 30 0000          ACIA   =$FC00 TAPE PORT
 40 0000          BELL   =$CB48 SUBROUTINE TO EMIT A SHORT TONE
100 C100          *      =$C100
110 C100 200DC1   START  JSR RT      READ CHARACTER FROM TAPE
120 C103 C9FF            CMP #$FF    IS IT $FF ?
130 C105 F0F9            BEQ START   IF YES, READ ANOTHER
140 C107 2048CB          JSR BELL    IF NO, EMIT TONE FOR BAD CHAR.
150 C10A 4C00C1          JMP START   CONTINUE TESTING TAPE
190 C10D          ;
192 C10D          ;      READ TAPE SUBROUTINE
199 C10D          ;
200 C10D AD00FC   RT     LDA ACIA    READ 6850 STATUS REGISTER
210 C110 4A              LSR A       SHIFT RECEIVE STATUS BIT TO CAR
220 C111 90FA            BCC RT      1 = RECEIVE DATA REGISTER FULL
230 C113 AD01FC          LDA ACIA+1 READ RECEIVE DATA REGISTER
240 C116 60              RTS
300 C117          ;
310 C117          ;       FILL TAPE WITH $FF
312 C117          ;
315 C117          WT     =$BF15
320 C000          *      =$C000
330 C000 A9FF            LDA #$FF
340 C002 2015BF   WRITE  JSR WT
350 C005 4C02C0          JMP WRITE
360 C008          ;
361 C008          ;      THIS SUBROUTINE IS IN THE BASIC ROM
362 C008          ;
399 BF15          *      =$BF15
400 BF15 48       WT     PHA         SAVE ACC. (CONTAINS CHAR.)
410 BF16 AD00FC   B1     LDA ACIA    READ STATUS REGISTER
420 BF19 4A              LSR A       SHIFT TWICE TO MOVE TRANSMIT
430 BF1A 4A              LSR A           STATUS BIT TO CARRY
440 BF1B 90F9            BCC B1      1 = TRANSMIT DATA REGISTER EMPT
450 BF1D 68              PLA         RECOVER CHARACTER TO BE SENT
460 BF1E 8D01FC          STA $FC01   WRITE CHARACTER TO TAPE
470 BF21 60              RTS
```

fast tape read and write programs to signal that the message transmission has been completed. I usually produce the tone by toggling a bit on my 6522 VIA (on the 500 board of my OSI C2-4P) and putting the signal into a speaker. For those machines lacking a parallel port, a bit can be toggled at the polled keyboard, (if you have a polled keyboard, all the newer OSI machines do). Figure 1 shows how. The 40 ohm, 1'' speaker is glued base down to a convenient spot inside the computer. The 542 keyboard has a prototyping area. Here I put a 7404 hex inverting buffer and an audio transistor to drive the speaker. I have put a row of switches on the panel of my machine. One of them can turn off this toggling bit to avoid the annoying tone produced when any of my regular programs are polling the keyboard. If your computer has no convenient prototyping area, you can glue the 7404 down with its ''feet'' in the air and *quickly* (with a clean pencil soldering iron) solder the necessary wires and components on.

Figure 1



T1 SK3010 or other audio transistor
I1 7404

```
  1 0000          ;
  2 0000          ;        BELL
  3 0000          ;
 10 0000          KYBD   =$DF00
 20 0000          PERIOD =$80   SETS THE LENGTH OF ONE HALF CYCLE
 30 0000          DURASN =$FE   COUNT DOWN ADDRESS FOR TONE LENGTH
100 C300          *      =$C300
101 C300 A90A     BELL   LDA #10      OUTER TIMING LOOP INIZ.
102 C302 85FE            STA DURASN
104 C304 A0FF     BE1    LDY #$FF     INNER LOOP INIZ.
105 C306 A900     PEEP   LDA #0       LOW HALF CYCLE
130 C308 8D00DF          STA KYBD
140 C30B 201EC3          JSR CYCLE
150 C30E A9FF            LDA #$FF     HIGH HALF CYCLE
160 C310 8D00DF          STA KYBD
170 C313 201EC3          JSR CYCLE
175 C316 88              DEY          INNER LOOP INCREMENT
176 C317 D0ED            BNE PEEP
180 C319 C6FE            DEC DURASN   OUTER LOOP INCREMENT
182 C31B D0E7            BNE BE1
188 C31D 60              RTS
190 C31E          ;
200 C31E          ;        CYCLE: HOLD VOLTAGE FOR HALF A CYCLE
210 C31E          ;
220 C31E A680     CYCLE  LDX PERIOD
230 C320 CA       CY1    DEX
240 C321 D0FD            BNE CY1
250 C323 60              RTS
297 C324          ;
298 C324          :        DRIVER
299 C324          ;
300 C400          *      =$C400
302 C400 2000FD   MAIN   JSR $FD00  WAIT FOR A KEYSTROKE
305 C403 2000C3          JSR BELL   MAKE A TONE
310 C406 4C00C4          JMP MAIN   DO IT AGAIN
```

Listing 2 shows a program which will toggle row 7 of the keyboard and produce a tone. The subroutine is normally called at line 101 ''BELL''. I have provided a DRIVER at line 302 for testing purposes.

©

Editor's Note:
Names and faces. . .
Rodger Olsen from Aardvark studies the crowd at the Fifth West Coast Computer Faire. Photo by Eric Rehnke.
RCL

Part 2: Implementing the IEEE-488 Bus on a SYM-1

DESIGNING AN IEEE-488 RECEIVER WITH THE SYM

Larry Isaacs, COMPUTE. Staff

This is the second part of an article describing the use of a SYM-1 to interface a PET to a Spinwriter with a serial interface. We will continue to divide the more complex functions into simpler sub-functions when necessary. Once the sub-functions are simple enough, they will be implemented. In the first part, the interface was divided into four sub-functions: INIT, PRINT, CYCLE, and INTERFACE. Implementations for PRINT and CYCLE have already been presented. Briefly, the PRINT routine handles the communication with the Spinwriter. By using the ETX/ACK protocol, the PRINT routine keeps the Spinwriter printing at its maximum speed. The CYCLE routine handles the handshaking necessary to transfer a byte from the IEEE 488 Bus to the SYM. For convenience, these routines are given again in the complete listing of the interface software found at the end of this article. Also, the hardware to connect the Spinwriter to the SYM is shown again in Figure 2.

Before we can begin work on the INTERFACE sub-function, we must first understand how the PET will try to communicate with the SYM using the IEEE 488 Bus. Now we will continue with a description of this communication procedure.

Communicating On The IEEE 488Bus

The next step is to become familiar with how the PET communicates on the IEEE Bus. This discussion will involve two more signal lines. These are the ATN (Attention) line and the EOI (End Or Identify) line.

Each communication on the IEEE 488 Bus can be described as a sequence of three parts. In the first part the PET identifies which device it wishes to communicate with. In the second part it sends or receives the data. And finally in third part, the PET terminates the communication sequence. Each part makes use of the byte transfer cycle described previously to transfer information. However, the information transferred in the first and third parts is differentiated from the second by the state of the ATN line. During the first and third parts the ATN line is low, indicating that the bytes transferred should be treated as commands and not data.

Here is a brief description of what happens during a communication sequence with a device, or devices, which only receives data, such as our printer. I will assume that prior to the beginning of the sequence, all devices on the bus are in the inactive state, i.e. the NRFD line is high.

The sequence begins with the PET setting the ATN line low. This brings all operating devices on the bus to the active state. The PET now executes a byte transfer cycle sending the device address to each device. Only those devices whose device address matches the one sent by the PET will continue with the communication sequence. All other devices will return to the inactive state at the end of this first part. The Commodore printers use device address 24 hex. The lower 5 bits contain the device number, in this case 4. The upper three bits, "001", indicate that the device is to receive data. A "010" in the upper three bits would indicate the device is to send data. Now the PET may end the first part by setting the ATN line high, or transfer another byte known as the secondary address before setting ATN high. The secondary address is used to address different functions or channels within the selected device.

The second part consists of the required number of byte transfer cycles to transfer the data to the device. In most cases, the PET will signal that the last data byte is being transferred by setting the EOI line low during the last cycle. Because the EOI isn't always sent, it wouldn't be a reliable signal to use for determining the end of this part of the communication sequence.

For the third part, the PET sets the ATN line low again, and executes a byte transfer cycle which sends $3F hex to all active devices. This is the UNLISTEN command, which tells all listening devices to stop receiving data.

One requirement for the interface which may not be obvious is that once the communication sequence has reached the second part, all commands except for the UNLISTEN command should be ignored. It would not be a violation of the IEEE 488 Bus Standard for the PET to activate a device which sends data at the same time as one which receives data, and have them communicate directly with each other.

There is one other IEEE signal line which should be included in the interface. This is the IFC (Interface Clear) line. Whenever this line goes low, the interface should return to the inactive state.

Now we are ready to deal with the hardware requirements for communicating on the IEEE Bus. We will be using 6522 #2 on the SYM for the necessary I/O signals since all of the I/O lines from both ports go to the A-A connector. If necessary, the 6522 supplied as 6522 #3 could be moved to the #2 socket, losing only a few features which aren't needed for this interface. The main hardware requirement concerns a requirement for the delay between ATN going low to the time when NRFD is set low by a device. The IEEE 488 Standard calls for a maximum of 200 nanoseconds for this delay. Though the PET can't operate this fast, it does operate too fast for the SYM to meet this requirement using just software. The solution to obtain the necessary speed is to selectively send the ATN signal back out the NRFD line. The SYM can then assume control of the NRFD line when it is ready. The only other hardware needed are a couple of open-collector gates for the Wire-or requirements of the NRFD and NDAC lines. The circuitry shown in Figure 1 will meet these requirements.

Interface

The main function of the INTERFACE sub-function is to handle the communication sequence for the IEEE

Listing 4

```
procedure INTERFACE

procedure ATNIRQ begin ... end; {handles the IEEE communication}
procedure IFCIRQ begin ... end; {resets the interface}

begin {INTERFACE procedure}
   repeat
      if INTERRUPT=TRUE then
         begin
            if IRQ=ATN then ATNIRQ;
            if IRQ=IFC then IFCIRQ
         end
   until 2+2=5 {hopefully repeat forever}
end;
```

Bus. The first decision we must make is how the INTERFACE software will know when a communication sequence has begun, or when the IFC line goes low. Since the IFC signal is supposed to reset the device regardless of its current state, this signal should be tied to an interrupt. For greater flexibility we will tie the ATN line to an interrupt as well. This will allow the SYM to do other things when not being used as an interface.

The use of interrupts now provides a basis for dividing the INTERFACE sub-function into smaller parts. Listing 4 shows my division of the INTERFACE sub-function.

At this point we are almost ready to write the assembly language for the remaining parts of the software. However, ATNIRQ needs one more division. This involves addressing the question of how much intelligence to put in the interface. One answer is to program ATNIRQ in a way that leaves the door open for expansion. This can be done easily using the secondary address to call different interface routines. The division for ATNIRQ is shown in Listing 5. The "case" statement in this listing is a multiway subroutine jump. If SECADDRS is 0 when the "case" statement is executed, the SENDASCII procedure will be executed. For other secondary addresses, the DUMPCHRS procedure will be executed.

Listing 5

```
procedure ATNIRQ

procedure ATNINIT; begin ... end; {get ready for communication}
procedure SENDASCII; begin ... end; {input data and print it}
procedure DUMPCHRS; begin ... end; {ignore data}

begin {ATNIRQ statements}
CYCLE; {get device address}
if DATA=MLA then
   begin
      ATNINIT;
      CYCLE; {get next byte, possibly a secondary address}
      if ATN=LOW then
         begin
            SECADDRS := DATA;
            CYCLE
         end;
      case SECADDRS of
            0 : SENDASCII;
         1..15 : DUMPCHRS;
      end {case statement}
   end {if statement}
 end; {ATNIRQ}
```

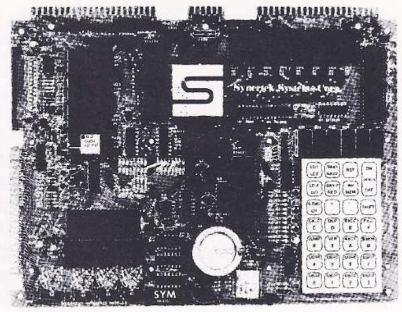
Now we can write the assembly language for INIT, then IFCIRQ, and finally ATNIRQ. Not clearly shown by the preceeding PASCAL programs is how the machine language should actually handle the interrupts. After an interrupt occurs, the first thing the machine language must do is save the register contents. Then it must test to see what interrupt occured. If it was an ATN interrupt, then the current stack pointer must be saved and ATN interrupts disabled before continuing with the rest of the ATNIRQ routine. If the interrupt was an IFC interrupt, the IFCIRQ routine should test to see if the ATNIRQ routine was executing. If it was, the IFCIRQ routine must restore the stack pointer to the value saved by ATNIRQ and reenable the ATN interrupt before restoring the registers and returning to the interrupted program.

The full listing of the assembly language for the interface is given in Listing 6. I've tried to write the assembly language so it can be easily expanded. Just remember that when you put a different routine in SCTABLE, the first data byte will have already been fetched by CYCLE when your routine is entered.

Summary

I've tried to make this article as much an example of interface design as one describing an actual interface. Most of the material presented dealt with needed facts or the steps involved in reaching a solution. I do not wish to imply that designing an interface should proceed from start to finish as easily as this article makes it seem. It is very likely that during your design, you will come upon a piece of new information or see a different approach which would have been highly useful at some previous step. This occured a few times during this design. Sometimes it is necessary or perhaps desireable to return to that previous step and take a different path. However, if you do enough preparation and planning before you begin the design process, you shouldn't have to backup too many times.

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.

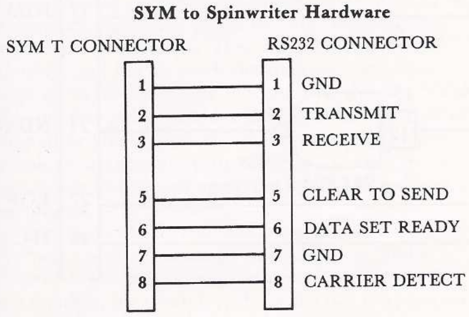
```
                ; IEEE INTERFACE
                ; WITH HARDWARE
                ; VERSION 2.5
                ;
                ; CONSTANTS
                UNLISTEN     .DE $3F
                BS           .DE $08
                UNDLN        .DE $5F
                LF           .DE $0A
                COLON        .DE $3A
                SPACE        .DE $20
                COMMA        .DE $2C
                CR           .DE $0D
                ;
                ; VARIABLES
                COUNT        .DE $E0
                SIGNALS      .DE $E1
                DATA         .DE $E2
                MLA1         .DE $E3
                SEC.ADDRS    .DE $E4
                TEMP         .DE $E5
                LENGTH       .DE $E6
                NL.FLAG      .DE $E7
                SCAN.CNT     .DE $E8
                F.LEN        .DE $E9
                SP.IEEE      .DE $EA
                ;
                ; ADDRESSES
                ACCESS       .DE $8B86
                TOUFL        .DE $A654
                SDBYT        .DE $A651
                TECHO        .DE $A653
                OUTCHR       .DE $8A47
                INCHR        .DE $8A58
                CRLF         .DE $834D
                TOUT         .DE $8AA0
                @2ACR        .DE $A80B
                @2DDRA       .DE $A803
                @2DDRB       .DE $A802
                @2PCR        .DE $A80C
                @2IER        .DE $A80E
                @2IORB       .DE $A800
                @2IORA       .DE $A801
                @2IFR        .DE $A80D
                OUTVEC       .DE $A663
                UIRQVC       .DE $A678
                IND.JMP      .DE $EE
                ;
                             .BA $200
0200- 20 86 8B  INIT         JSR ACCESS     ;INITIALIZATION
0203- A9 24                  LDA #$24
0205- 85 E3                  STA *MLA1      ;MY LISTEN ADDRESS
0207- A9 90     INIT.SYM     LDA #$90
0209- 8D 54 A6               STA TOUFL      ;ENABLE CRT
020C- A9 10                  LDA #$10
020E- 8D 51 A6               STA SDBYT      ;SET FOR 1200 BAUD
0211- A9 00                  LDA #$00
0213- 8D 53 A6               STA TECHO      ;OUTPUT & NO ECHO
0216- A9 A0                  LDA #L,TOUT    ;SET OUTPUT VECTOR
0218- 8D 64 A6               STA OUTVEC+$1
```

```
021B- A9 8A              LDA #H,TOUT
021D- 8D 65 A6           STA OUTVEC+$2
0220- A9 53              LDA #L,INTERFACE
0222- 8D 78 A6           STA UIRQVC     ;SET USER IRQ VECTOR
0225- A9 02              LDA #H,INTERFACE
0227- 8D 79 A6           STA UIRQVC+$1
022A- A9 02              LDA #$02
022C- 85 E0              STA *COUNT
022E- A9 00   INITPORTS  LDA #$00
0230- 8D 0B A8           STA @2ACR      ; NO LATCHING
0233- 8D 03 A8           STA @2DDRA     ;2PA7-2PA0 ARE INPUTS
0236- A9 07              LDA #$07
0238- 8D 02 A8           STA @2DDRB     ;3PB2-3PB0 ARE OUTPUTS
023B- A9 04              LDA #$04
023D- 8D 0C A8           STA @2PCR      ;INTERRUPTS
0240- 20 47 02           JSR EN.IEEE    ;ENABLE IRQS
0243- 58                 CLI
0244- 4C 44 02 IDLE      JMP IDLE       ;WAIT REAL FAST
              ;
              ;
0247- 78      EN.IEEE    SEI
0248- A9 83              LDA #$83       ;ENABLE ATN AND IFC
024A- 8D 0E A8           STA @2IER      ;   INTERRUPTS
024D- A9 06              LDA #$06
024F- 8D 00 A8           STA @2IORB     ;NDAC=1,NRFD=ATN
0252- 60                 RTS
              ;
              ;
0253- 48      INTERFACE  PHA  ;SAVE REGISTERS
0254- 98                 TYA
0255- 48                 PHA
0256- 8A                 TXA
0257- 48                 PHA
0258- AD 0D A8           LDA @2IFR
025B- 10 1D              BPL EXIT.INTF
025D- 29 03   IEEE.IRQ   AND #$03       ;WHICH INTERRUPT?
025F- C9 01              CMP #$01
0261- F0 1D              BEQ ATN.IRQ
0263- C9 02              CMP #$02
0265- F0 03              BEQ IFC.IRQ
0267- 4C 7A 02           JMP EXIT.INTF
026A- AD 01 A8 IFC.IRQ   LDA @2IORA     ;CLEAR INTERRUPT
026D- A9 01              LDA #$01
026F- 2C 0E A8           BIT @2IER      ;IEEE ACTIVE?
0272- D0 06              BNE EXIT.INTF        ;EXIT INTERFACE
0274- A6 EA   IEEE.OFF   LDX *SP.IEEE
0276- 9A                 TXS  ;RESTORE STACK POINTER
0277- 20 47 02           JSR EN.IEEE
027A- 68      EXIT.INTF  PLA
027B- AA                 TAX
027C- 68                 PLA
027D- A8                 TAY
027E- 68                 PLA
027F- 40                 RTI
              ;
              ;
0280- BA      ATN.IRQ    TSX
0281- 8E EA 00           STX SP.IEEE    ;SAVE STACK POINTER
0284- AD 01 A8 ATNINIT   LDA @2IORA     ;CLEAR INTERRUPT
0287- A9 05              LDA #$05
```

```
0289- 8D 00 A8               STA @2IORB     ;SET NDAC=0 NRFD=0
028C- A9 01                  LDA #$01
028E- 8D 00 A8               STA @2IORB     ;TURN OFF ATN=NRFD
0291- 8D 0E A8               STA @2IER      ;TURN OFF ATN IRQS
0294- 58                     CLI
0295- A9 00                  LDA #$00
0297- 85 E4                  STA *SEC.ADDRS         ;INIT SEC. ADDRS
0299- 20 EF 02               JSR CYCLE
029C- A5 E2                  LDA *DATA
029E- C5 E3                  CMP *MLA1
02A0- F0 0C                  BEQ DEVICE1    ;BRANCH IF MY ADDRESS
02A2- A9 02       EXIT.IEEE  LDA #$02
02A4- 8D 00 A8               STA @2IORB     ;RELEASE ATN=NRFD
02A7- 2C 00 A8    @15        BIT @2IORB
02AA- 30 FB                  BMI @15        ;WAIT FOR ATN=1
02AC- 10 BC                  BPL IFC.IRQ    ;BR ALWAYS
                  ;
02AE- 20 EF 02    DEVICE1    JSR CYCLE
02B1- 24 E1                  BIT *SIGNALS           ;SECONDARY ADDRESS?
02B3- 10 09                  BPL @3         ;BRANCH IF ATN IS OFF
02B5- A5 E2                  LDA *DATA      ;GET SECONDARY ADDRESS
02B7- 29 0F                  AND #$0F       ;ALLOW 16 SEC.ADDRS'S
02B9- 85 E4                  STA *SEC.ADDRS
02BB- 20 EF 02               JSR CYCLE      ;GET FIRST CHAR.
02BE- A5 E4       @3         LDA *SEC.ADDRS
02C0- 0A                     ASL A
02C1- AA                     TAX
02C2- BD CF 02               LDA SCTABLE,X          ;FIX POINTER TO
02C5- 85 EE                  STA *IND.JMP           ;    SELECTED ROUTINE
02C7- BD D0 02               LDA SCTABLE+$1,X
02CA- 85 EF                  STA *IND.JMP+$1
02CC- 6C EE 00               JMP (IND.JMP)
02CF- 37 03       SCTABLE    .SI SENDASCII          ;NORMAL PRINTING
02D1- 47 03                  .SI DUMPCHRS
02D3- 47 03                  .SI DUMPCHRS
02D5- 47 03                  .SI DUMPCHRS
02D7- 47 03                  .SI DUMPCHRS
02D9- 47 03                  .SI DUMPCHRS
02DB- 47 03                  .SI DUMPCHRS
02DD- 47 03                  .SI DUMPCHRS
02DF- 47 03                  .SI DUMPCHRS
02E1- 47 03                  .SI DUMPCHRS
02E3- 47 03                  .SI DUMPCHRS
02E5- 47 03                  .SI DUMPCHRS
02E7- 47 03                  .SI DUMPCHRS
02E9- 47 03                  .SI DUMPCHRS
02EB- 47 03                  .SI DUMPCHRS
02ED- 47 03                  .SI DUMPCHRS
                  ;
                  ;
02EF- A9 03       CYCLE      LDA #$03
02F1- 8D 00 A8               STA @2IORB     ;NRFD=1 NDAC=0
02F4- 2C 00 A8    @1         BIT @2IORB     ;TEST DAV
02F7- 70 FB                  BVS @1         ;BRANCH IF DAV=1
02F9- 6A                     ROR A
02FA- 8D 00 A8               STA @2IORB     ;NRFD=0 NDAC=0
02FD- AD 01 A8               LDA @2IORA
0300- 49 FF                  EOR #$FF
0302- 85 E2                  STA *DATA
0304- AD 00 A8               LDA @2IORB
```

```
Ø3Ø7- 85 E1              STA *SIGNALS
Ø3Ø9- A9 ØØ              LDA #$ØØ
Ø3ØB- 8D ØØ A8           STA @2IORB     ;NRFD=Ø NDAC=1
Ø3ØE- 2C ØØ A8  @2       BIT @2IORB
Ø311- 5Ø FB              BVC @2         ;BRANCH IF DAV=Ø
Ø313- A9 Ø1              LDA #$Ø1
Ø315- 8D ØØ A8           STA @2IORB     ;NRFD=Ø NDAC=Ø
Ø318- 6Ø                 RTS
Ø319- 2Ø 47 8A  PRINT    JSR OUTCHR     ;PRINT AND INC. COUNT
Ø31C- E6 EØ              INC *COUNT
Ø31E- DØ ØC              BNE RETURN
Ø32Ø- A9 Ø3     ACK      LDA #$Ø3       ;ASCII ETX
Ø322- 2Ø 47 8A           JSR OUTCHR
Ø325- 2Ø 58 8A           JSR INCHR      ;WAIT FOR ACK
Ø328- A9 Ø2              LDA #$Ø2
Ø32A- 85 EØ              STA *COUNT
Ø32C- 6Ø        RETURN   RTS
                ;
                ;
Ø32D- A5 E2     @18      LDA *DATA
Ø32F- 29 7F              AND #$7F
Ø331- 2Ø 19 Ø3           JSR PRINT
Ø334- 2Ø EF Ø2  NEXT     JSR CYCLE
Ø337- 24 E1     SENDASCII BIT *SIGNALS
Ø339- 1Ø F2              BPL @18        ;BR IF ATN=1
Ø33B- A5 E2              LDA *DATA
Ø33D- C9 3F              CMP #UNLISTEN
Ø33F- DØ F3              BNE NEXT
Ø341- 4C A2 Ø2           JMP EXIT.IEEE
                ;
                ;
Ø344- 2Ø EF Ø2  NEXT2    JSR CYCLE
Ø347- 24 E1     DUMPCHRS BIT *SIGNALS
Ø349- 1Ø F9              BPL NEXT2
Ø34B- A5 E2              LDA *DATA
Ø34D- C9 3F              CMP #UNLISTEN
Ø34F- DØ F3              BNE NEXT2
Ø351- 4C A2 Ø2           JMP EXIT.IEEE
Ø354- ØØ                 .BY $Ø
                         .EN
```

SYM to Spinwriter Hardware

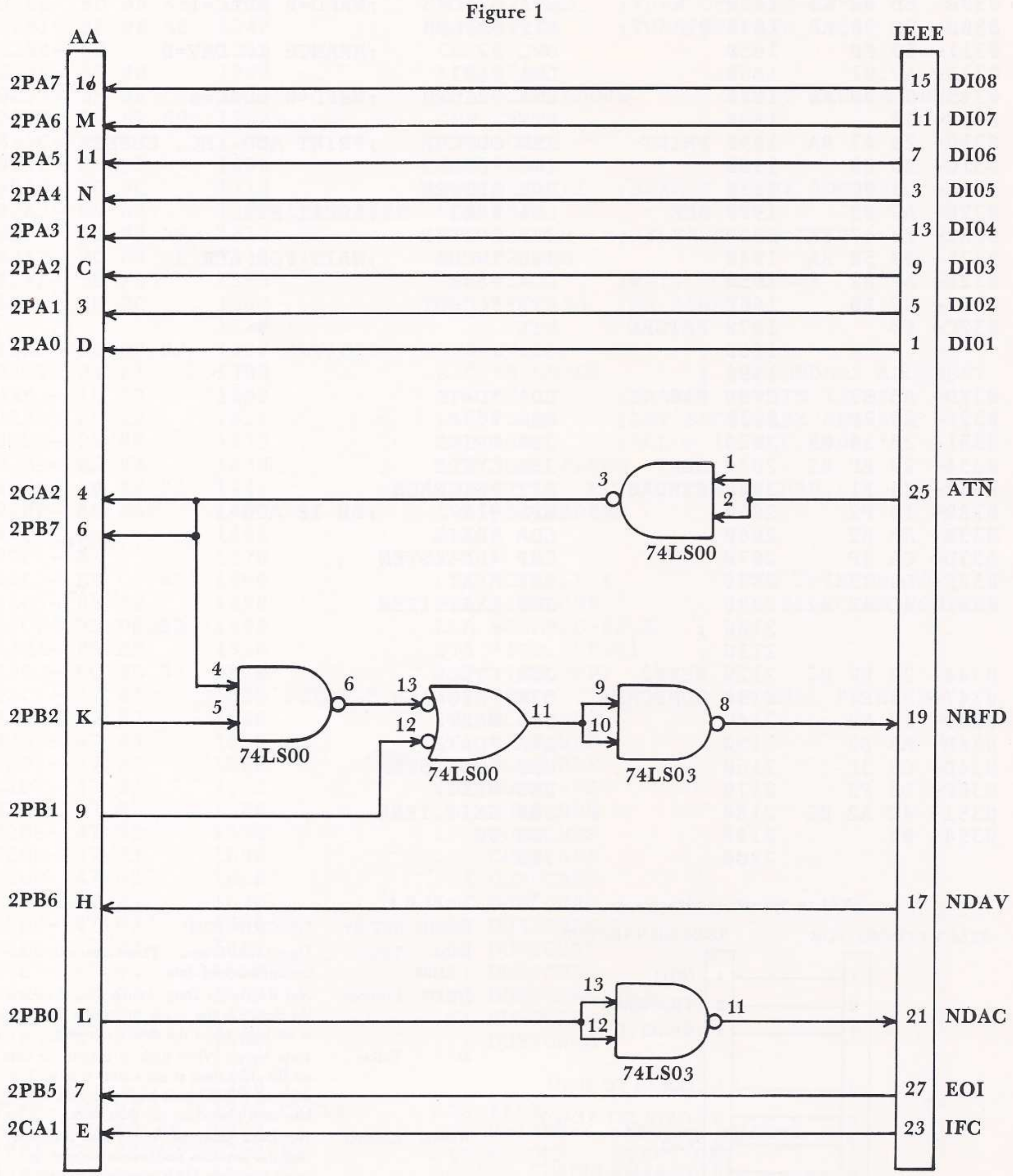


Editor's Note: For those of you who don't have issue 1, we're reprinting these two charts. RCL

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.

Figure 1


AA
IEEE
2PA7 1ɸ 15 DI08
2PA6 M 11 DI07
2PA5 11 7 DI06
2PA4 N 3 DI05
2PA3 12 13 DI04
2PA2 C 9 DI03
2PA1 3 5 DI02
2PA0 D 1 DI01
2CA2 4
2PB7 6
3
1
2
25 ATN
74LS00
4
5
6
13
12
11
9
10
8
2PB2 K
19 NRFD
74LS00
74LS00
74LS03
2PB1 9
2PB6 H 17 NDAV
13
12
11
2PB0 L
21 NDAC
74LS03
2PB5 7 27 EOI
2CA1 E 23 IFC

©

SYM High Speed Tape

Gene Zumchak

The SYM has two different tape formats, the low speed or KIM format, and its own high speed format that can handle 185 bytes per second, which is not bad at all . . . if it works. The high speed format has given problems from the beginning. The new SYM monitor, version 1.1 was changed significantly in the tape routines to overcome the early problems. Also, newer SYMs use a different bias network on the tape input comparator and a fatter (.22 mfd) input coupling capacitor (C16). (Synertek advises that a few users have improved their tape reads by reducing C16, a typical value being .05 mfd.)

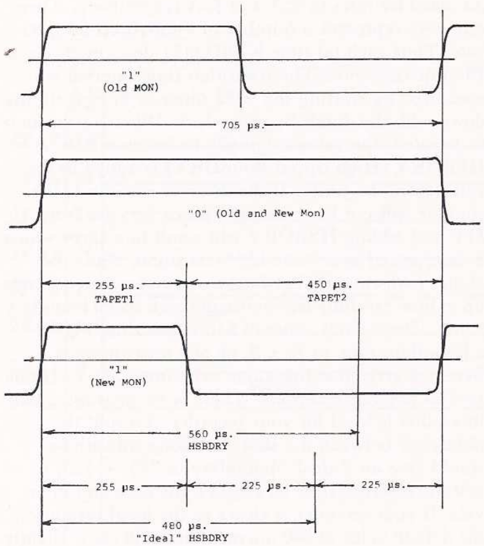
If you have an early SYM and still use the original version 1.0 monitor you won't be able to benefit from this discussion. I recommend very strongly that you obtain the new monitor. It's available from SYM Users Group, P.O. Box 315, Chico, CA 95927, for $16, and includes the resistor mod kit.

Nevertheless, even if you have the hardware mods and the new monitor, there is no guarantee that you will get reliable tape reading. The differences in success appear to be most affected by the tape recorder. Often-time a cheap discount store recorder will give good results when a more expensive name brand unit will not. Frequency response of the recorder does not seem to be a criterion for predicting success. The SYM high speed format, and most high speed techniques depend upon measuring the time interval between transitions on the tape. Misinterpret one transition time and it's all over. The transitions are put on the tape very accurately. However, when the tape is played back, the high frequency components may experience significant phase shifting, affecting the zero crossing positions. Thus the high frequency shifting, and not so much the frequency response, appears to be the culprit. Fortunately, the new SYM monitor has some variables built into the tape routines that allow you to "tweak" the tape read/write programs to accomodate your recorder. These variables are shown in the accompanying figure, reproduced by permission of Synertek.

In the SYM format, the bit period is constant. A "one" is two transitions per bit period, and a "zero" is one transition per bit period. In the original monitor, the two intervals for the one were symmetical. In the new monitor, however, the first interval, (the only one measured) is narrower than the second, máking it easier to distinguish between a short period (one) and a long period (zero). The intervals are specified by variables TAPET1 and TAPET2 which are initilized by reset to $33 and $5A respectively. These numbers represent a number of 5-microsecond intervals. Thus each bit time is $8D (141 dec.) intervals or 705 microseconds. The transition time interval is measured by starting the 6532 timer at $FF, counting down with the divide by eight clock. When a transition is detected, the value originally in location $A632 = HSBDRY (High Speed BounDRY) is added to the value from the timer. If the interval was short, the counter will not have counted down very far from $FF and adding HSBDRY will result in a carry which is interpreted as a "one-bit" transition. Thus the ability to distinguish between a one and a zero depends upon how carefully we choose the high speed boundary value. The default value of $46 (70 decimal) gives a boundary time of 70 x 8, or 560 microseconds. Synertek arrived at this value experimentally by trying several popular recorders. There is no guarantee that this value is ideal for your recorder. To split the difference between the short and long transitions would give an "ideal" boundary of 255 + 225, or 480 microseconds, or 60 ($3C) 8-microsecond intervals. If your recorder is closer to the ideal response, the default value of 560 microseconds will cause slightly narrow zero intervals to be interpreted as ones giving a bad reading. Before I took a look at the numbers, I experimentally determined the value of HSBDRY for my Panasonic recorder to be about $3C. Actually there was quite a range from $40 down to $39, but HSBDRY definitely needed to be smaller. Interestingly, I still can load tapes only over a very narrow range of volume settings.

If indeed it is the phase shifting of high frequency components that affects zero crossings, then perhaps low-pass filtering the tape output before it goes onto the tape would improve performance. Then again, I do need the tone control as high as it will go to give best results. It would seem that with the diode clipping at the input of the comparator, the tape read would be relatively insensitive to amplitude, with a high volume being ideal. However, with my SYM that is not the case. Clearly, a great deal of experimenting can be done pre-filtering tape dump output before it is recorded, and conditioning the playback output before it is decoded.

So far we have discussed only changing the value of HSBDRY to improve our read capability. However, the tape dump parameters TAPET1 and TAPET2 can also be modified. To generate SYM compatible tape, their values should not be changed radically, and their sum should equal $8D. On the other hand, if the sum is changed, the bit time and the corresponding number of bytes per second will change. We can make the tape speed faster or slower, and still read it back with the regular SYM programs by changing HSBDRY correspondingly. Just for kicks, I made TAPET1 $22 and TAPET2 $46, and was able to get fairly reliable loads with HSBDRY $30. This is a byte rate of approximately 250 bytes per second. It may be possible to double the SYM's high speed rate



NAME	LOCATION	DEFAULT VALUE
HSBDRY	$A632	$46 (70 dec.) 350 µs.
TAPET1	$A635	$33 (51 dec.) 255 µs.
TAPET⊘	$A63C	$5A (90 dec.) 450 µs.

and still get good loads. The important thing, however, is to get reliable loads at the regular high speed.

Unfortunatley, there are still a number of problem sources that have nothing to do with SYM hardware and software. You may be using a bad tape. Your recorder may be excessively noisy, or generate motor noise. You might suspect the latter if the Sync display indication occasionally flickers even when set at the optimum volume setting. Sometimes a capacitor (.05 to .1mfd) from the input of the comparitor (pin 3) to ground will solve this problem. To help find other problem sources, a list of guidelines, provided by Synertek, are reproduced at the article's end.

In summary, SYMMERs still having problems with tape loading and using the new monitor may only need to adjust the value of HSBDRY ($A632), thanks to Synertek's forsight in making the tape parameters variables. Remember, however, that this value, and all system RAM is initialized by RESET and will have to be fixed after each Reset.

There is certainly a lot of experimentation that can be done on the SYM high speed tape reading and writing. I hope that the information in this brief article will inspire other SYMMERs to do some investigation. I'm sure that others besides myself will want to hear about any discoveries you make.

Twenty Important Cassette Recording Guidelines

1. Use high quality tape (Maxell UD or equivalent).
2. Use shortest tapes possible. You can shorten tapes to several minutes in length if you enjoy splicing.
3. Use shielded cable between your computer and the cassette recorder.
4. Keep heads and pinch rollers clean.
5. Keep heads aligned for tape interchangability.
6. Avoid recording too close to beginning of tape.
7. Make sure cassette is properly seated in recorder.
8. If you have trouble with a cassette try another. You can have a bad spot on tape or a warped cassette.
9. Highest setting of tone control is usually best.
10. A dirty recorder volume control can cause tape dropouts.
11. Make sure cassette connection plugs make good contact.
12. Rewind cassettes before removing them from recorder.
13. Store cassettes in dust-proof containers.
14. Avoid exposing cassettes to heat or magnetic fields.
15. Before recording, wind cassette to one end and fully rewind.
16. Cassette recorders will give you problems once in a while (They don't like certain cassettes, etc.). If one gives you problems most of the time replace it.
17. Make sure that MIKE plug is connected before recording. On most recorder the TAPE light will glow while recording.
18. You may have to record with the EAR plug out for some tape recorders.
19. Always use AC adaptor with recorder for best results.
20. When a tone control is available, adjust it to the highest possible setting (maximum treble). ©

KIM Rapid Memory Load/ Dump Routine

Bruce Nazarian
1007 Wright Street #3
Ann Arbor, MI 48105

This routine works well for mass entering of stuff like long programs from a hex dump or similar, where you can tell at a glance where any errors in your entries are. A few words of additional explanation about it:

For those users who would rather have a Carriage Return activate the address entry portion and the associated functions, substitute ASCII CR ($0D) at location $010E. This will do the trick and is the same as Markus Goenner's function from his TTY load routine from K.U.N. Thanks go to him for the use of some of his programming techniques.

The directions also indicate that the program will list until it senses a key pressed at the end of a line. This is true, but the user should only use one of the DATA keys on the keypad, not ST or RS.

Finally, the routine will only indicate the stopped address after the user commands RUBOUT thru his terminal. Then the KIM monitor will print the current pointer, which will be the address where it stopped dumping.

If you want the routine to present one line of hex at a time, and wait on a key depression before looping back again and printing another line, make this change:

0147 20 6A 1F JSR KEYIN (Instead of the getkey subroutine)
014A D0 FB BNE 0147
014C EA EA NOP's to fill previous coding

```
0100                           ORG $0100
0100  D8               ENTER   CLD            Clear decimal mode
0101  A9  00                   LDA #$00       Zero out the input buffers
0103  85  F8                   STA INL        Low. . . . .
0105  85  F9                   STA INH        And High. . . .
0107  20  2F  1E               JSR CRLF       Use KIM Subroutine to send functions
010A  20  5A  1E       ADDR    JSR GETCH      Input one character.. (of starting addr)
010D  C9  20                   CMP #$20       Check for go ahead.. (Insert 0D for CR)
010F  F0  05                   BEQ DATA       If yes, load address from buff in pointer.
0111  20  AC  1F               JSR PACK       If no, load character into INL,INH
0114  F0  F4                   BEQ ADDR       ...and loop back again
0116  20  CC  1F       DATA    JSR OPEN       Move INL,INH, to POINTL,POINTH..
0119  20  2F  1E       DECIDE  JSR CRLF       (Saves bytes, doesn't it?)
011C  20  5A  1E       INPUT   JSR GETCH      Now input some Hex for the code...
011F  C9  4C                   CMP #$4C       'L' (Load memory)?
0121  F0  2E                   BEQ LOAD       Yes, branch to LOAD portion (0151)
0123  C9  51                   CMP #$51       'Q' (Dump from memory)?
0125  D0  F5                   BNE INPUT      No, ignore invalid characters;Loop..
0127  A9  0F           DUMP    LDA #$0F       Set up byte counter (16 decimal)
0129  8D  7F  01               STA COUNT      stick it in $017F
012C  20  2F  1E               JSR CRLF       New line, please..
012F  20  1E  1E               JSR PRTPNT     Output the current pointer address
0132  20  9E  1E               JSR OUTSP      ...and space it...
0135  20  9E  1E       GET     JSR OUTSP      ...again...
0138  A0  00                   LDY #$00       Set up Y-Register for Indirect addressing
013A  B1  FA                   LDA (POINTL),Y Load contents of pointed address
013C  20  3B  1E               JSR PRTBYT     ...and print as two hex digits...
013F  20  63  1F               JSR INCPT      Increment the double-byte pointer
```

```
0142  CE   7F  01           DEC COUNT        Decrement the byte counter
0145  10   EE               BPL GET          And loop back if not finished yet
0147  20   6A  1F           JSR GETKEY       After 16th byte, test for end of list
014A  C9   15               CMP #$15         ...and if no key is pressed,
014C  F0   D9               BEQ DUMP         go back and output another 16 bytes...
014E  4C   64  1C           JMP CLEAR        else jump to Clear input buffs..
0151  20   2F  1E  LOAD     JSR CRLF
0154  20   5A  1E  READ     JSR GETCH        Input one character..
0157  C9   0D               CMP #'CR'        ..and if it is a carriage return..
0159  F0   F6               BEQ LOAD         ..let it function, but ignore it..
015B  C9   1B               CMP #'ESC'       ..or if it is "Escape"...go 015F
015D  D0   06               BNE STORE        ..if not, must be valid.. Store it.
015F  20   80  01           JSR STRING       ..else send '? KIM ?' prompter...
0162  4C   64  1C           JMP CLEAR        ..and clear buffers..exit load routine
0165  20   AC  1F  STORE    JSR PACK         Pack character into INL,INH
0168  D0   EA               BNE READ         If packed value is zero, skip it..
016A  20   5A  1E           JSR GETCH        Get second byte of Hex code
016D  20   AC  1F           JSR PACK         ..and pack it also..
0170  A0   00               LDY #$00         Set up for indirect addressing
0172  A5   F8               LDA INL          Bring in packed value..
0174  91   FA               STA (POINTL),Y   .. and store it at pointed address
0176  20   63  1F           JSR INCPT        Increment the double-byte pointer
0179  18                    CLC
017A  90   D8               BCC READ         Branch always..
017C  EA   EA  EA           NOP              Waste some space
017F  [XX]         COUNT    [This location used to hold the variable byte cntr]
0180                        ; Subroutine "STRING" to send KIM prompter
0180                        ORG $0180
0180  A2   0C      STRING   LDX #$0C         Set up X-reg as counter
0182  BD   90  01  STRNG2   LDA TABLE,X      Get character at TABLE + X
0185  20   A0  1E           JSR OUTCH        Ship it out...
0188  CA                    DEX              Decrement the counter
0189  10   F7               BPL STRNG2       Loop is not finished
018B  60                    RTS              Else return to mainline when done
018C  EA   EA  EA           NOP              NOP's to fill

190   20   3F  20  TABLE    .BYTE 'SP,?,SP,
0193  4D   49  4B           M,I,K
0196  20   3F  00           SP,?,NUL,
0199  00   0A  0D           NUL,LF,CR
019C  0D                    CR'
```

Some Instructions To Help It All Make Sense:

1. This routine is set up for an I/O device of the user's choosing, as long as it is fed thru the KIM internal TTY port.. Users with other I/O will have to modify the coding to suit their particular situation.
2. The routine is self-contained on Page One and leaves all other memory free for user programs, but be prepared, as always, to re-read the routine from cassette should the stack overwrite the routine.
3. Execute as follows:
   After loading the coding, a "GO" executed at address $0100 will get the ball rolling.. your terminal should immediately execute a CR/LF sequence and will pause... Begin by typing in the four digit address you wish to start loading, or dumping from.. If you err in typing, just correct by typing in the correct address again, just like the KIM TTY monitor.. A "SPACE" after the correct address is in place will enter that address into the pointer.. The program will again send CR/LF and pause.. now, enter "L" if you wish to use the rapid load routine, or "Q" if you wish a formatted memory dump from your indicated address.. If LOAD was chosen, you may now begin entering data in two-digit HEX and the pointer will be taken care of for you automatically.. a good way to do this is

to enter two hex digits, and then space, as the routine will ignore the packed space character and only enter the valid hex... If DUMP was chosen, the routine will now commence to dump the contents of memory consecutively from your indicated address like this:

0200 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0210 EA EA EA .... etc.

IT WILL LIST CONTINUOUSLY UNTIL YOU PRESS A KEY ON THE KIM KEYPAD AND HOLD IT DOWN AT THE END OF A LINE.. It will then stop and indicate the stopped address.

©

KIM-1 Tidbits

Harvey B. Herman
Chemistry Department
University of North Carolina at Greensboro
Greensboro, N.C. 27412

I have been using KIM for a number of years and wish to share programs which I have developed or modified with the readers of Compute II.

The first item is a modification to the KIM tape verify program from Issue #13 of 6502 User's Notes. This program has a small bug which affects TTY use. The TTY delay characters (CNTL30/CNTH30) are stored in $17F2 and $17F3 and are overwritten by a section (VEB) of the original verify program. Instead of the comforting KIM message on completion of the program, all I got was a meaningless chugging. The following program (origin $300) circumvents the problem by shortening the VEB section so the delay characters remain intact. I now include this in KIM Microsoft BASIC, as the User program, so I can check tapes after a SAVE.

Item 2 is a modification to KIM Microsoft BASIC (serial number 9011) which allows one to append programs on tape to the current one (if any) in memory. Line numbers must be higher in the appended program and cannot overlap. Otherwise the only noticeable change is that one must remember to NEW before LOAD when appending is not desired. I have found this very helpful in conjunction with a renumbering program, written in BASIC (see 6502 User's Notes no. 13, p. 12).

I hope these programs will be found useful and plan to share other tidbits with Compute II readers in the future.

```
                   ;
                   ;KIM TAPE VERIFY PROGRAM
                   ;
                   ;HARVEY B. HERMAN
                   ;
                                    .BA $300
                                    .OS
                   CHKL             .DE $17E7
                   CHKH             .DE $17E8
                   VEB              .DE $17EC
                   LOAD12           .DE $190F
                   LOADT9           .DE $1929
0300- D8           VERIFY           CLD
0301- A9 00                         LDA #$00
0303- 8D E7 17                      STA CHKL
0306- 8D E8 17                      STA CHKH
0309- A2 06                         LDX #$06
030B- BD 16 03     LOADP            LDA PROG-1,X
030E- 9D EB 17                      STA VEB-1,X
0311- CA                            DEX
0312- D0 F7                         BNE LOADP
0314- 4C 8C 18                      JMP $188C
0317- CD 00 00     PROG             .BY $CD $00 $00
031A- 4C 1D 03                      .BY $4C $1D $03
031D- D0 03        PATCH            BNE FAILED
031F- 4C 0F 19                      JMP LOAD12
0322- 4C 29 19     FAILED           JMP LOADT9
                                    .EN
```

```
                   ;
                   ;APPEND MODIFICATIONS TO
                   ;KIM MICROSOFT BASIC
                   ;SERIAL NUMBER 9011
                   ;
                   ;HARVEY B. HERMAN
                   ;
                                    .BA $2785
                   ;ADJUST TAPE LOAD POINTERS
2785- 38           NEWLOAD          SEC
2786- A5 7A                         LDA *$7A
2788- E9 03                         SBC #$03
278A- 8D F5 17                      STA $17F5
278D- A5 7B                         LDA *$7B
                   ;NAIVE HARVEY
278F- B0 02                         BCS SKIP
2791- E9 00                         SBC #$00
2793- 8D F6 17     SKIP             STA $17F6
                   ;ORIGINAL CODE CONTINUES
                                    .BA $2744
                   ;ASSIGN ID 01 TO TAPES
2744- A9 01                         LDA #$01
                                    .BA $2026
                   ;POINTER TO NEWLOAD
2026- 84 27                         .SI NEWLOAD-1
                                    .EN
```

©

The Game Corner

Each issue, as material permits, we'll present at least one game. We prefer games in BASIC, thereby enabling users of the "other" machines ready access. Robert Lock

SYM-1 Depth Charge Game

Hardware requirements for this game are 4K of memory, SYM BASIC and the KTM-2/80 keyboard.

Jack Gieryic
2041 138 Ave. N.W.
Andover, MN 55303

Shortly after I received my KTM-2/80 keyboard a fellow Symmer showed me an article entitled "The Klingons Are Coming" by Mark Herro in the April 1978 issue of **73**. Almost immediately I saw fantastic possibilities for the game using a keyboard capable of X-Y cursor positioning. Within two weeks of spare time I came up with my new version which offers a higher degree of visual stimulation and more of a challenge.

There are no instructions displayed when the program is initiated as all this does is waste memory. Instead I have included a "snapshot" of a game already in progress. An explanation of this will be as good as a set of rules and directions.

Each guess consists of entering three coordinates separated by commas and terminating the entry by hitting the RETURN key. The first number is your east-west coordinate and can vary from 0 to 54. The second number is your north-south coordinate and can vary from 0 to 18. The third number is your depth and can also vary from 0 to 18. In the example shown the previous guess had an east-west value of 25, a north-south value of 10 and a depth of 15. On the surface map a graphics character X is plotted at the surface coordinates. This will provide you with a plot of your guesses as the game progresses. Your previous guess appears at the left end of the second line.

After you enter your guess (hitting the RETURN key) the depth gauge will

```
5 X=Ø:E=27:A=18:B=54:C=18
10 S=5
14 FIRST=Ø:SUCC=Ø
15 GOSUB9ØØ
17 FORL=1TOS
18 PRINTCHR$(E)+"= 5          "
19 PRINTCHR$(E)+"=  ";
20 INPUT"YOUR COORDINATES ARE ";D,M,F
22 PRINTCHR$(E)+"=  "+CHR$(E)+"K"
25 IFFIRST=1THENGOTO37
33 FIRST=1:X=INT(B*RND(1)):Y=INT(A*RND(1)):Z=INT(C*RND(1))
37 IFL=S-3THENPRINTCHR$(E)+"= TENEMY CLOSING IN"
42 GOSUB8ØØ
44 IFD<>XTHENGOTO54
46 IFM<>YTHENGOTO58
48 IFF<>ZTHENGOTO62
50 SUCC=1:L=S:GOTO75
54 IFD<XTHENPRINTCHR$(E)+"=  EAST"
56 IFD>XTHENPRINTCHR$(E)+"=  WEST"
58 IFM<YTHENPRINTCHR$(E)+"= DSOUTH"
60 IFM>YTHENPRINTCHR$(E)+"= DNORTH"
62 IFF<ZTHENPRINTCHR$(E)+"= JDEEPER"
64 IFF>ZTHENPRINTCHR$(E)+"= JTOO DEEP"
65 IFL=S-1THENPRINTCHR$(E)+"= TLAST CHANCE"
67 PRINTCHR$(E)+"=!                 "
68 PRINTCHR$(E)+"=! ";D;M;F
70 NEXTL
75 IFL=STHENGOTO9Ø
76 GOSUB7ØØ
78 S=S+1:PRINTCHR$(E)+"=  YOU'RE HIT...TOO BAD...YOU LOSE  "
80 INPUT"TRY AGAIN?  THEN ENTER A 1 ";T
82 IFT=1THENGOTO14
84 END
90 S=S-1:PRINTCHR$(E)+"=  YOU SUNK THE ENEMY...GOOD GOING  "
92 GOSUB5ØØ
94 GOTO8Ø
500 PRINTCHR$(E)+"R"+CHR$(E)+"G"
505 PRINTCHR$(E)+"="+CHR$(Y+34)+CHR$(X+48);
507 PRINTCHR$(124)+CHR$(124)+CHR$(124)
510 PRINTCHR$(E)+"="+CHR$(Y+35)+CHR$(X+48);
512 PRINTCHR$(124)
515PRINTCHR$(E)+"="+CHR$(Y+35)+CHR$(X+5Ø);
517 PRINTCHR$(124)
520 PRINTCHR$(E)+"="+CHR$(Y+36)+CHR$(X+48);
522 PRINTCHR$(124)+CHR$(124)+CHR$(124)
525 PRINTCHR$(E)+CHR$(1Ø3)+CHR$(E)+CHR$(114)
529 PRINTCHR$(E)+"=! ";
530 RETURN
700 PRINTCHR$(E)+"G"+CHR$(E)+"R"
710 PRINTCHR$(E)+"=%H";
712 FORT=1TO9:PRINTCHR$(124);:NEXTT
```

be cleared. You will "see" your depth charge hit the water (SPLASH). The gauge will slowly descend (the rate is baud rate dependent) to your selected depth and then explode (BLAM). At this point sonar will project information on the screen telling you where to direct your next guess. This information appears in the center of the top line. In the "snapshot" example sonar tells you to go further west on your next guess. This means a number smaller than the previous guess of 25. A more northward guess is indicated by sonar for your second number (something smaller than 10) and the enemy is deeper than 15. A reasonable "next guess" would be 15,5,17.

This process is repeated until you either sink the enemy or else run out of chances and are hit instead. If you do guess the lucky spot then that spot will be surrounded by a box and an ego gratifying message will appear in the upper left of your screen. In order to start the next game enter a 1 and hit the RETURN key. If you are unlucky and fail to find the enemy within the allotted chances then **you** will be hit. This condition will come across loud and clear to you. Again enter a 1 and hit RETURN to start the next game.

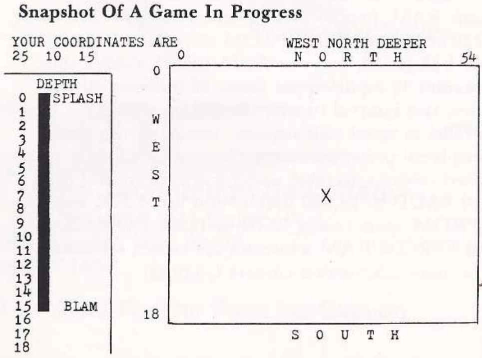
In order to create some challenge the game begins with a limit of 5 guesses. For every game you win this number will decrease by one for the next game. Likewise, if you are hit then this number will be increased by one for the next game. With this feature it becomes nearly impossible to win all the time. We wouldn't want that anyway now would we?

```
714 PRINTCHR$(E)+"'H";
716 FORT=1TO9:PRINTCHR$(124);:NEXTT
720PRINTCHR$(E)+"=&H"+CHR$(124)+CHR$(E)+CHR$(1Ø3);
722 PRINT"YOU"+CHR$(E)+"G"+CHR$(124)+CHR$(E)+CHR$(1Ø3);
724 PRINT"ARE"+CHR$(E)+"G"+CHR$(124)
728 PRINTCHR$(E)+CHR$(1Ø3)
730 PRINTCHR$(E)+"=*CHH   HH  II  TTTTTT"
732 PRINTCHR$(E)+"=+CHH   HH  II  TTTTTT"
734 PRINTCHR$(E)+"=,CHHHHHHH  II    TT"
736 PRINTCHR$(E)+"=-CHHHHHHH  II    TT"
738 PRINTCHR$(E)+"=.CHH   HH  II    TT"
740 PRINTCHR$(E)+"=/CHH   HH  II    TT"
795 PRINTCHR$(E)+CHR$(114)
799 RETURN
800 IFD>54THENGOTO89Ø
801 IFD<ØTHENGOTO89Ø
802 IFM>18THENGOTO89Ø
803 IFM<ØTHENGOTO89Ø
804 IFF>18THENGOTO89Ø
805 IFF<ØTHENGOTO89Ø
810 PRINTCHR$(E)+"G"+CHR$(E)+"R"
812 PRINTCHR$(E)+"="+CHR$(M+35)+CHR$(D+49)+CHR$(126)
814 PRINTCHR$(E)+CHR$(114)+CHR$(E)+CHR$(1Ø3)
817 FORK=1TO19:PRINTCHR$(E)+"="+CHR$(K+35)+"$          ":NEXTK
820 PRINTCHR$(E)+"=$%SPLASH"
830 PRINTCHR$(E)+"G":R=Ø
835 FORK=1TO7
838 PRINTCHR$(E)+"="+CHR$(R+36)+"$"+CHR$(12Ø-K)
839 NEXTK
840 PRINTCHR$(E)+"="+CHR$(R+36)+"$"+CHR$(96);
848 IFR<FTHENR=R+1:GOTO840
849 PRINTCHR$(E)+CHR$(1Ø3)
850 PRINTCHR$(E)+"="+CHR$(R+36)+"% BLAM ";
890 RETURN
900 PRINTCHR$(E)+"H"+CHR$(E)+"J":FORX=1TO2Ø:NEXTX
901 PRINTCHR$(E)+"=5.18":PRINTCHR$(E)+"G"+CHR$(E)+"R"
904 PRINTCHR$(E)+"="+CHR$(34)+" 0000000000F"
906 FORX=1TO19:PRINTCHR$(E)+"="+CHR$(X+34)+"+U":NEXTX
908 PRINTCHR$(E)+"="+CHR$(34)+"ØE"
910 FORX=1TO55:PRINTCHR$(E)+"="+CHR$(34)+CHR$(X+48)+"O":NEXTX
912 PRINTCHR$(E)+"="+CHR$(34)+CHR$(1Ø4)+"F"
914 FORX=1TO19:PRINTCHR$(E)+"="+CHR$(X+34)+CHR$(1Ø4)+"U":NEXTX
916 PRINTCHR$(E)+"=6"+CHR$(1Ø4)+"H"
918 FORX=1TO55:PRINTCHR$(E)+"=6"+CHR$(X+48)+"P":NEXTX
920 PRINTCHR$(E)+"=6ØG"
922 FORX=1TO19:PRINTCHR$(E)+"="+CHR$(X+34)+"ØV":NEXTX
923 PRINTCHR$(E)+CHR$(114)+CHR$(E)+CHR$(1Ø3)
926 PRINTCHR$(E)+"=!FN O R T H"+CHR$(E)+"=7FS O U T H";
930 PRINTCHR$(E)+"=##DEPTH"
932 PRINTCHR$(E)+"=#/Ø":PRINTCHR$(E)+"=!1Ø"
936 PRINTCHR$(E)+"=!"+CHR$(1Ø2)+"54"
938 PRINTCHR$(E)+"=)/W"+CHR$(E)+"=+/E"+CHR$(E)+"=-/S"+CHR$(E)+"=//T"
940 PRINTCHR$(E)+"=-"+CHR$(1Ø5)+"S"+CHR$(E)+"=/"+CHR$(1Ø5)+"T"
942 PRINTCHR$(E)+"=)"+CHR$(1Ø5)+"E"+CHR$(E)+"=+"+CHR$(1Ø5)+"A"
944 FORX=1TO1Ø:PRINTCHR$(E)+"="+CHR$(X+35)+"!";:PRINT(X-1):NEXTX
946 FORX=11TO19:PRINTCHR$(E)+"="+CHR$(X+35)+" ";:PRINT(X-1):NEXTX
995 PRINTCHR$(E)+"=  ";
999 RETURN
```

In Conclusion

When entering the program into your SYM you should use a ? in place of the word PRINT. All zeros in the listing have a slash thru them to distinguish them from the letter O. This does not include the statement numbers as there is no ambiguity here. Sorry to you SYMMERS who do not have a KTM-2/80 keyboard. No, this program can't be run with the KTM-2 keyboard without a large scale change. If any of you KTM-2 owners are interested in adapting this to your SYM-1 then drop me a SASE and I'll get you pointed in the right direction. If any of you others have cursor positioning on your 80 character terminal then you should be able to convert this program with a bit of work.

Snapshot Of A Game In Progress



©

Capute II

Wherein we acknowledge recent goofs. . .

Robert Lock

Those of you who are COMPUTE readers will recognize "**capute II**". A portion of the back page will be reserved for reporting fixes and bugs from previous issues. In COMPUTE, we call it **CAPUTE!**, so it's appropriate to call this one **CAPUTE II!**

If you find a bug, please take the time to drop me a note. The only fixable error I'm aware of was the blotching of Dan McCreary's phone number. In reporting Dan's 1802 Simulators for KIM and APPLE, we somehow have his phone number as 714-281-4748. His CORRECT phone number is 714-281-5758. There apparently was an error in the Issue #3, COMPUTE article on reading PET tapes with the AIM. Hopefully the fixes will be in time for the next issue. ©

6500 Based Single Board Computer

Compas Microsystems, 224 S.E. 16th Street, Ames, Iowa 50010, 515-232-8187 has announced CSB 2 - the newest module in the COMPAS SINGLE BOARD (CSB) family. CSB 2 is a complete stand-alone module based on the popular 6500 processor and features an 86 pin gold plated edge connector which adheres to the ROCKWELL SYSTEM 65 bus standard. This standard is very similar to the MOTOROLA EXORciser standard and EXORciser based cards may be used with CSB 2 with only minor modification.

CSB 2 includes a 6502 processor, 2K bytes of static RAM, four sockets for mounting INTEL 2716 or 2732 or 2732 or 2764 EPROM chips, one VIA (6522), one PIA (6520) and one ACIA (6551). CSB 2 thus provides 30 input/output lines, 10 buffered output lines, two interval timers, a serial to parallel, parallel to serial shift register, input latching on peripheral ports, serial input/output (RS232-C) with speed ranges selectable under software control from 110 BAUD to 19,200 BAUD and up to 32K bytes of EPROM space (using I2764 or I2364 ROM Chips). All EPROM/RAM addresses are switch selectable. For more information contact Compas. ©

More on Modems

Robert Lock

While Eric Rehnke's column list of modem manufacturer's is not intended to be comprehensive, here are a few more names that are involved in personal/industrial/business markets. Even with this update, I'm sure we've left out companies who should be included. If your company is involved in modem manufacture, drop me a note with the appropriate descriptive information.

Novation
18664 Oxnard Street
Tarzana, CA 91356

Among other things, a new direct connect originate/answer modem.

Universal Data Systems
5000 Bradford Drive
Huntsville, Al 35805

New direct connect originate only modem

Datec, Inc.
300 E. Main Street
Carrboro, NC 27510

Originate only and originate/answer line with half/full duplex switchable

And finally, here are the addresses of the Source (TCA) and Micronet (Compuserve):

Compuserve
Personal Computing Division
5000 Arlington Centre Blvd.
Columbus, OH 43220

Telecomputing Corporation of America
1616 Anderson Road
McLean, VA 22102 ©

The wizardry of SYM-1 and the unique "Tubeless Terminal."

Individually, they're powerful stories. Together they're simply unbeatable. The SYM-1 Single Board Computer and the KTM-2 "Tubeless Terminal" Keyboard Modules meet personal, educational and OEM applications head-on. Only from Synertek Systems.

Over 12,000 SYM-1 computers are in use today. In controller applications ranging from radio telescopes to biomedical equipment. With wide acceptance in classrooms throughout the world. And including fast-growing popularity in the consumer marketplace because of its low cost ($239 suggested U.S. retail) and availability of peripheral units like the KTM-2. It all adds up to quick expansion to a versatile computer unit.

You don't need a crystal ball to see the reasons why. SYM-1 can also expand to two language ROMs (BASIC and a Resident Assembler/Editor), a PEX-1 Port Expansion and Connector Kit, and the SRM Static RAM kits. User support is provided by over 500 pages of documentation, distributor field application engineers, an in-house OEM specialist and an independent SYM users group.

The KTM-2 "Tubeless Terminal" Keyboard Modules combine everything but a 5V power supply and CRT (that's why we call it "tubeless") to give you a complete OEM or general purpose personal terminal on a single board. Plus the digital logic of a smart terminal with full ASCII keyboard. The video display logic is placed on the same board space other boards need for keyswitches and encoding logic alone.

With two different versions available, we've got the "Tubeless Terminal," just right for your application – both the 40-character KTM-2 (with a TV it's ideal for timesharing) and the 80-character KTM-2/80.

For complete technical information on SYM-1 and the unique "Tubeless Terminal," contact your area Synertek Systems distributor or call direct (408) 988-5682 or 988-5689.

Synertek Systems manufactures a growing line of evaluation microcomputers and keyboard modules, OEM modules, and development systems for both standard and custom applications.

SYNERTEK SYSTEMS

A SUBSIDIARY OF HONEYWELL
150 S. Wolfe Road
Sunnyvale, California 94086
(408) 988-5682 or 988-5689
TWX: 910-338-0135