ISSN # 0748-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 20 September—October, 1985 $2.50 US

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Production Assistant
Judie Overbeek

Circulation
Donna Carlson

Technical Editor
Lance Rose

Contributing Editor
Ernie Brooner

Contributing Editor
Neil Bungard

Contributing Editor
Bill Kibler

The Computer Journal® is a bimonthly magazine for those who interface, build, and apply microcomputers.

The subscription rate is $14 for one year (6 issues), or $24 for two years (12 issues) in the U.S. Foreign rates on request.

Entire contents copyright © 1985 by The Computer Journal.

Advertising rates available upon request.

To indicate a change of address, please send your old label and new address.

Postmaster: Send address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbis Falls, Montana, 59912.

Address all editorial, advertising and subscription inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

Editor's Page

Eight Bit Systems are NOT Dead!

Various publications have carried death notices for CP/M and S-100, but despite these announcements there is still a lot of life in these old systems. It's true that the selection of new hardware has been limited and the number of new systems sold is small compared with the IBM-PC, but the CP/M systems serve their purpose and many people will continue to use them until there is something with significant operating improvements available.

Some people in the industry have lost track of CP/M because they don't see flashy multi-million dollar advertising for new CP/M hardware and software. Those integrated do-everything program and appliance computer ads are for the non-technical business computer users who aren't sure what hardware or software they really need. The CP/M and other eight bit users are quietly using the thousands of available programs to do what they want without a lot of fuss and excitement.

The new eight bit products are for the programmer or developer who personally works with the computer and is interested in what they can do with computers, while the IBM-PC type products are for business users who are interested in what the computer can do for their business. A recent survey disclosed the interesting fact that a large percentage of computerists who were supplied with an IBM-PC at their work place chose a CP/M or Apple II system for their personal computer at home for non-employment related use. Business users who wanted to continue word processing, spreadsheet, or other business related work at home of course chose a computer compatible with the one at work.

It appears that the IBM-PC is firmly established as the first choice for appliance type business applications (mainly because of the large number of business software programs). The IBM-PC market is primarily for business use software and hardware enhancements, and there is also a large market for programming utilities for the people developing products for this market. The new products for the CP/M market are for multiuser and industrial hardware, programming languages and utilities, system enhancements; and peripherial devices, SBC microcontrollers, and sensors for interfacing to the real world.

Some of the companies supplying new CP/M software are BV Engineering (Engineering programs), Softaid (MTBASIC), Echelon (Z-System enhanced operating system), Softadvances (DSD 80 debugger), Barnes Research & Development (BMON software In-Circuit Emulator), Poor Person Software (utility programs), Dynacomp (Engineering programs), Public Domain Software Center (rents public domain software), Hayden (Structured Assembly Language Programming for the Z-80), SLR Systems (Z80 assemblers and linkers), and Afterthought Engineering (Wiremaster wire wrap and PC layout). There are others, but these are ones with which I am currently working. You'll notice that these are all tools or Engineering programs, not general purpose business programs for the unsophisticated user.

The 8-bit systems that we are using are not the final answer, and we will be changing to 16-bit or 32-bit systems in the near future, but it is not necessary to abandon what we are now using (if it is doing the job) to replace it with other hardware currently being offered. By incorporating new chips and enhanced operating systems we can continue to use what we have with little or no disadvantage until we upgrade to equipment with improvements which are not ready at this time.

Hitachi HD64180 Superchip

One of the developments which will enable us to greatly improve the performance of our CP/M systems is the new HD64180 CPU which is upward compatible with the Z-80 and 8080 instruction sets. I am still waiting for more information, but it has MMU (Memory Management Unit) control of 512K byte physical address space; two channel DMAC with memory-memory, memory-I/O, and memory-memory

(continued on page 36)

Build the Circuit Designer 1 MPB
Part 1: Designing a Single Board Computer

by Neil Bungard

Introduction

Since the microcomputer revealed itself to the public in the early 70's its applications have branched into two distinct catagories:

1. Computer systems
2. Controllers

Computer systems are primarily used to do iterative numerical calculations and store relatively large retrievable data bases. They typically use higher level languages, and the system electronics is complicated.

A microprocessor used as a controller typically does not spend much time doing numerical calculations. Its time is spent making decisions, inputting and outputting, and the majority of the programming on these systems is done in assembly language.

Circuit Designer-1, MPB is a single board computer which is specifically suited for control applications. It utilizes an Intel 8035 CPU and has the following "on board" capabilities.

- 4K of external program memory.
- 64 bytes of data memory on the CPU.
- ¼K of external data memory.
- Ability to generate 256 I/O device and/or control bytes.
- 8 decoded input device pulses.
- 8 decoded output device pulses.
- 12 bit address bus interface.
- 8 bit bi-directional data bus interface.
- 8 bit parallel I/O port interface.
- CPU control signal interface.
- Software controlled event counter.
- Software controlled interval timer.
- Single step capability.
- Power up reset.

The circuit Designer-1 MPB is truely a smart, single board control device and with a simple memory circuit, which will be presented in this article, CD-1 becomes an affordable dynamic training tool for understanding and using chip level microprocessor techniques. Alternatively, one can use EPROMs or EEPROMs, an inexpensive personal computer, and a commercially available 8048 assembler to utilize the CD-1 as a cost effective development system. This configuration will be discussed in a subsequent article.

Circuit Description

The simplest method of understanding any system is to break the system into its component parts, gain an understanding of their operations, then determine the interaction of the component parts within the system. CD-1 is divided into nine component parts which should complete a functional description of its "on board" capabilities. A complete schematic diagram is shown in Figure 10.

The CPU. The heart of CD-1, MPB is the CPU. The CD-1 utilizes Intel's 8035 CPU which is a member of the 8048 family. The 8035, 8048, and 8748 are pin compatible microprocessors with differences only in the type and amount of resident program memory. The 8048 has 1K of masked read only memory which is programmed at the factory. The 8748 has 1K of electrically programmable read only memory, and the 8035 has no internal program memory. A pin diagram of the 8035 is shown in Figure 1. Available on the CPU is an 8-bit bi-directional data bus, two 8-bit parallel I/O ports, five input control signals, five output control signals, two crystal connection inputs, and four power connection pins. The function of these individual pins will be discussed in detail as they are used in the periphial circuits of the CD-1.

Also resident on the 8035 is an 8-bit accumulator, a 12-bit program counter, and 64 bytes of scratch pad memory. The scratch pad memory is internal read/write memory and consists of 16 8-bit general purpose registers, a 16 byte stack, and 32 bytes of general storage memory.

The Multiplexed Address Bus. You may have noticed the absence of address pins on the CPU. The 8035 uses bus multiplexing to derive its address lines, which reduces the number of pins required on the CPU. Refer to Figure 2 for the address multiplexing circuit. For each external memory operation the CPU must latch the address of the current memory read, or memory write, operation. The eight lower order bits of the address are made available on D_0 through D_7, and the four higher order bits are placed on $P2_0$ through $P2_3$. When the address information is stable, the CPU sends out a negative going pulse (ALE) which latches the address into two 74LS373's. Once latched, the CPU uses this address to access either program or data memory.

Program Memory. The program memory is the only memory space that can contain program instructions. The 8035 multiplexes 12 program memory address lines (A_0 through A_{11}). This means that the CPU is capable of addressing 4096 (2^{12}) memory locations. Refering to Figure 3, A_0 through A_{10} are obtained directly from the 74LS373 address latches. Address bit A_{11} is decoded via a 74LS32 and a 74LS86 to determine which 2K program memory block (which 2716) is to be addressed. $\overline{\text{PSEN}}$ is the pulse which strobes the instruction into the CPU. When $\overline{\text{PSEN}}$ goes low it will be "ORed" with A_{11} and $\overline{A}_{11}$. The result of this "ORing" will output a low pulse to the $\overline{\text{CS}}$ on the appropriate program memory. The active memory will then supply an 8-bit word which is strobed into the instruction register via the data bus.

There is a subtle characteristic concerning the program memory which you should be aware of. **The CPU cannot write into program memory.** Program memory is read only memory (ROM). This means that if you wish to write dynamic programs into the CD-1's program memory that it must be done via direct memory access (DMA). This will be explained in detail later in this article.

As I have mentioned before, program memory is used to store CPU instructions. The 8035 retrieves the instructions from program memory by the following sequence of events:

1. The 8035's internal program counter contains the address of the next instruction to be fetched.
2. The CPU makes available the contents of the program counter to the address bus multiplexer (two 74LS373's) via D_0 through D_7 and $P2_0$ through

P2$_3$, and latches this address information into the 74LS373's by pulsing $\overline{ALE}$ low.

3. With the address latched into the 74LS373's, it is available on the address inputs of the program memory. The CPU then pulses $\overline{PSEN}$ low and the instruction contained in the applied address is strobed into the CPU's instruction register and executed.

External Data Memory. External data memory is read/write memory. It has its own set of read/write control signals generated by the CPU, $\overline{RD}$ and $\overline{WR}$ respectively, and it has a unique set of instructions for its operation. The CPU's program counter cannot be set in this memory space so that the data placed in this space cannot be used as program instructions by the CPU. This memory is used primarily for data storage. Refer to Figure 4 for the circuit diagram of the data memory. When the CPU wants to read data from, or write data to, the data memory it must first output a memory address, as it did for the program memory. The extermal data memory uses only the eight lower order address lines so that a maximum of 256 (2^8) memory locations can be accessed. Once the address is latched into the 74LS373 it goes directly to the two 2112's (256 × 4 random access memories). If the CPU wishes to do a memory read it will pull $\overline{RD}$ low which will pull $\overline{CS}$ on the 2112's low, causing the 2112's to be activated. With $\overline{RD}$ active low, $\overline{WR}$ will be high causing the memories to be in the read mode. Since the 2112's are 256 × 4 bit memories, two 2112's are required to create 8-bit words. The 2112's place their 4 bits each on the data bus, and the CPU strobes this data into its accumulator. If the CPU wishes to do a memory write it will place the data on the data bus and pull $\overline{WR}$ low. With $\overline{WR}$ low the memories are in the write mode. A $\overline{WR}$ low also pulls $\overline{CS}$ on the 2112's low activating the memories and writing the data into them. On the CD-1 only the upper 248 of the 256 external data memory locations can be utilized as memory. The eight lower locations are used for I/O device code generation. This is explained in detail below.

The I/O Device Codes. Communication with external input and output devices are accomplished by two methods on the CD-1. The 8035 can communicate with external devices through its parallel I/O ports, or it can

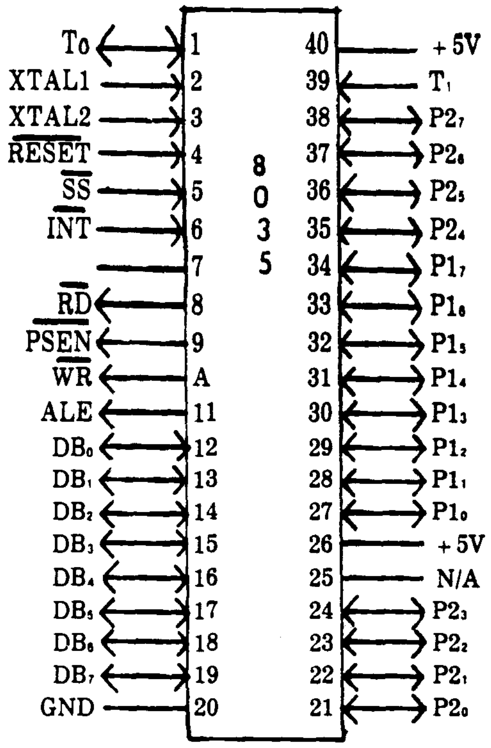


Figure 1: Pinout of the 8035/8048/8748 C.P.U.

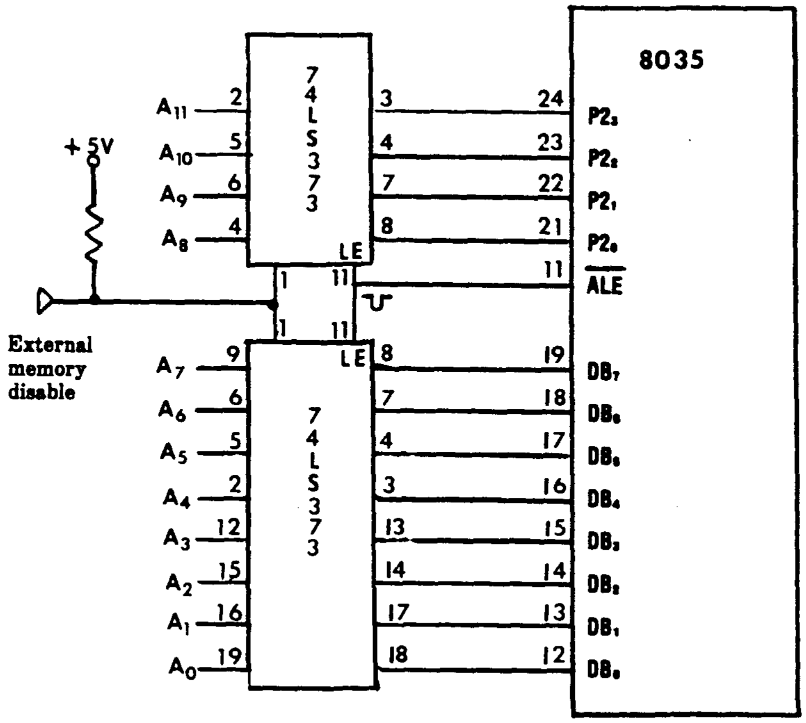


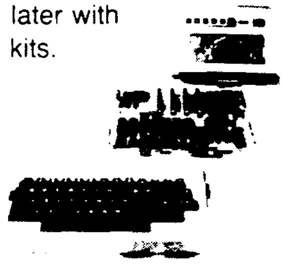
Figure 2: Address Multiplexing Circuit.

GEMINI
THE AUTONOMOUS ROBOT

IS NOW PRICED FOR EVERYONE!

Buy each subassembly as a kit or factory assembled and create your own GEMINI Robot.

Or, for convenience, start with GEMINEX our starter kit, and expand to GEMINI later with upgrade kits.



Either way,
Buy a piece of tomorrow
TODAY!

CALL or WRITE For Our FREE Brochure.

arctec|systems
9104 Red Branch Road
Columbia, Maryland 21045
(301)730-1237
Telex 87-781

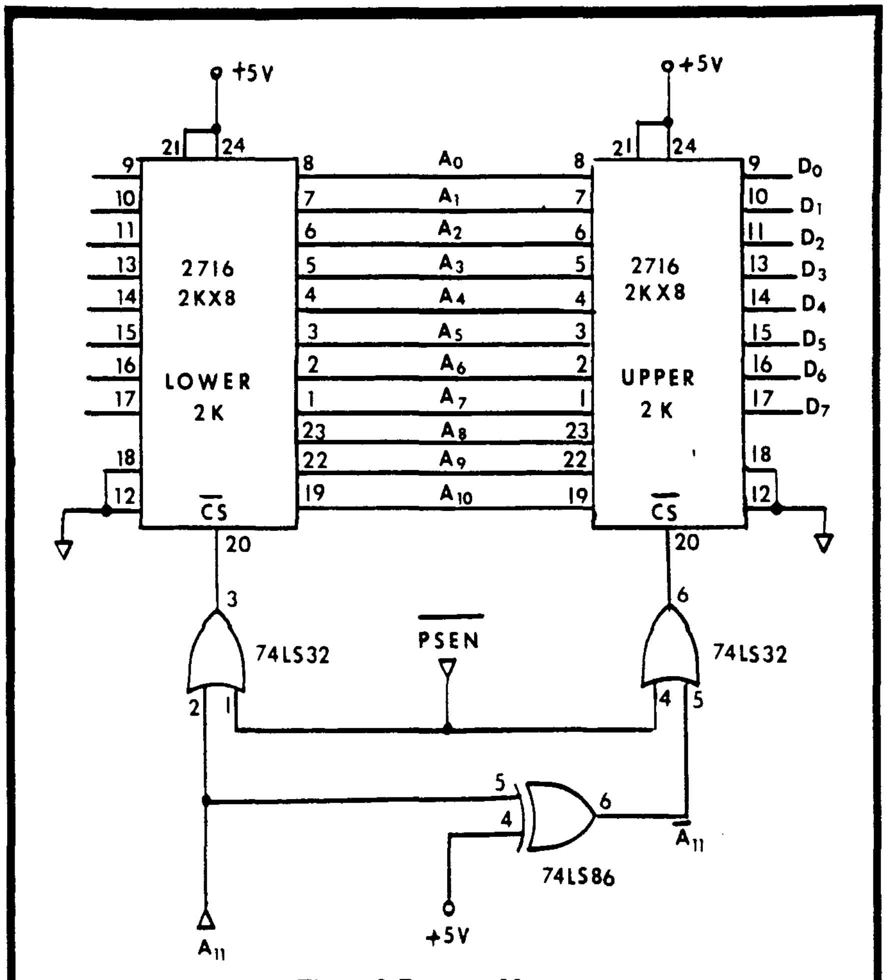


Figure 3: Program Memory.

use memory mapped I/O (MMIO). When using MMIO the 8035 must use its external data memory space. This means that the external data memory must share its 256 possible memory locations with the I/O devices. The circuit for accomplishing MMIO on the CD-1 is shown in Figure 5. When performing MMIO, the CPU is actually doing a memory read or a memory write operation to external data memory.

Address lines A_0 through A_2 of the external data memory are decoded by two "3 to 8 line" decoders (74LS138's). $\overline{WR}$ from the CPU is connected to the enable of one of the 74LS138's and $\overline{RE}$ is connected to the enable of the other 74LS138. The remaining address lines are decoded through the 74LS260 and if any of these address lines are high, both 74LS138's are disabled. This allows memory locations 000H through 007H to be uniquely decoded on the outputs of the 74LS138's. If a memory read is being performed $\overline{RE}$ will be active enabling the "input device code" 74LS138 and one of the eight input device code lines will then go low. This negative going pulse will be used to strobe information onto the data bus from an external device. If the CPU is performing a memory write, $\overline{WR}$ will be active and the "output device code" 74LS138 will be enabled, generating a unique device code pulse which is used to strobe the data being output by the CPU into an external device.

One important characteristic must be considered when performing MMIO operations with the CD-1. **The CPU will perform external data memory operations and generate device code pulses simutaneously.** This will cause bit conflicts on the data bus when input device operations are performed for all but the eight "on board" device codes (the external data memory is automatically disabled for these eight). For the remaining 248 possibilities it is the responsibility of the external devices to disable the external data memory's output while the devices are writing to the CPU. This is done by placing a logic "1" on the input control pin "Memory Disable" of the external memory (Figure 4).

Eight input and eight output device code pulses are generated "on board" by the CD-1, and with additional circuitry as many as 256 I/O devices may be controlled directly.

8-Bit Parallel I/O Port. The 8035 has two 8-bit parallel I/O ports (see Figure 1). I/O port 1 is divided into two 4-bit words. The four lower order bits $P1_0$ through $P1_3$, are used to output the four higher order address bits, A_8 through A_{11}. These bits are output to the address bus multiplexing circuit (see Figure 2). The four higher order bits of port 1, $P1_4$ through $P1_7$ are used by the priority interrupting circuit which will be discussed later. Since I/O port 1 is used for "on board" operations it is not available for interfacing on the CD-1. Parallel I/O port 2, however, is an interfacing option. The eight pins of I/O port 2 on the CPU go directly to an interface socket on CD-1, and I/O port 2 has a unique set of inputting and outputting instructions to control its operation. The major characteristic to consider when using I/O ports 1 or 2 is that they are quasi-bidirectional. This implies the following characteristics:

1. When parallel data is output on I/O port 1 or 2 the data is automatically latched and held on the port by the CPU until the next operation.
2. When using an I/O port for input you must first set the inputting bits to a logic 1 via an output instruction. The only way external logic can input data to I/O port 1 or 2 is by pulling individual pins from a logic 1 to a logic 0. External logic cannot create a high level on any pin of I/O port 1 or 2 once it has been set low via an output instruction.

Eight Level Priority Interrupt. The interrupt function is typically one of the most interesting circuits in control computer application. The CD-1 is no exception, as it contains an eight level priority interrupt capability. The priority interrupt scheme of the CD-1 is both software and hardware dependent. The circuit for the priority interrupt scheme is shown in Figure 6, and the software flow chart is shown in Figure 7.

Requesting an interrupt is accomplished by first executing an external interrupt enable instruction (the interrupts on the 8035 are always disabled when the CPU is reset). Next the interrupting device must place a logic 0 on one of the CD-1's interrupt request inputs ($\overline{INT_0}$ through $\overline{INT_7}$). The interrupt request lines are "ANDed" through a 74LS30, and when any interrupt line goes low, a logic 0 is applied to the $\overline{INT}$ pin (Pin 6) on the CPU

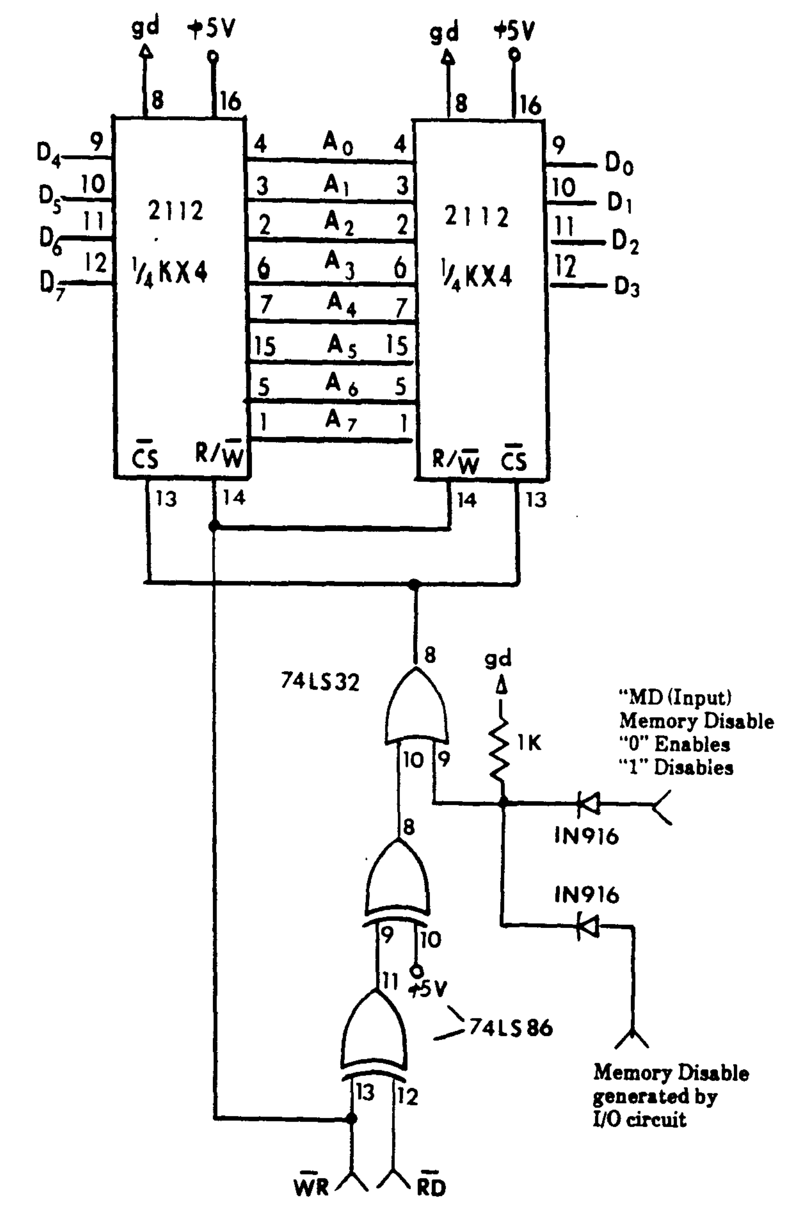


Figure 4: External Data Memory.

MODEL 100 C COMPILER

Now you can write efficient programs for your TRS-80 model 100 with ease. Or, learn the essentials of C programming while traveling!

C/100 · THE "PORTABLE" C COMPILER

Cassette version	$49.00
Disk/Video interface version	$59.00
Model II version (run on mod II, then download object code to model 100)	$79.00
Model III version (as above for Mod III)	$79.00

Write or call for information on other TRS-80 software.

MODELS II, 12, 16
MODELS III, 4

TRS/C C COMPILER
Full K&R with source to the function library. UNIX compatible $85.00

ZSPF EDITOR
SPF, the choice of most mainframe programmers, is now available for Z80 machines. And it's panel driven so you can customize it! $75.00

business utility software
109 minna ste 423 san francisco ca 94105
(415) 397-2000

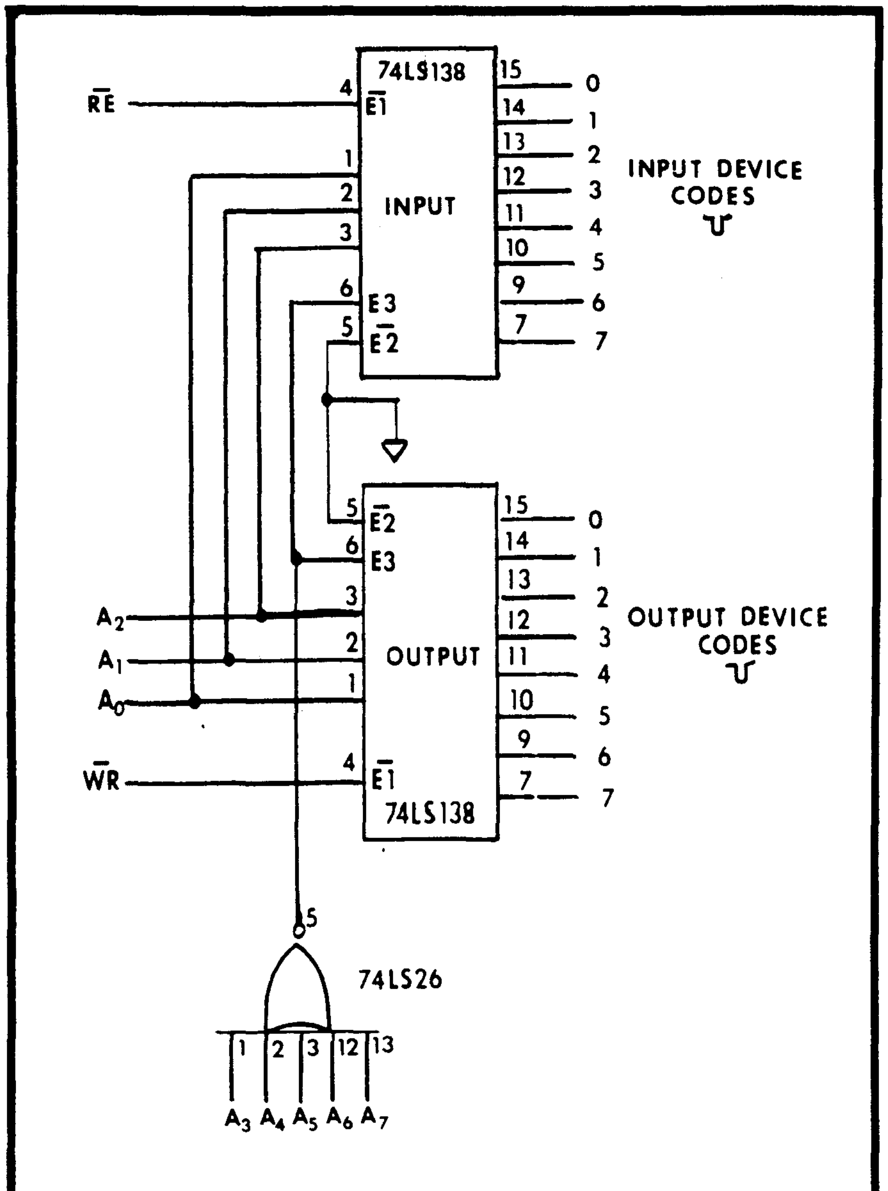


Figure 5: Memory Mapped I/O.

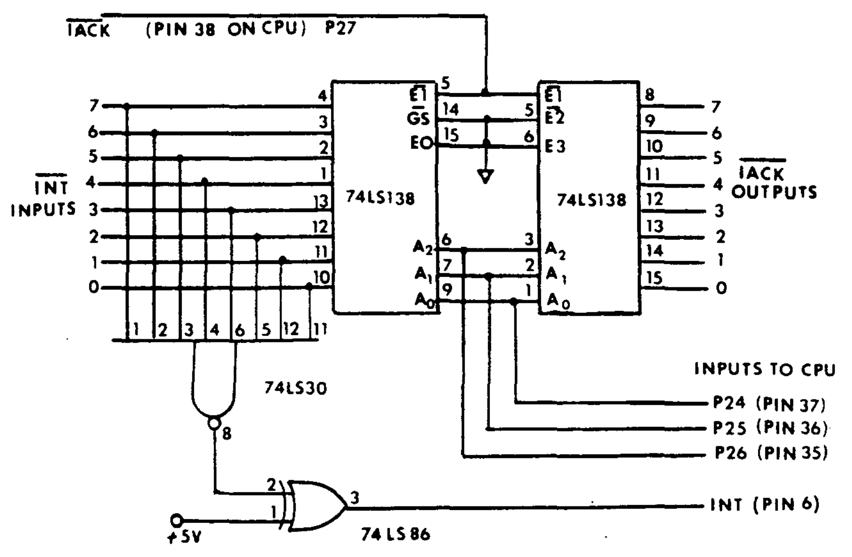


Figure 6: 8 Level Priority Interrupt.

If the CPU sees a logic 0 on its $\overline{INT}$ input, and the external interrupts have been enabled via a program instruction, it will react in the following manner:

1. The 8035 will complete the instruction it is currently executing.
2. The 8035 will execute a call instruction to program memory location 003H.

The software routine for determining interrupt priorities can be placed at location 003H or it can be placed elsewhere in program memory and a jump instruction placed at 003H directing the CPU to the priority interrupt routine.

Referring to Figure 6, the interrupt request lines are tied to the inputs of a 74148. The 74148 is an 8 line to 3 line octal priority encoder, and its truth table is shown in Table 1. Referring to Table 1, pins A_0 through A_2 on the 74148 output a binary number which represents the highest priority input which is set to a logic 0. A_0 through A_2 are also connected to the four highest order bits of I/O port 2 on the 8035, and upon being interrupted the CPU is instructed to immediately input I/O port 2 to the accumulator. All bits except bits 4, 5, and 6 are then masked out, and comparisions are executed in order to branch to the appropriate service routine. A program listing of the procedure is shown in Figure 8. Immediately after the branch is accomplished the service routine should acknowledge that the interrupt has been received. This is done via a negative going pulse on the appropriate interrupt acknowledge ($\overline{IACK}$) output of the CD-1 (see Figure 6). The $\overline{IACK}$ is generated in the following manner:

A 3-bit binary word representing the service routine being executed is output on A_0 through A_2 of the 74148, and is also present on A_0 through A_2 of a 74LS138 (a 3 line to 8 line multiplexer). The truth table of the 74LS138 is shown in Table 2. With a 3-bit binary word on A_0 through A_2 of the 74LS138, the CPU can pull $\overline{IACK}$ low by outputting a logic 0 on P_7 of I/O port 2. P_7 of I/O port 2 is tied to the enable pin of the 74LS138 and when this line goes low, the interrupt is acknowledged. The appropriate $\overline{IACK}$ ($\overline{IACK_0}$ through $\overline{IACK_7}$ is tied to the interrupting device and signals the device to remove

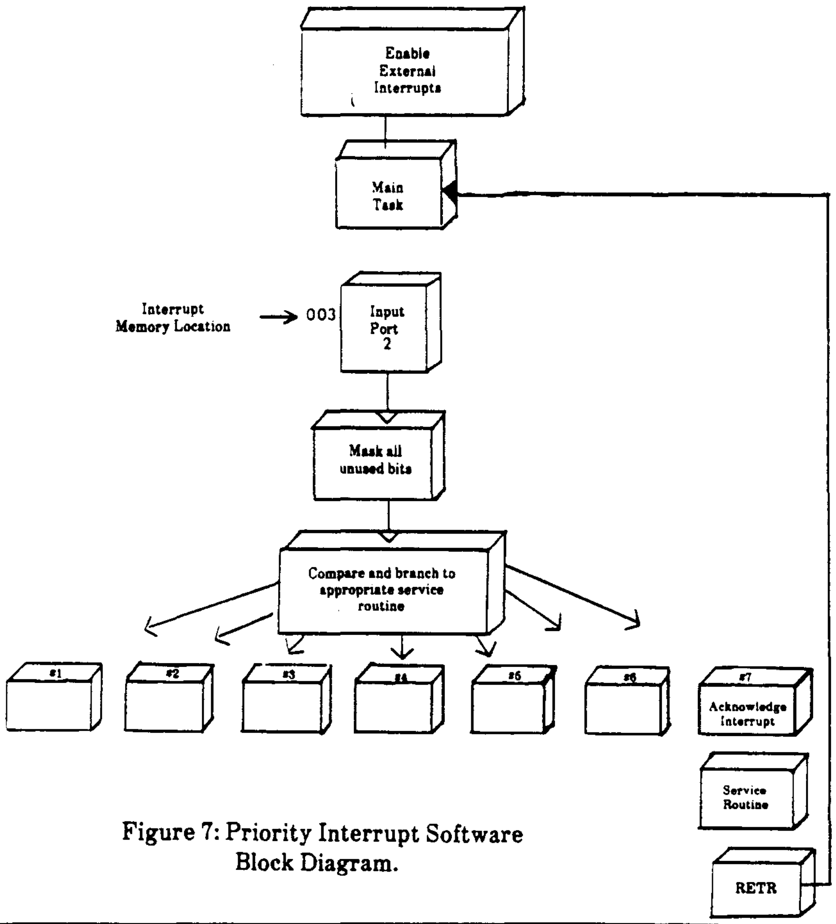

Enable External Interrupts
Main Task
Interrupt Memory Location
003
Input Port 2
Mask all unused bits
Compare and branch to appropriate service routine
#1
#2
#3
#4
#5
#6
#7
Acknowledge Interrupt
Service Routine
RETR

Figure 7: Priority Interrupt Software Block Diagram.

Address	Machine Language	Assembly Language	Comments
03	23	MOV A, # Data	:Set INTA Lo
04	70	(Data)	
05	3A	OUTL P2, A	:Set P2 for input
06	0A	IN A, P2	:Input P2 (interrupt identification)
07	A9	MOV R1, A	:Save interrupt identification
08	23	MOV A, # Data	
09	F0	(Data)	:Set INTA Hi
0A	3A	OUTL P2, A	
0B	F9	MOV A, R1	:Reclaim interrupt identification to accumulator
0C	D3	XRL A, # DATA	
0D	00	(Data)	:Is interrupt identification 00h?
0E	C6	JZ	:If so jump to service :If not
0F	50	(Address)	:Reclaim interrupt identification, if so
		. . .	
		. . .	
		. . .	:Do the same for all 8 possible interrupting devices
		. . .	
		. . .	
		. . .	
2E	F9	MOV A, R1	:Reclaim interrupt identification
2F	D3	XRL A, # Data	:Is Interrupt identification 074?
30	07	(Data)	
31	C6	JZ	:If so jump to service :If not
32	C0	(Address)	
33	93	Retr	:Return

Figure 8: Program listing of Branch To Service Routine.

74148 Truth Table

INPUTS									OUTPUTS				
E1	Ø	1	2	3	4	5	6	7	A2	A1	AØ	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

Table 1

74LS138 Truth Table

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2	C	B	A	YØ	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

NOTES: (1) G2=G2A + G2B
(2) H=High level, L=Low level, X=Irrelevant

Table 2

Typically the advantages of interrupts are:

1. The Computer can execute its main task and not be concerned with polling its servicable devices.
2. Interrupts should have a speed advantage over polling devices. This advantage may or may not occur using the CD-1 since the priorities of the interrupting devices are determined via a software routine. However if a single interrupt is to be handled by the CD-1, then the service routine for that single device may be placed at memory locations 003H and the speed advantage over polling a particular device can be realized.

Programmable Event Counter & Interval Timer. A programmable event counter and interval timer are resident on the CPU of the CD-1, which means that no external circuitry is required to support these functions on the CD-1. The 8035 contains a counter/timer register which is accessed via the accumulator and initial values are loaded into this register with specific program instructions. The counter/timer is also started and halted using program instructions and a summary of the counter/timer instructions and their functions are shown in Table 3.

High-to-low logic transitions on pin 39 of the CPU increment the counter/timer register. Furthermore, when the counter/timer register is incremented from FFH to 00H, two events occur simultaneously:

1. A counter/timer interrupt request is generated which initiates a call to program memory address 007H.
2. A "time-out" flag is set, which may be tested by executing a JTF branch-on-condition instruction.

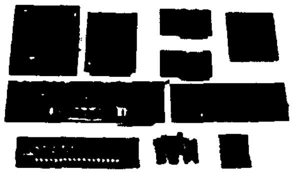
The counter/timer may be operated as a timer by executing a STRT T instruction. In this case the internal system clock increments the timer register every 80 microseconds, assuming a 6 MHz crystal. The counter/timer may also be operated as a counter by executing the STRT CNT instruction, and when this is done, high-to-low transitions on the signal input at T1 (pin 39 on the 8035) increments the counter. The minimum time interval between high-to-low transitions on T1 is 7.2 microseconds, and there is no maximum time restriction between the high-to-low transitions. Once T1 goes high it must remain high for at least 100 nanoseconds.

JFT	Jump on current page if timer has timed out, that is, if timer flag is 1. The timer flag is reset to zero by this instruction.
MOV A,T	Read timer/counter.
MOV T,A	Load timer/counter.
STOP TCNT	Stop timer/counter.
STRT CNT	Start counter.
STRT T	Start timer.

Table 3: Summary of Counter/Timer instructions and functions.

The Single Step Circuit. A single step circuit has been included on the CD-1 so that the processor can execute a single program instruction at a time.

MICROCOMPUTERS AND INTERFACES



We have six singl[illegible]rd computers, two video boards an[illegible]faces for the IBM-PC and AP[illegible]ers. You can use our pr[illegible] security systems, heat control, light cont[illegible] automated slide sho[illegible], irrigation systems, [illegible]tems, automated p[illegible]ot control just to n[illegible]available. For catalog call [illegible]ite to:

JOHN BELL ENGINEERING, INC.
400 OXFORD WAY
BELMONT, CA 94002
(415) 592-8411

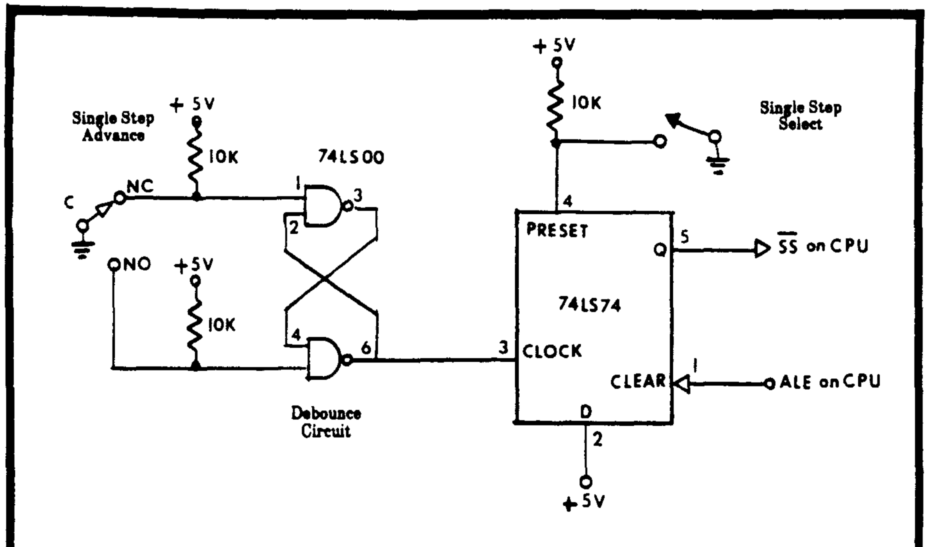


Figure 9: Single Step Circuit.

The single step circuit is shown in Figure 9.

If you do not want the processor to single step, the "preset" input must be grounded via the SS select switch. This forces the "Q" output of the 74LS74 high, and instructions will execute at normal operating speeds. If you wish to single step through the program, you must unground the "preset" input of the single step circuit, which will allow the "clear" input on pin 1 of the 74LS74 to become active. ALE from the CPU is connected to the "clear" input of the single step circuit, and consequently, when ALE goes low the output "Q" is also driven low; thus the $\overline{SS}$ input on the CPU goes low. Now when ALE next goes high, the CPU will maintain it high until the SS advance is depressed, and instruction execution is halted. Depressing SS advance creates a low-to-high transition on the "clock" input of the 74LS74 which forces the SS input on the CPU high, thus allowing the processor to advance one program instruction. It should be noted that while the CPU is stopped between instructions in a single step mode, the current program counter contents are output via the data bus (D_0 - D_7) and $P2_0$-$P2_3$.

Single stepping is a valuable program debugging aid, and allows the user to cease the CPU's execution after each instruction and test conditions within the circuit under investigation.

Conclusion

In conclusion, the material presented in this article will enable the reader to construct, and understand, the CD-1 single board computer. In addition, I

IC Number	Type	Power Connection Pins +5V	GND
IC1	8035	40	20
IC2	74LS373	20	10
IC3	74LS373	20	10
IC4	2116	24	12
IC5	2116	24	12
IC6	74LS260	14	7
IC7	74LS138	16	8
IC8	74LS138	16	8
IC9	74LS32	14	7
IC10	2112	16	8
IC11	2112	16	8
IC12	74LS86	14	7
IC13	NA	NA	
IC14	74LS00	14	7
IC15	74LS74	4	11
IC16	74148	16	8
IC17	74LS30	14	7
IC18	74LS138	16	8

would like to emphasize that although this article presents the 8035 CPU in a single board configuration (CPU and several support IC's), the real virtues of the 8035/8748 lie in its minimum component configuration. The minimum componenent configuration for the 8035 would be the 8035 itself, and one address latachable memory IC. The minimum component configuration for the 8748 would be the 8748!

These "minimum systems" would not be extremely versatile, of course, but the point is that with no additional hardware, switches and light emitting diodes could be connected to the CPU, a program entered, and the system could perform a useful function. Most CPUs are not designed to be used in this fashion, and consequently require a greater number of support ICs.

In the next issue of *The Computer Journal* we will look at the instruction set of the 8035, and explain how to use the CD-1. In addition, we will discuss the idea of using read/write memory in the 8035's program memory space to enable you to write and execute simple programs using the CD-1 single board controller. ■

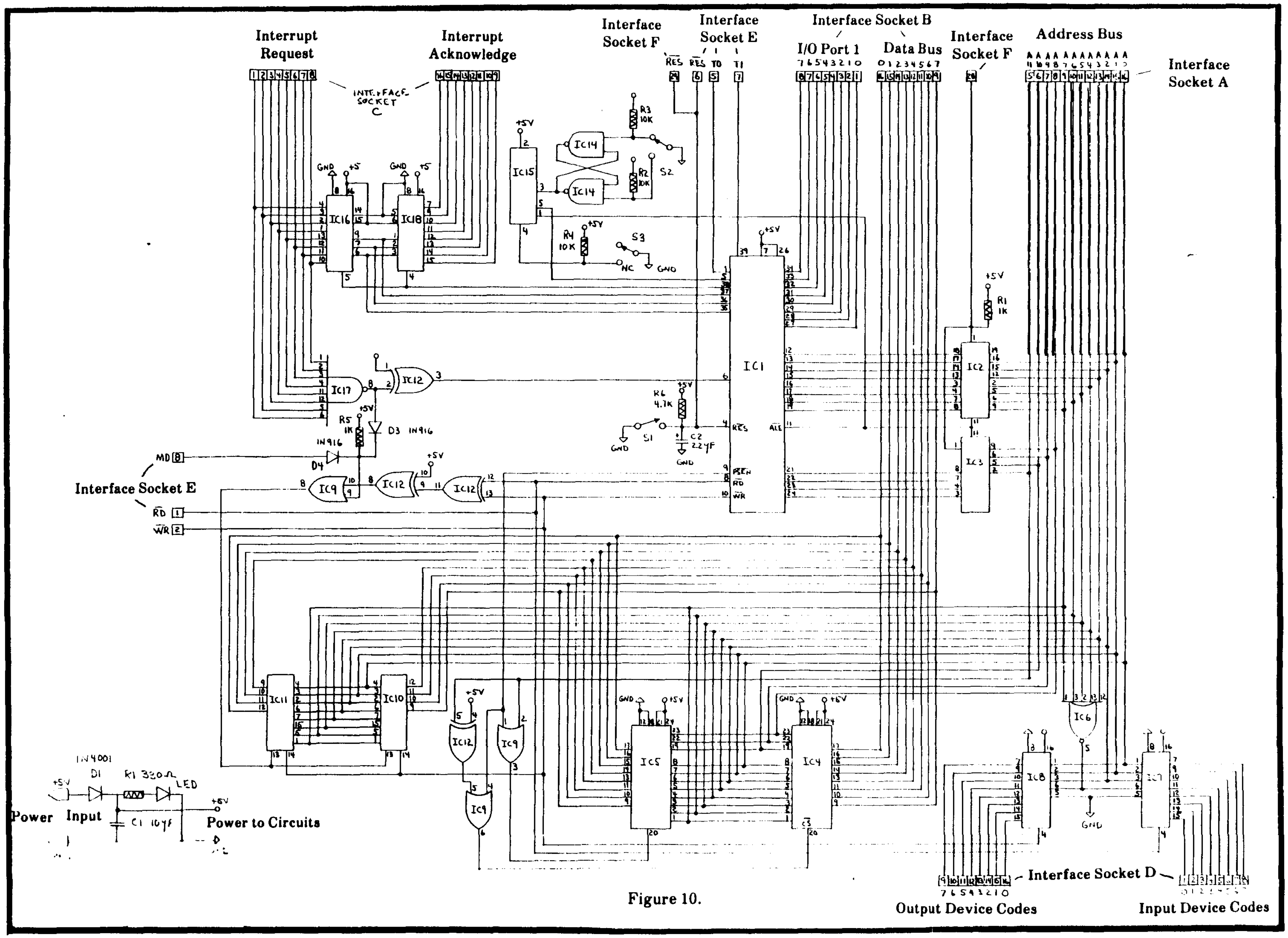

Interrupt Request
Interrupt Acknowledge
INTERFACE SOCKET C
Interface Socket F
Interface Socket E
Interface Socket B
I/O Port 1
Data Bus
Interface Socket F
Address Bus
Interface Socket A
RES
RES
T0
T1
IC1
IC2
IC3
IC4
IC5
IC6
IC7
IC8
IC9
IC10
IC11
IC12
IC14
IC15
IC16
IC17
IC18
R1 1K
R2 10K
R3 10K
R4 10K
R5 1K
R6 4.7K
C2 2.2μF
S1
S2
S3
D3 IN916
D4
IN916
+5V
GND
NC
PSEN
RD
WR
ALE
MD
Interface Socket E
RD
WR
1N4001
D1
R1 330Ω
LED
C1 10μF
Power Input
Power to Circuits
CS
Interface Socket D
Output Device Codes
Input Device Codes

Figure 10.

Using Apple II Graphics from CP/M

Turbo Pascal Controls Apple Graphics

by Ted Carnevale

One of the most important features of the Apple II® family of microcomputers is their built-in graphics capability. Programs that run under DOS, PRODOS, or the p-System can be enhanced by using the built-in graphics commands of Applesoft BASIC or p-System Pascal.

However, graphics commands are generally lacking in high-level languages under CP/M® . This lack is ironic since Apples with Z80 cards account for a major portion of the microcomputers that run CP/M. Furthermore, many of the CP/M cards have fast Z80s and extra RAM that make them more powerful than other CP/M microcomputers. A few Apple versions of some languages (e.g.MBASIC and Microsoft's FORTRAN) do include some graphics commands, but these implementations are incomplete and work only with the older, slower Z80 boards.

This article tells how I wrote a handful of routines that allow me to use the Apple's graphics features. These routines work on an Apple or (pre-lawsuit) Franklin with a PCPI AppliCard or Franklin Z80 board. The number of changes needed to make them work with other hardware is restricted to a few sections that transfer data and commands between the Z80 and the 6502. The hardware-specific code is small, clearly identified, and easily replaced. Other modifications could be added to use the extended high-resolution mode of the IIe and IIc.

The Dilemma: Graphics or CP/M?

We use CP/M on an S-100 bus microcomputer at the lab, but I was satisfied with DOS and the p-System at home for a long time. When I started to use the Apple to work on programs for the lab, I ran into trouble.

Many of our data analysis programs are written in Turbo Pascal, which is just different enough from Apple's p-System Pascal to cause problems with program portability. Also, I had grown used to Turbo Pascal's compiler, which is much faster than the p-System compiler. Finally, moving back and forth between different operating systems and editors was a nuisance.

So I bought a Z80 card. The PCPI AppliCard seemed like a good choice. The Z80 on this board has its own 64K of RAM so it can run at 6mHz, 50 to 300% faster than most dedicated CP/M micros! With an optional 128K RAM expansion used as a disk emulator, the AppliCard turns the Apple into a machine that can run circles around most other 8-bit micros.

Before long, I was running WordStar and Turbo Pascal at home. It was great to be able to transfer files to the lab via modem, without having to worry about incompatibilities between DOS and CP/M!

Then it came time to work on graphics programs. We were developing routines for our bitmapped graphics display in the lab, and I wanted to try out some ideas at home. However, there wasn't any commercially available software that would let me use the Apple graphics routines from CP/M. Borland International was unaware of any graphics extensions to Turbo Pascal, and PCPI didn't know of any high-level language that ran under CP/M which would do what I wanted.

The Clues

The first step toward solving my problem was suggested by the *Apple Reference Manual*. This manual tells what addresses in the 6502's RAM correspond to the high- and low-resolution graphics pages, and where the *soft switches* are that activate graphics- or text- mode display. Machine language and Applesoft BASIC programs select among text, low-, and high-resolution display modes by activating one of these soft switches. Drawing on one of the graphics screens is accomplished by writing bytes to the proper addresses in one of the memory areas set aside for graphics display (graphics memory).

The second step was to find out how to activate the soft switches and write to the graphics memory from CP/M. Some Z80 boards share the RAM on the Apple's motherboard with the 6502. With these boards, the Z80 might be able to write directly into the graphics memory and toggle the soft switches without the cooperation of the 6502.

However, the AppliCard is actually a separate microcomputer with its own RAM. As far as its Z80 is concerned, the Apple is just a peripheral device that it communicates with through a parallel port. The details of this communication are described in PCPI's *AppliCard OEM Manual.* This manual gives examples of how the Z80 can tell the 6502 to write one or more bytes to any address in the motherboard's RAM.

Based on this information, I wrote some simple procedures in Turbo Pascal that passed data back and forth between the 6502 and the Z80, and combined them with other procedures that selected the display modes. Then I added a routine that was supposed to write a series of horizontal lines across the graphics screen of my choice. This was a crude, 'brute force' procedure that calculated the address of each byte in the Apple's display memory.

The resulting program could draw lines across low resolution page 1 or high resolution page 2. Trying to write over low resolution page 2 or high resolution page 1 made the computer hang, so that it had to be turned off and rebooted. Apparently these locations were needed by the drivers that PCPI used for the 6502 to communicate with its peripherals.

As limited as it was, this program was a success because it showed that what I wanted to do was possible. However, a lot more work lay ahead if I had to use this crude approach to draw lines and set colors.

A Refinement: Tapping the ROMS

The PCPI OEM manual also told how to make the 6502 execute a routine starting at a specified address. If I only knew where the routines to set colors and draw points and lines were located in ROM and what parameters they needed, I could save a lot of programming effort.

This information came from a book by Jeffrey Stanton entitled *Apple Graphics and Arcade Game Design* (The Book Co., LA, Calif. 1982). This is probably the single most complete reference on Apple graphics. The descriptions of the firmware graphics routines are actually only a very small part of the useful and fascinating material in this book.

The Apple's ROMs contain machine language routines to draw points and lines and to select the colors used for drawing on the low and high resolution screens. These routines are used by Applesoft, and probably by p-System Pascal as well.

In an Apple running CP/M with a Z80 card, they are still in ROM waiting to be called by the 6502. All that has to be done is to load the correct parameters into the 6502's registers and RAM, and then make the 6502 call the graphics routine with a JSR instruction. For example, the 'hlin' routine that draws a horizontal line on a low resolution screen at row v between columns h1 and h2 (where h1 ◄h2) starts at location $F819. It is called with the value of v in the A register, h1 in the Y register, and h2 in location $2C. The other graphics routines are accessed in a similar manner.

Calling the Graphics Routines From CP/M With Pascal

The listings give the Turbo Pascal routines that I use to draw on the graphics screens from CP/M. I have grouped them into several **include** files according to function. This helps keep program size and compilation time to a minimum.

However, it has the undesirable side effect of making the scope of some identifiers (names for constants, variables, procedures or functions) global. Since a global identifier is defined outside of all procedures or function definitions, it may be referenced by any procedure or function. As a result, naming conflicts and other side effects may occur. To avoid these, identifiers which probably should be 'hidden' from most application programs start with an underscore, which Turbo Pascal allows as the first character of an identifier.

The first of the include files is PCP.INC, which contains all of the elementary routines needed for the Z80 to communicate with the 6502. These access the special functions that the AppliCard uses to transfer data back and forth between the 6502's RAM on the motherboard and the Z80's RAM on the AppliCard. Further details about these functions are contained in PCPI's AppliCard OEM Manual. Since the hardware and software supplied by other Z80 board manufacturers are different, these routines would have to be changed before they are used with other Z80 boards.

Most of the routines in PCP.INC write or read one or more bytes to or from the 6502's address space, and are used to set the 6502 up for graphics operations. They can also be used to write or read data directly to or from the graphics screens. The last procedure in this file is "callapl," which makes the 6502 perform a JSR to a specified location.

Some constants and routines are common to both low and high resolution graphics. These include the addresses of the soft switches used to select among text, graphics, full screen or mixed graphics and text modes, and the routines that set these switches. These are gathered into APLGR/G.INC, which should be included after PCP.INC and before either APLGR/L or APLGR/H.

The registers of the 6502 must be loaded with the appropriate parameters before it executes a JSR to one of the ROM graphics routines. Since PCPI didn't provide any way to load the registers, I decided to create my own routines for this purpose.

These register-loading routines are written into an otherwise unused area in the 6502's RAM by special initialization procedures in two separate Turbo Pascal include files—lorespatch in APLGR/L.INC for low resolution graphics, and hirespatch in APLGR/H.INC for high resolution graphics. These include files also contain other procedures such as those needed to clear a screen, position the cursor, plot a point or line, choose the drawing color, or switch from text to graphics display. These other routines are self-explanatory, but lorespatch, hirespatch, and the 6502 register-loading routines require some explanation.

Before a register-loading routine is called, the AppliCard puts the drawing parameters in a common data area in the Apple's RAM. Next, the AppliCard

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

The MMSFORTH System. Compare.

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

MMSFORTH

A World of Difference!

The total software environment for IBM PC, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):

MMSFORTH System Disk (IBM PC)	$249.95
MMSFORTH System Disk (TRS-80 1, 3 or 4)	129.95

- Personal License (optional modules):

FORTHCOM communications module	$ 39.95
UTILITIES	39.95
GAMES	39.95
EXPERT-2 expert system	69.95
DATAHANDLER	59.95
DATAHANDLER-PLUS (PC only, 128K req.)	99.95
FORTHWRITE word processor	175.00

- Corporate Site License Extensions . . . . from $1,000
- Some recommended Forth books:

UNDERSTANDING FORTH (overview) . . .	$ 2.95
STARTING FORTH (programming)	18.95
THINKING FORTH (technique)	15.95
BEGINNING FORTH (re MMSFORTH) . . .	16.95

Shipping/handling & tax extra. No returns on software.

Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

tells the 6502 to JSR to one of the register-loading routines.

The LORES.A65 file is the source for the routines used to load the registers for low resolution graphics calls. A similar file generated the 6502 code needed to set up for high resolution calls. The code areas used to set up the 6502 for low and high resolution graphics calls are separate, so that parameters may be passed to both the low and high resolution ROM routines without having to rewrite the register-loading routines into the 6502's RAM. I wrote these register-loading routines in 6502 assembly language and assembled them with the 6502 cross-assembler that was part of the PCPI OEM package, but any 6502 assembler could have been used.

The instructions in each register-loading routine set up the 6502 for the graphics call, and then make it JSR to the proper graphics routine in the Apple's ROM. These short code segments for passing parameters to the Apple ROM's low and high resolution routines were designed to share a common data area—three bytes starting at $9000. Data written by the AppliCard to $9000 and $9001 are destined for the A and Y registers, respectively. Location $9002 is used to hold a byte that will be loaded into the X register or location $2C or $2D, depending on which graphics routine is being called.

I incorporated the hex opcodes and addresses from these assembly listings into the procedures lorespatch and hirespatch using a nonstandard but convenient feature of Turbo Pascal called a typed constant (the arrays LORESTUFF and HIRESTUFF) that allows variables to be initialized. Users of other Pascals may have to use a series of standard constants and assignment statements to enter the patches.

Sample Programs

The program LOWRES.PAS demonstrates how the low resolution routines are called from a Pascal program. The files PCP.INC, APLGR/G.INC, and APLGR/L.INC are included into the source to make the low resolution routines available. The main program block begins by calling lorespatch to install the register-loading routines into the Apple's RAM. After printing a signon message, the main program selects and clears low resolution graphics page 1, and then returns to the TEXT mode to display the next message. Next it runs through the low-resolution colors as it draws a series of horizontal and then vertical lines across the screen.

Low resolution graphics may be useful for games and pattern generation, but high resolution is better suited for plotting data. The program SINES.PAS draws a cycle of a sine wave repeatedly in different colors on the high resolution screen. It shows that the high resolution procedures are installed and called in a manner similar to that used for low resolution graphics.

It also illustrates a crude but effective general approach to plotting real world data (data in world coordinates) on a display screen, which is patterned after the more extensive SIGGRAPH and GKS approaches. The basic idea is that measurements along the real world X and Y axes are scaled and shifted so that points in a rectangular region on the X,Y plane will be mapped into a corresponding rectangular area on the display.

The lower left and upper right corners of the rectangular area in world coordinates are described by variables of the data type

```
realdata = RECORD
   x:real; {horizontal "real world"
      coordinate}
   y:real; {vertical "real world"
      coordinate}
END;
```

The variables "lowerleft" and "upperright," which are diagonally opposite corners of this rectangular area, specify its location and size.

The area on the display that is to correspond to points in this rectangle is described by variables of the type

```
screendata = RECORD
   x:integer; {horizontal coordinate of a
      point on the display}
   y:integer; {vertical coordinate of a
      point on the display}
END;
```

This implementation emulates only a tiny fraction of the SIGGRAPH and GKS standards. Therefore I decided to call the area of the screen to which the real world data are mapped a "frame" rather than a "window." The variable "frameloc" is the position of the left upper corner of the frame, and "framesize" is the width and height of the frame.

The variables "lowerleft," "upperright," "frameloc" and "framesize" are used to set up the X and Y scale factors and offsets that will be needed later to map the world coordinates onto the display. Procedure "setframe" in PLOTTER.INC calculates these scale factors, which are returned in the variable "frame." Because these scale factors and offsets are calculated by a procedure that is different from the procedures that use them, the variable "frame" is declared in the main program heading so that it is global.

PLOTTER.INC also contains two other procedures. The cursor is moved to a point on the display at a location that corresponds to its world coordinates, and a point is drawn there, by the procedure "point." To draw a line from the last location of the cursor to a specified point, the procedure "plotline" is used.

The high resolution screen is easily dumped to a dot matrix printer with a program like DUMPSCRN.PAS. This program was written for use with a Grappler parallel interface card and an Epson MX-80 printer, but it could be modified to run with other interface cards and other printers. Alternatively, data could be read from the high resolution screen one line at a time and used to drive some other output device if necessary. If this were done, the high order bit of every byte should be omitted, since this bit selects color and does not represent a point on the screen.

The programs SAVSCRN.PAS and GETSCRN.PAS perform a simple screen save and restore. SAVSCRN reads the high resolution screen one line at a time and writes it to disk in a file that occupies 8K bytes. GETSCRN reads such a file and writes its contents into the high resolution display, obliterating anything that was there before. These two programs could be modified for data compression, reducing the disk space required. GETSCRN could also be changed to allow ORing the file contents with the data already on the screen, so that the contents of several files could be superimposed on one another.

Conclusions

These routines will run as-is with the PCPI AppliCard on an Apple II+/IIe or Franklin (with old-style, Apple-copy ROMs). They should require little modification to run with other high level languages.

Users with different Z80 boards will probably need to change the routines that communicate between the Z80 and the 6502. In different CP/M implementations, portions of the graphics memory may be allocated to vital systems routines so that writing anything to them may cause a crash. Some experimentation may be needed to find which of the low- and high-resolution screens can be safely used.

In writing these routines, I deliberately chose a syntax that was similar to Applesoft BASIC's graphics commands. The command names can be changed to suit any preference.

Although these routines provide only a few commands, I have found them to be quite useful. I have incorporated them in programs that test random number generators, generate fractal images, and solve differential equations that simulate nerve signals. These routines could easily be expanded to include additional commands, like relative cursor move and draw, and curve- and circle-drawing algorithms. In particular, PLOTTER.INC could be replaced by a set of procedures and functions that implement SIGGRAPH or GKS routines, such as clipping algorithms. ■

INDEX OF LISTINGS

File	Purpose	Listing
PCP.INC	Data transfer between Z80 and 6502	Listing 1
APLGR/G.INC	Constants, types and routines shared by low- and high-resolution graphics	Listing 2
APLGR/L.INC	Register-loading routines for low-resolution graphics	Listing 3
APLGR/H.INC	Register-loading routines for high-resolution graphics	Listing 4
LORES.A65	6502 assembly language source for low-resolution register-loading routines	Listing 5
LOWRES.PAS	Demonstration of low-resolution graphics functions	Listing 6
SINES.PAS	Demonstration of high-resolution graphics functions	Listing 7
PLOTTER.INC	Routines for mapping "world coordinates" onto the high-res display--used by SINES and other high-resolution programs	Listing 8
DUMPSCRN.PAS	Dumps high-resolution screen to dot-matrix printer	Listing 9
SAVSCRN.PAS	Saves contents of a high-resolution screen in a file on disk	Listing 10
GETSCRN.PAS	Fills a high-resolution screen with the contents of a file produced by SAVSCRN	Listing 11

```
LISTING 1.  PCP.INC
===================

{PCP.INC contains primitive routines to communicate between
 the PCPI Z80 card (Applicard) and the Franklin or Apple.
 Copyright 1984 by N.T.Carnevale.
 Permission granted for nonprofit use.}

{Include before any APLGR file.
 Variables and constants which are to be hidden from the
 user's programs start with an underscore} .

CONST
   {ports}
   _HOSTOUT=0;
   _HOSTIN=$20;
   _HOSTSTAT=$40;
   {commands to transfer more than one byte}
   _RDHOST=1;  {Z80 -> 6502}
   _WRHOST=2;  {6502 -> Z80}
   {commands for single byte transfers--vers.9 of PCPI ROM}
   _RDBYTE=6;
   _WRBYTE=7;
   {command for 6502 to execute a procedure}
   _CALL=3;

{*************************************************************
These three routines are low level "primitives" that should
probably never be called from procedures outside this file}

FUNCTION _recvbyte:byte; {get a byte from the 6502}
CONST READY=$80;
BEGIN
  WHILE (READY AND port[_HOSTSTAT]) = 0 DO ;  {wait til ready}
  _recvbyte:=port[_HOSTIN];  {get byte}
END;

PROCEDURE _sendbyte(datum:byte);  {send a byte to 6502}
CONST BUSY=1;
BEGIN
  WHILE (BUSY AND port[_HOSTSTAT]) <> 0 DO ;  {wait til ready}
  port[_HOSTOUT]:=datum;  {send byte}
END;

PROCEDURE _sendword(data:integer);
{send a word (low byte first) to the 6502}
VAR a:RECORD CASE boolean OF
        TRUE:  (i:integer);
        FALSE: (b:array [1..2] of byte);
      END;
BEGIN
  a.i:=data;
  _sendbyte(a.b[1]);  _sendbyte(a.b[2]);
END;

{*************************************************************
Now the blocks that may be referenced by other ones}

FUNCTION _rdhostbyte(apladdr:integer):byte;
{get a byte from the 6502's RAM at address apladdr}
BEGIN
   _sendbyte(_RDBYTE);   _sendword(apladdr);
   _rdhostbyte:=_recvbyte; {get data}
END;

PROCEDURE _wrhostbyte(apladdr:integer; datum:byte);
{send a byte to address apladdr in the 6502's RAM}
BEGIN
   _sendbyte(_WRBYTE);   _sendword(apladdr);
   _sendbyte(datum);
END;

PROCEDURE _rdhostdata(sourceaddr,destaddr,bufsize:integer);
{transfers bufsize bytes from the 6502's RAM to the Z80's RAM.
 Arguments are the starting addresses of the source and
 destination, and length of the buffer area which is to
 receive the data.  Call thusly:
   _rdhostdata(apladdr,ADDR(buffer),SIZEOF(buffer));
}
VAR
  i:integer;
  b: byte;

BEGIN
 _sendbyte(_RDHOST);
 _sendword(sourceaddr);
 _sendword(bufsize);
 b:=PTR(destaddr);
 FOR i:=bufsize DOWNTO 1 DO BEGIN
   b^:=_recvbyte;
   b:=PTR(ORD(b)+1);
 END;
END;

PROCEDURE _wrhostdata(sourceaddr,destaddr,bufsize:integer);
{transfers bufsize bytes from the Z80's RAM to the 6502's RAM.
 Arguments are the starting addresses of the source and
 destination, and length of the buffer area which is to
 receive the data.  Call thusly:
   _wrhostdata(ADDR(buffer),apladdr,SIZEOF(buffer));
}
VAR
  i:integer;
  b:^byte;
BEGIN
  _sendbyte(_WRHOST);
  _sendword(destaddr);
  _sendword(bufsize);
  b:=PTR(sourceaddr);
  FOR i:=bufsize DOWNTO 1 DO BEGIN
    _sendbyte(b^);
    b:=PTR(ORD(b)+1);
  END;
END;

PROCEDURE _callapl(apladdr:integer);
{executes routine in the 6502's RAM starting at apladdr.
 This routine must end with a "return" command.
 NOTE:  for locations > 32K, either use negative integers
 or hex constants}
BEGIN
  _sendbyte(_CALL);
  _sendword(apladdr);
END;

{end of PCP.INC}
```

```
LISTING 2.  APLGR/G.INC
=======================


{APLGR/G.INC enables calling graphics routines in Apple's
 ROMs from Turbo Pascal programs.
 Requires the PCPI Z80 card (Applicard).
 Copyright 1984 by N.T.Carnevale.
 Permission granted for nonprofit use.}

{Include after PCP and before APLGR/L or APLGR/H}

{contains these routines used by both hi- & lores graphics:
PROCEDURE _setpartition(part:partition);
PROCEDURE _selectpage(pagenum:integer);
PROCEDURE textscreen(pagenum:integer);
}

CONST
  _BPL=40;  {# bytes/line of hi or lo res display}

  {software switches for control of graphics features}
  _GRFX=$C050;          _TXT=$C051;
  _FULSCRN=$C052;       _MXD=$C053;
  _PG1=$C054;           _PG2=$C055;
  _LRS=$C056;           _HRS=$C057;

  {temporary storage for parameters}
  _AREG=$9000;  _YREG=$9001;  _LOCXX=$9002;  _XREG=$9002;

TYPE
  partition=(FULLSCREEN,MIXED);
  _screenmode=(TEXT,GRAPHICS);
```

(continued on page 29)

Soldering and Other Strange Tales

First in a Series on Soldering, Unsoldering, and PC Board Repair

by James O'Connor

Edison discovers electron flow in a vacuum. Lee DeForest perfects the vacuum tube. Bardeen, Brattain and Shockley of Bell Labs invent the transistor. Jack Kilby of Texas Instruments builds the first integrated circuit. Intel produces the microprocessor. Who first used solder for electronics? Does anyone know the name of this genius? All those great advances of electronics, literally held together by a common thread of solder. Would modern electronics even be possible without soldering?

Recently, I reorganized my scattered collection of magazines and books on electronics, and it took me almost forever since I kept reading them. A recurring theme was 'How to Solder' but as I read one article after another two trends became obvious. Either articles were very brief, basic even and omitted things I now know can be perplexing to anyone who solders; or they were very technical, discussing things like the eutectic point (I won't even elaborate on this) providing interesting, but not very practical data for actual soldering.

Thus a more comprehensive article devoted to soldering for the occasional kit builder is needed. It should cover all sorts of techniques and answer questions like "Whats that brown scrunge around all the solder joints?" Strikingly the most missing information was how to do things like unsoldering, or how to fix a damaged circuit trace. Most articles seem to assume that everybody just starts soldering with perfect precision and never solders a part in the wrong place. I certainly have done that. And sometimes parts even fail, yet more reason for unsoldering them.

So this series really developed inside out since it didn't seem fair to write about these unheard of topics without also describing the basic soldering process. Just note that we're limiting the scope to the problems and methods of soldering for electronics kit builders and those who occasionally need to repair a unit. Soldering small parts to printed circuit boards (PC boards), wires to lugs (metal loops on switches or terminal strips) etc. Certainly very familiar to many but perhaps not to all readers, maybe this series will provide the impetus for a software fanatic to turn into a hardware hacker, stranger things have happened before.

The Theory

This series is meant to be a little offbeat so let's start by talking about welding because that is what soldering is NOT. Due to some minor similarities some people think the two are related, but they are really quite different and it will help to realize this. In welding two pieces of identical or compatible metal are brought together and heated very hot, until they become molten or liquid, and in this state they mix together. After cooling, the result is a single piece formed by welding. A close analogy would be to take two ice cubes, let them melt a little, press them together and pop 'em back in the freezer, take them out later and there is one piece of ice.

Journey now to the realm of the microscopic surface of a printed circuit board. At the level of an electron microscope we would see that the circuit traces are not smooth shiny surfaces, rather they are pitted and cratered like a giant moonscape. Most surfaces that appear smooth at our human level of experience are very rough at the level of individual molecules. Think what would happen if a liquid were spilled onto such a surface, it would flow into and through the various pores and crevices. Most common glues are liquids, and when put onto a surface they seep into it, and then the solvents in the glue evaporate and it hardens. Once hard it is stuck into the surface and if another surface is nearby the two are stuck together. This is precisely how wood and paper glues work. Long ago people discovered that some metals would act like glue on other metals if they could be made to flow like a liquid. How did they make these metals flow? They heated them!!

The trick was to find a metal or an alloy of two metals that would turn liquid at a low temperature. Low enough to make the process practical with simple tools, and avoid the danger of melting the two metals to be joined. This is just the opposite of what you want when welding. There was another problem to be overcome, there are contaminants in the form of corrosion and tiny particles of dirt on the surface of any metal. To allow the melted glue-metal to flow into the surface these must be removed. They can be scraped and polished off, but to get them out of the crevices below the top surface some chemical must be used. This chemical acts to clean away the interfering corrosion.

Thus the basic scenario of soldering—metal-glue heated till it flows as a liquid over a well cleaned surface of another metal, remove the heat, the glue-metal hardens, and the joint is completed.

Let's follow the steps to make a solder connection, such as attaching the lead from a resistor to a printed circuit board. The lead is pushed thru the hole in the board and is surrounded by the printed trace. Both appear shiny so there is no need to scrape or clean them. Apply the soldering iron for about three seconds, then feed in solder on the opposite side of the iron. If both the lead and the trace have been heated correctly, they will be hot enough to turn the solder to liquid. It will flow around them both, like any liquid it will 'wet' them. Remove the iron and keep everything motionless for a couple of seconds and the solder hardens.

Sounds simple doesn't it. Then why do so many have so much trouble with so simple a task? There are probably many reasons, but if I had to chose just one I would chose, heat! The whole key to soldering is the right amount of heat in the right place at the right time. Without heat, soldering just won't work; with too much heat, damage occurs. Applying heat properly is really easy with the right tools, so lets talk about the tools next.

Soldering Tools

There are only three really important rules about soldering tools:

1. Use the right tool.
2. Use the right tool.
3. Use the right tool.

To put it more succinctly, the easiest way to get into trouble is with the wrong tool. But what is the wrong tool, what's the right tool? To answer that question we're going to list some brands and specifications but more importantly we'll first talk about the function of the soldering tool, an understanding of which should allow us to always select the right tool for any task.

Some soldering iron parts have descriptive names like screwdriver tip, chisel tip, conical tip etc. and you might be excused for thinking the job of the soldering iron is similiar to a screwdriver or a chisel. Not at all, the purpose of the soldering iron is to deliver heat, precisely the right amount in the right place. The names only describe the shape and not the real function of these parts.

A good solder joint occurs when the two parts being soldered are hot enough to melt the solder, to make it liquid. Simple enough? In theory it is but in real practice there are some problems and the right soldering iron solves those problems, the wrong one causes them.

Consider again the simple task of soldering a resistor onto a printed circuit board. All we have to do is to heat both thee resistor lead and circuit trace to about 750 degress Fahrenheit right where they abut each other, apply some solder and let it cool. First problem; both the resistor lead and trace being made of metal are good conductors of heat. When the soldering iron is applied to the junction both will heat up and at the same time the heat will start flowing away thru both.

If the soldering iron is too small then the amount of heat flowing away will prevent the joint from getting hot enough. Even worse the joint may eventually heat up enough but by then the resistor body has absorbed a great deal of heat and due to that may change value, the circuit trace may also have absorbed too much heat and delaminated (lifted off the board). So, we may make the solder connection while destroying the resistor and damaging the trace. And do you know what that means? We will have to unsolder the whole thing, repair the board, get a new resistor and try again. The follow-on parts of this series will discuss those topics, unsoldering and circuit board repair, because soldering is part knowledge and part art so even the best soldering artist may need to make such repairs.

Many newcomers to soldering are rightly fearful of the damaging effects of heat so they tend to chose the smallest soldering iron available, unfortunately as we've seen this can cause damage. The real 'trick' is to overwhelm the the joint with enough heat that it quickly reaches the required temperature, make the connection, remove the iron and let the heat dissipate.

It would seem that you should use the biggest soldering iron available, and occasionally I have observed some very skilled, experienced technicians use an oversized tool to make a connection but they only did so lacking a more appropriate tool. Circuit boards can be

EVERYTHING YOU NEED . . . $279.00

Now it's easy to program the Heath-Zenith HERO-1® Robot with an Apple® II. HERO® Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics. The HERO® Macros come with 30 pages of documentation.

[For example, the line: 1130 > MVWRIM GRIP. OPEN. 60. FAST instructs the HERO® to open his gripper 60 units at fast speed. Motor position is expressed in base-10.]

Transfer to HERO® with ROBI . . . an affordable interface for the robotics experimenter . . . is simple.

- **ROBI is a complete package. No additional hardware required for Apple® or HERO®.**
- **ROBI installs quickly in an Apple® II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO®.**
- **With ROBI and the Cross Assembler, the programmer uses Apple®'s memory to write the program, and HERO®'s memory to run the program.**
- **Not "copy protected," archival copies may be made as needed.**
- **ROBI offers expansion potential.**

VISA and MasterCard accepted.

The Cross Assembler with HERO® Macros sells for $100.00; the ROBI Interface sells for $199.00. Both as a package — $279.00.

To order, or for more information, call (303) 670-6137.

BERSEARCH
Information Services

26160 Edelweiss Circle
Evergreen, Colorado 80439

APPLE® is a trademark of Apple Computer. HERO® is a trademark of Heath E[illegible] crowded places, with very fine closely spaced traces, an oversize tool can cause a problem called 'bridging' which is accidently joining two parts when they aren't supposed to be. Plus ultra precise timing is needed since too large a tool can supply heat too fast.

Look at almost any selection of soldering irons, in a store or catalog, and you'll notice the great variety. Indeed, some large catalogs are devoted entirely to soldering irons and accessories, and many have several pages of tools. We won't delve into all this since these are all instances of soldering tools for a specific job in industry such as soldering very small items, or very large etc. For simple kit building, all you really need is a basic soldering iron preferably with interchangeable tips.

Soldering irons are rated in watts, the more watts the more heat the iron can supply. They all reach a temperature between 700 to 800 degrees Fahrenheit. Thus the wattage is not related to temperature, but really to the physical size of the parts to be soldered. For very small parts there are 12 watt irons, for most circuit board work there are 25 and 30 watt irons, for circuit boards with large wide traces and for large switch lugs and terminal strips there are 40 and 60 watt irons.

Let's digress for just a moment. With basic soldering irons you control the actual temperature of the joint by how long you apply the iron to it, this is a matter of acquired experience. It's easy to learn and the solder itself acts as a temperature gauge. For industrial work there are soldering stations that self-regulate their temperature. These are more elaborate and expensive, they're fine if you'll be doing frequent soldering and are very convenient but hardly necessary for kit building.

For most circuit board and wire to lug soldering, a 25 to 30 watt soldering iron is just right. If you don't have such an iron then get one. It will pay for itself even if you use it to build just one kit, and the entire package should cost less than $25 dollars. There are literally dozens of suitable irons availble so if we don't mention some chalk it up to lack of space. The most inexpensive type of iron would be the Radio Shack #64-2067 which is a 30 watt pencil type iron, list price is only $3.49. This a good tool except that the tip is built in, that is you can't change it and once it wears out you replace the whole iron. Some tips cost more than this iron, for one kit this tool will do an adequate job.

Radio Shack also sells two packages that include a 30 watt soldering iron, #64-2802 includes soldering accessories only and sells for $4.95 while #64-2801 which sells for $14.95 also includes a diagonal cutter, needle-nose pliers, and three screwdrivers. If you don't have any pliers the 2801 package is a good choice, while the smaller 2802 package is not my first choice. I think you would do better to purchase those items individually, you'll be able to chose better items and buy them in bulk.

Next up the scale would be Radio Shack #64-2070, a 25 watt model with a replaceable tip, but there is only one style and size of tip, list price $5.49, tips are $1.29. There is also a gun handle model #64-2065, list price $3.99, tips $.89. Most people find the straight iron type easier to control but a few prefer the gun style, the choice is up to you. Moving up, Radio Shack also sells model #64-2055 which features a switch to select either 15 or 30 watts, list price $6.95, tips are $.89 but again only one style is available.

Top of the line is the modular unit, This consists of a handle #64-2080, list price $4.99, to which you add a heating element in this case the 27 watt unit #64-2081, price $4.79, and a set of tips such as #64-2084, price $3.69. These are three different style of tips, broad, medium, and fine point. Grand total cost, $13.47.

We've skipped the Radio Shack 42 Watt iron as that is too much wattage for most kit building. The 75-Watt Soldering Gun #64-2191, is also too powerful. It's a different device because it heats its tip by sending low voltage current through it. Resistance to the current flow causes it to heat up just like the element in a light bulb. These gun type tools are very good for heavy soldering tasks especially since they heat up almost instantly and you might want to have one if you plan to really get involved in soldering. The torch type tools in the Radio Shack catalog are not suited to any electronics assembly that I know of.

What about other manufacturers? There are at least two that deserve mention, but you'll have to scout up sources for their products. The tool I use is a Weller WP25 soldering iron (25-Watts, of course) which costs about $17.00, I paid less but then I have had it for over ten years now. This unit can be used with eight different style tips, my favorites are the so-called screwdriver tips, these are really more oval shaped with smoothly rounded edges. Not good for driving screws at all, but great for soldering as the rounded shape easily butts up against the lead and circuit trace while still allowing a channel for the solder to flow around the joint. The three I employ are the ST1 a 1/16" tip, the ST2 3/32" and the ST3 1/8", these cost about $3.00 each. The ST2 is used for almost 90% of my work, switching to the ST1 for very small work or the ST3 for larger items.

Heathkit sells a package of tools (Model GGP-1270) for kit builders that includes the Weller WP25, it sells for $39.95 and again if you don't have any tools this is a quality assortment to begin with. They also sell the WP25 seperately as Model GDP-207A, price $14.95. But they don't offer the interchangeable tips thru their catalog.

The other big name is Ungar. They have a very extensive line of soldering irons and tips, in fact so extensive that I think the hardest task is selecting the right tool from the variety. Ungar offers a Hobbyist iron Model CM-25 for about $8.00 that is a simple inexpensive unit. Then they have the Modular Standard Line wich includes a handle such as the 777 or 776 both about $7.00. These handles accept heaters in either the Ungar S or HP series. One type such as the 533-S, 536-S, or 539-S includes both heater and tip, for about $9.00 each. While the 537-S, cost about $8.00, accepts thread-on tips such as the PL-151 screwdriver tip which cost about $3.00. And there are even more available, including some special accessories. At the end of this article I list some sources for catalogs that include both the Weller and Ungar line, you may wish to obtain these catalogs just to see the great variety of soldering tools available.

The Radio Shack Modular line and the Ungar Modular are so similiar that it is fair to conjecture that Ungar makes the Radio Shack units but this doesn't guarantee compatability between the various accessories.

We'll need one more item to complete the package and that is the most important of all, a suitable stand for the iron. Why is it so vital? Because it provides safety. Hot soldering irons can burn people, char things, melt them, set some things afire, and just plain wreck havoc. The soldering iron holder over-comes most of these dangers. You'll still need to exercise care when using a hot iron but it's much safer with a holder. Some kits come with a holder that consists of a bent piece of wire on a small plastic disc, these are still better than nothing at all but just barely. Do purchase a real holder, Radio Shack #64-2079 is a satisfactory unit, list price $5.79. The Weller PH25 cost about $8.00 or PH60 cost about $10.00 are also good. Don't fail to buy a holder because of the cost, it's worth it, you'll save the price if it prevents just one accident.

There are only two things to keep in mind when selecting a holder, first be sure that your iron fits the holder, some units are made for specific irons and won't hold other brands or types. For instance, Ungar makes one holder for their Standard irons and another for the Princess irons. Secondly, some holders are made to be free standing while others work best only if screwed to the workbench, just be sure to get the type that fits your situation.

The holder will include a sponge, use this to keep the iron properly 'tinned'. When you first heat the iron, stroke it across the dampened sponge on each edge and then melt just enough solder onto the tip to coat it. As you work, heat will cause a layer of 'crud' to build up on the tip, this stuff is an insulator that can block heat transfer. Clean it off by stroking the tip over the sponge, if the tip comes away with a bluish color, re-tin it with solder to keep it working at maximum efficiency.

One final precautionary hint—none of the irons we've mentioned glow when hot, nor do they have any indicating lights to show that they are on. This can be a real hazard since you may go off and leave the iron on, if you have a stand this won't hurt anything but your electric bill (no worse than leaving a 25-Watt bulb on). Without a stand this can cause a fire, yet more reason for having a stand. A good idea is to use one of those barrier type outlets with a built in power-on light, with this you can always be sure that everything is off when you finish.

To summarize, all you'll need is a 25 or 30-Watt soldering iron, ideally with changeable tips, and a good holder for it and you're all set for 99% of all the soldering required for most kits.

Using The Tool

If you do get one with changeable tips, then how to do you select the right tip for a given task? It's really quite simple but it does involve a little judgement. Take a look at the circuit board you'll be working on, estimate the average size of the circuit pads where the joints will be, and chose a tip that is approximately 60 to 80% as wide as an average sized pad. This is all very approximate, precise measurments are not required. Try to avoid using a tip that is wider than most pads as that can induce bridging.

Of course some pads may be much smaller while others are much larger, this is where you need to switch tips especially for the larger ones. Pads that connect to the ground plane of a circuit board are often very large, indeed they may blend into wide expanses of trace. These 'big game' pads often gather in the vicinty of the electronic equivalent of a waterhole, namely voltage regulators. The sheer size of them draws away heat too quickly, thus switching to a larger tip can make the difference when soldering these big pads.

This leads to a pet peeve of mine, most kit instructions blithely ignore the problems that occur when soldering different size items. That is the instructions will often have you solder things following some pattern totally unrelated to the requirements of soldering, such as working in only one area. This is one instance where it may be wise to skip some steps because you can see that you have the wrong size tip on your iron. Just be sure to mark the steps you've skiped so that later you can go back and re-do them.

Instructions that have you install things by size, that is starting with the smaller items and ending with the larger ones are usually just fine for soldering, start with a small tip if needed, switch to a medium size, ending with the large size. Do be methodical, as this type of instruction predisposes you to installing parts in the wrong location or forgeting a part. Perhaps in a future article we can discuss the fine art of interpreting instructions.

Flux

What an odd sounding word, look it up in the dictionary and you find that it means: flow, state of change, a discharge of liquid, and an aid to soldering. The other meanings are rarely used anymore, so clearly we are interested in its association with soldering.

Let's start at the beginning. The first

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- •79 STANDARD
- •DIRECT I/O ACCESS
- •FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- •ENVIRONMENT SAVE & LOAD
- •MULTI-SEGMENTED FOR LARGE APPLICATIONS
- •EXTENDED ADDRESSING
- •MEMORY ALLOCATION CONFIGURABLE ON-LINE
- •AUTO LOAD SCREEN BOOT
- •LINE & SCREEN EDITORS
- •DECOMPILER AND DEBUGGING AIDS
- •8088 ASSEMBLER
- •GRAPHICS & SOUND
- •NGS ENHANCEMENTS
- •DETAILED MANUAL
- •INEXPENSIVE UPGRADES
- •NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

people to solder must have used something as a flux but didn't call it that, we don't know if they called it anything. Eventually some onlooker must have asked "What do you call that stuff?" and the ancient solderer may have mumbled, "I call it the stuff that helps me to solder, it changes the metals so they bond." Immediately, they went to their Funk & Wagnalls or whatever they had then and looked for a word that meant change, or helps to change, preferably a catchy little four letter word, and lo they found 'flux'.

So flux helps change the metals, but what does it change? One property of metals is that they tarnish and corrode. Some more rapidly and obviously than others but they all do it. Corrosion is the tendancy of metals to combine with oxygen which forms a metallic oxide, steel is the prime culprit—there is even a word to describe the process, rust. Steel corrodes quite slowly and the resulting oxide, rust is crumbly and unstable. Most other metals corrode so rapidly that the resulting oxide forms a skin or coating that seals off the surface of the metal preventing further corrosion. Zinc does this which is why steel is often coated with zinc, in order to protect the steel, in the process called galvanizing. Copper, nickel and other metals used on leads and traces also tarnish and corrode and that can block the molten solder from bonding to them.

One way to clean the metal would be to sand it, plumbers do this when soldering copper pipe. The sanding breaks the surface coating but more tarnish extends down into the metal and this must be removed by some chemical action that can flow into the minute pores of the surface. The plumber uses a mild acid mixture to accomplish this.

Electronics technicians could also use acid, and in the early days actually did, but the acid remains after soldering and can continue to eat away at the metal. The pipes that a plumber solders are massive enough that only a minute amount of metal is destroyed before the acid is used up, but electronic joints are tiny in comparison. Early users of acid fluxes for electronic work soon discovered that the connections would fail after a few months, literally eaten away by the residue of acid flux. This is why there are so many dire warnings about using acid fluxes for building kits or any other electronic work. It is also why you shouldn't buy flux at a hardware store. They carry flux meant for plumbing work and the salespeople often don't know that such fluxes should never be used for electronics.

So what can be used for electronics, clearly early users had to find something other than acid. They found the right product in the form of rosin. Rosin (sometimes also spelled resin) is a material that occurs in pine trees and is obtained along with the distillation of turpentine. It is virtually inert, that is harmless to most substances when solid, but when heated it becomes a liquid solvent. In its liquid state it does a fairly good job of dissolving the tarnish layer on metals used in electronics. Rosin is easy to work with, early users ground it into a powder and applied it prior to soldering, combining it with some liquid solvent to form a paste. This worked fine but then someone had the idea of combining solder and rosin, they developed a way of extruding the solder as a hollow wire with the rosin inside, what is today called 'rosin-core solder'. Purchase solder at an electronics store and you can't help but obtain flux in this, its most convenient form.

If this were any old article about soldering and fluxes this would be the end of the story but there is more, even a minor mystery of sorts, that is still to come.

As you use rosin-core solder it melts and part of the rosin also melts flowing out to clean away the tarnish. Some of the rosin also goes up in a little puff of smoke. Nowhere in any tutorial have I ever seen the smoke mentioned, needless to say many a first time solderer has been puzzled and even alarmed by this phenomena, after all smoke is associated with fire and burning. The smoke is supposedly harmless, at least I don't know of any studies indicating that is an occupational hazard. I do know that many people, myself included, find it an irritant to the eyes and nose. Invariably just when you're straining to see what you're doing, the smoke makes you want to sneeze. This effect seems to wear off quickly and everyone is affected differently, so if soldering makes you sneeze perhaps now you know that it's really the 'flux'.

What about the rosin that doesn't go up in smoke? It usually forms a coating around the completed solder joint. When still molten it will be clear in color and as it quickly hardens (one or two seconds) turns a brownish shade. Here comes yet another pet peeve of mine. In any tutorial on soldering there always appears the picture of the ideal solder joint, AFTER the rosin has been cleaned away. The trouble is that with rosin core solder, you always get excess flux on the joint which will partly obscure it. The experienced solderer learns to peer through this coating to check the joint, but the newcomer is often confounded by the appearance of what is probably a perfectly good connection. Later we'll talk about cleaning away flux and you can do this to verify that a solder joint is good or bad until such time as you can tell from experience.

There is yet one more phenomena that occurs. When the flux melts it also tends to bubble like any hot liquid, these bubbles will literally pop out of the joint and land anywhere from a quarter inch to a half inch away where they then harden. This same thing happens in welding, except that the bubbling liquid is molten metal as are the solidifed bubbles. Again many novice solderers may think the rosin bubbles are soldified solder, they can look very metallic and do pop out of the solder joint. Just poke these bubbles and you'll find that they crumble into dust.

By now experienced readers may think we've exhasuted everything there is to say about flux, but wait remember we promised a mystery, here it is.

Rosin is considered inert, that is it doesn't corrode anything and won't conduct electricity, and for a long time everyone seemed to agree on this. Instructions with any electronic kit never mentioned any potential problems with the rosin left on the circuit boards—indeed they rarely mentioned it all. But then a few years ago articles began to appear that instructed us to clean away the flux after the soldering was completely finished. Indeed some of these warned of dire results if the flux was not fully purged from the board.

As a firm believer in following instructions, I like many others dutifully complied. And then I discovered something never mentioned in the instructions (so what's new), it's really hard to clean away rosin flux. You need to use strong vaporous solvent to disolve the stuff and flush it away. Some of these solvents can damage the parts mounted on the board. Parts that have plastic are very susceptible, relay housings, connectors etc., and since you have to invert the board to clean it you can't prevent the solvent from reaching such parts. Rosin is also very sticky when it is dissolved, recall that it is also called resin. You have to flush the board in solvent several times to remove most of the flux, and even then there is a thin sticky residue left behind.

The mystery arises because not all kit makers recommended this step, examining circuit boards in commercial products also showed some that had been cleaned others not. Who's right and why had supposedly harmless inert flux suddenly turned into an insidious destroyer of circuits? Gradually, I pieced together a possible explanation. I have no way of knowing if it is factual but I will present the hypothesis and you can be the judge.

Soldering by hand is a labor intensive step, American electronics makers in an effort to compete with the low cost labor of the Far East have long sought to eliminate it. Two ways to do this are dip soldering or wave soldering. In these two techniques all the parts are simply mounted (stuffed is the technical term, and yes it is inelegant) on a board which is then conveyed to a large heated tank containing molten solder. Dip soldering is just what the name implies, the lower surface of the board is dipped into the solder, in wave soldering the board is suspended just above the surface of the tank while a wave is formed in the solder which sweeps past the bottom of the board. Both methods allow all the connections to be made in one step.

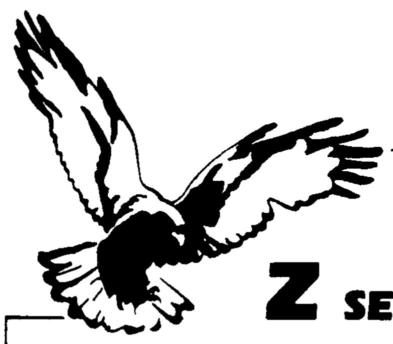
In either of these techniques flux must be used, but obviously rosin core is impractical. In fact rosin isn't quite up to the job, since it has to melt it couldn't work fast enough to thoroughly clean. So in wave or dip soldering the flux is applied just before the soldering station (the whole process is usually automated) and is an mild acid flux.

To recap, the boards are stuffed with parts, they move to a station where an acidic flux is applied, then to a wave soldering tank. Guess what has to come next? If you guessed a station where any remaining flux is cleaned off you're right. This flux has to be cleaned off because of it's corrosivness. And the final station on the line, why a visual inspection station, yet another good reason for cleaning the flux away.

My feeling is that some people familiar with how boards are soldered and then cleaned in an automated production facility incorrectly extended that to what happens when boards are hand soldered using good ole harmless rosin-core solder. You can verify this hypothesis by the fact that most circuit boards from America are indeed well cleaned while boards in products made in the Far East rarely are (although this is changing as automation is spreading throughout the world.)

The bottom line is that if you use rosin-core solder then you really don't have to clean it all away. You can do so if you wish for aesthetic reasons or to inspect the connections.

One more comment about flux removal. If you purchase finished products under warranty you will probably be warned that the warranty is void if repairs or alterations are made. Let's assume that you wish to alter or repair the item and that you can open it up without breaking any seals or leaving other telltale clues as to what you've done. If you do any soldering then you should either clean away any flux if the board is otherwise fluxless or be sure not to if that is its original condition. Otherwise should



Z SETS YOU FREE!

Z Operating System, an 8-bit OS that flies! Optimized HD64180/Z80 assembly language code — full software development system with proven linkable libraries of productive subroutines — relocating (ROM and RAM) macro assembler, linker, librarian, cross-reference table generator, debuggers, translators and disassemblers — ready to free you!

High performance and flexibility! Productivity results from dynamically customized OS environments, matching operator tasks and machine

Real-time control kernel option allows quick software development for industrial control applications other tools and utilities for office desk-top personal computing functions, local area networks to Ethernet, AppleTalk, Omninet, ArcNet, PC-Net (Sytek) — from micro to mainframe command, control and communications Distributed processing application programs are easily developed

- Extreme organizational flexibility, each directory another environment
- Multiple Commands per line
- Aliases (complex series of commands known by simple names) with variable passing
- Named Directories with absolute password security
- Full-screen command line editing with previous command recall and execution
- Shells and Menu Generators, with shell variables
- Command-file search Paths, dynamically alterable
- Screen-oriented file manipulation and automatic archiving and backup
- 512 megabyte file sizes, 8 gigabyte disks handled
- Auto disk reset when changing floppies
- TCAP database handles characteristics of over 50 computers and terminals, more easily added
- Tree-structured online help and documentation subsystem
- 76 syntax-compatible support utilities

Your missing link has been found — Z! Now fly with eagles! Fast response, efficient resource utilization, link to rest of computing world — shop floor to executive suite, micro to corporate mainframe. Call 415/948-3820 for literature.

Echelon, Inc. 101 First Street • Suite 427 • Los Altos, CA 94022 • 415/948-3820 you return the item for a warranty repair then flux or lack thereof may give you away.

The corollary is that if you ever need to repair or inspect a circuit board, the presence of flux on a clean board invariably indicates some sort of repair or rework occurred. Many times the parts replaced will have failed again and this will give you a clue as to what needs fixing. You should also inspect the joints to be sure the previous technician properly soldered them, sometimes that's the problem. Soldering failures are the very reason for this article.

It wouldn't be fair to spend so much time talking about removing flux without discussing how to do so. Let's imagine that you want to clean an entire board and that you're sure there are no parts on the board that could be damaged by cleaning it, such as switches, relays etc. You'll need a solvent. You can purchase spray cans specifically labeled for removing rosin based flux. You can also use some common solvents such as Isopropyl Alchohol or VMP Naptha, both can damage plastics so don't use these if there are vulnerable parts. Don't use paint thinner as most leave some residues after evaporating. And never, **NEVER** use gasoline or kerosene, both are far too dangerous to ever use as a cleaner or solvent and that which makes them so hazardous also makes them do a lousy job.

Next you'll need a safe place to work, even the recommended solvents all produce vapors that must be ventilated. Don't work near any flames, open or closed, or anything that can spark, or make sure such things are turned off while you work and until the fumes have been expelled. Label any switches so that no one can accidentally turn them on, while you're working.

You will also need a place to work where the liquid solvent and flux can NOT cause physical damage. Don't work above a fine piece of furniture, a finished floor, or wall-to-wall carpet. All of these solvents can disolve finishes, leave white streaks, and the rosin can stain fabrics. And some of them, the one that Radio Shack sells for example, will really do a number on plastics. In fact the Radio Shack brand will mar plastics even if just the overspray hits them, so work well away from instruments or other objects in plastic cases.

Spray the solvent onto the board while holding it at a slight incline, work from top to bottom so the dissolved flux flows away from the board. If the flux is particulary stubborn use a retired toothbrush (test first to be sure the solvent won't disolve it) scrubbing around the connections to encourage the cleaning action.

To clean one or two connections, such as after a repair, the following technique is easier and safer. Take a cotton swab, hold it right up to the spray nozzle, and spritz it just enough to wet it. Roll the swab around the joint, don't scrub as the sharp edges will catch and tear it up. The cotton will absorb the flux and leave a neat looking repair. You may need to do it several times for maximum cleaning. By the way the Radio Shack product is great for this job, just be sure that when wetting the cotton tip that any overspray will not touch plastic surfaces.

Solder

Electronics solder is an alloy of 60% tin and 40% lead. These proportions

(continued on page 27)

APPLE][,][+,///,//e &//c OWNERS

UPGRADE THAT TIRED 6502 TO 16 BITS !!

65802 CPU $49.95

16 bit version of the 6502. Pin for pin and completely software compatible with the 6502 CPU. You can upgrade your Apple][,][+, ///,//e or//c to a 16 bit computer simply by replacing the 6502 with the 65802 without losing the ability to run any old software.

ProDOS ORCA/M (list $79.95) $69.95

This ProDOS version of ORCA/M comes with the complete 65802 instruction set. If you intend to develope software for this new CPU, then this package is a must. Chosen by the designers of the 65802 as the standard 65802 assembler.

16 Bit Upgrade Starters Package $109.95

This package includes 65802 CPU and the ProDOS ORCA/M. All you need to start.

Coming Soon: FORTH, PASCAL P-code Upgrade & MORE!

TO ORDER, SEND CHECK OR MONEY ORDER TO:

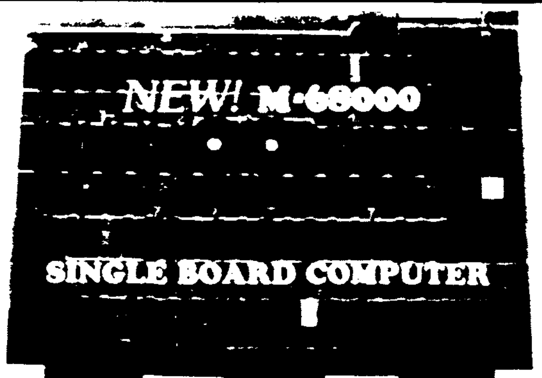
ALLIANCE COMPUTERS
PO BOX 408
CORONA, NY 11368

POSTAGE AND INSURANCE INCLUDED. (718) 426 2980

All CPU's will be sent by Postal Service, 1st class insured or UPS, insured. Please specify USPS or UPS. UPS doesn't deliver to POB's. Software will be sent by UPS Blue Label. If you want UPS Next Day Air, add $5.00 (CPU's only!). Most all orders sent out same day. COD add $3.00. APO's and FPO's welcomed.

Foreign orders: Please make payment in US dollars drawn on a US bank. Add $5 for Registered Mail and Air Mail Postage (except Canada). No foreign COD's.

PLEASE INCLUDE YOUR PHONE NUMBER WITH ORDER



FEATURES:

16 bit Motorola 68000 CPU operating at 5 MHz or 10 MHz, 20K of on board fast static RAM, 16K bytes of on board EPROM space, 7 autovectored interrupts, 3 memory/device expansion buses, 2 serial communication ports (RS-232 C), 16 bit bidirectional parallel port, 5-16 bit counter/timers with vectored interrupt and time of the day clock. On board monitor allows to download and debug programs generated on APPLE II, TRS-80 and CP/M using our M68000 Cross Assembler.

PRICE:

M68K Bare board with documentation	$ 99.95
M68MON monitor & mapping PROM's	$135.00
MD512 Memory/Disk Contr. (Bare Board)	$ 99.95
M68KE Enclosure with P.S. and card cage	$249.95
M68000 Cross Assembler	$149.00
M68K Documentation only	$ 15.00
Shipping & handling (Domestic)	$ 4.50
(foreign)	$ 20.00

CALIFORNIA RESIDENTS ADD 6% TAX

EMS Educational Microcomputer Systems

P.O. BOX 16115, IRVINE, CA 92713

(714) 854-8545

Build a S-100 Floppy Disk Controller Board

WD2797 controller for CP/M 68K

by Joseph Kohler

Any discussion of a floppy disk controller board which ignores the software to run the controller is incomplete. The board described in this article is presently being used with the 68008 CPU card given in reference 3 so all software will be for the 68008. The controller may be used with any S-100 CPU card provided the CPU chip is fast enough to perform the time critical functions of the routines mrd and mwr discused later.

If you wish to run CP/M 68K or any other operating system you will need a floppy disk controller board and a boot rom. Here is a design using the WD2797 which operates both 5 and 8 inch drives in single and double density.

Hardware

Addressing: A 8131 (see Figure 1) is used together with dip switches and pullup resistors to select 8 addressess in the I/O space to which the board will respond. Assertion of MATCH-H, SOUT and PWR results in assertion of PUT-H and PUT-L (see Figure 2). Writes to the board with A2 set to 0 access the WD2797 (see Figure 3) and with A2 set to 1 they access the control latch (see Figure 2). Assertion of MATCH-H, SINP and PDBIN results in the assertion of GET-H and GET-L (see Figure 2). Reads from the board with A2 set to0 access the WD2797 (see Figure 3) and with A2 set to 1 they access the wait flip-flop (see Figure 4).

Clock: A 4MHZ clock (see Figure 1) is divided by 2 and 4 using two flip-flops to produce clocks of 2MHZ and 1MHZ for use by the WD2797. Bit 5 of the control latch determines whether a 1MHZ or 2MHZ clock reaches the WD2797. This choice is made by using two tristate drivers of an LS125 (see Figure 4) and enabling one or the other to pass either the 1MHZ or 2MHZ signal to the WD2797. The WD2797 requires the 1MHZ for 5 inch drives and the 2MHZ clock for 8 inch drives.

Control latch: Overall control of the board resides in an LS273 8 bit latch. The meanings of the various bits are shown in Table 1.

This design simplifies the hardware at a slight cost in software complexity. A byte must be kept in ram which reflects the state of the LS273, since the LS273 is write only. In order to make a change you must get the ram copy, change only the desired bits in the ram copy and then write it to the LS273.

When RESET or POC is asserted so is MCLR-L, which initializes the LS273 so that no drive is selected and MOTOR-ON-Lis negated. The EPROM is enabled and PHANTOM is asserted on the S-100 bus (provided the switches to use the EPROM and PHANTOM are on).

Wait flip-flop: The read which references the wait flip-flop is simply used to create a pulse which clocks the wait flip-flop, with no intent to return data to the CPU. Assertion of POC or RESET clears the wait flip-flop.

My first exposure to the idea of stalling the CPU was in an early issue of *BYTE* (see reference 1) where a USRT was employed to serve as a floppy disk controller. The wait flip-flop is just a device to synchronize data transfers between the CPU the and WD2797.

INTRQ: If an error occurs during data transfers between the CPU and disk the WD2797 asserts INTRQ. Furthermore, INTRQ once asserted remains asserted until the WD2797 status is read. As long as INTRQ is asserted the wait flip-flop is held in a clear state no matter what happens at its clock input. Thus in case of error the routines mrd and mwr run to completion (transferring meaningless data) without wait states. INTRQ is negated by the routine mbusy when the status is read (see Figures 3 and 4).

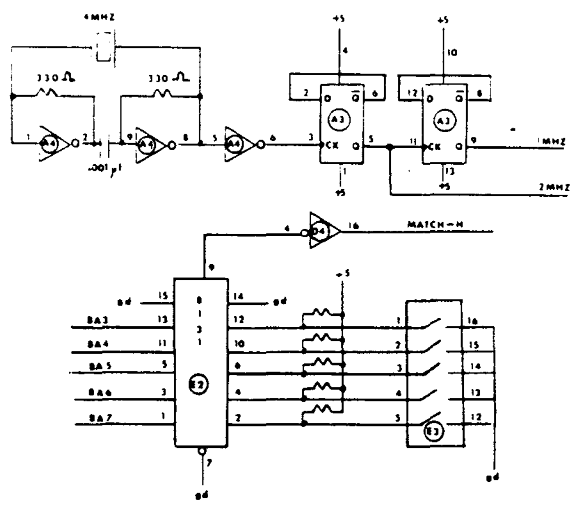


Figure 1: Clock and base port selection.

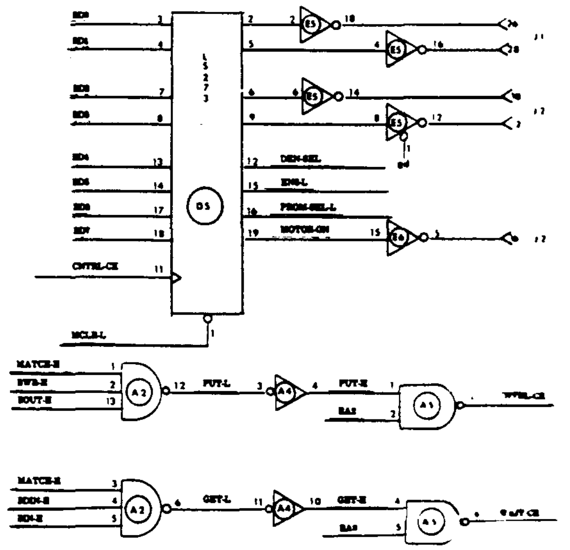


Figure 2: Control latch and decoding for CNTRL-CK and WAIT-CK.

BA0	5	A0
BA1	6	A1
BA2	3	$\overline{\text{CS}}$
GET-L	4	$\overline{\text{RE}}$
PUT-L	2	$\overline{\text{WE}}$
BD0	7	DAL0
BD1	8	DAL1
BD2	9	DAL2
BD3	10	DAL3
BD4	11	DAL4
BD5	12	DAL5
BD6	13	DAL6
BD7	14	DAL7
2797-CK	24	CLK
DEN-SEL	37	$\overline{\text{DDEN}}$
EN5-L	17	5/8
DRQ	38	DRQ
INTRQ	39	INTRQ
Unused	19	$\overline{\text{MR}}$

Figure 3: WD2797 data and control.

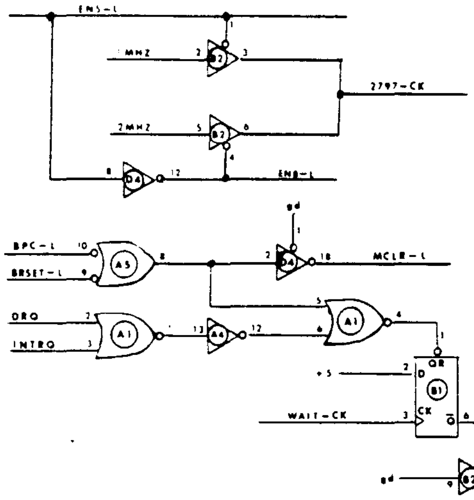


Figure 4: Clock selection and wait sync circuit.

bit 0	alow selects 8 inch drive 0
bit 1	a low selects 8 inch drive 1
bit 2	a low selects 5 inch drive 0
bit 3	a low selects 5 inch drive 1
	At most one of the above bits should be low at the same time.
bit 4	a low (high) selects double (single) density
bit 5	a low (high) enables the 1MHz (2MHz) clock and the buffers to operate 5 (8) inch drives
bit 6	a low (high) enables (disables) the EPROM and the PHANTOM
bit 7	a low (high) negates (asserts) the 5 inch motor on signal

Table 1

DRQ: Whenever the WD2797 wants to transfer a byte of data it asserts DRQ which clears the wait flip-flop waking up the CPU. Details may be found in the discussion of the routines mrd and mwr (see Figures 3 and 4).

Drive interface: The interface between drives and the WD2797 follows a well defined stándard. The rules are that signals received are pulled up by a 150 ohm resistor, and signals sent must be driven by something capable of pulling a line (which is pulled up by a 150 ohm resistor) low. An LS244 or LS240 meets this requirement. There is also enough hystersis to reliably receive the raw read data.

Bit 5 of the control latch is used to select the LS240 and LS244 (see Figures 6, 7 and 8) drivers and receivers for whichever type of drive is being read or written. The two signals which do the selection are EN5-L for 5 inch and EN8-L for 8 inch drives. Note EN8-L is the inversion EN5-L (see Figure 4) so exactly one of these signals is active at any time.

EPROM: Reads from the EPROM occur when PROM-SEL-L is asserted by the control latch, A15 is 0 and SMEMR is asserted (see Figure 12). In order to transfer ROM data to the CPU the dip switch E3, which connects pin 10 to pin 7 must be closed (see Figure 11), otherwise the OE pin of the EPROM (see Figure 11 and 12) will be pulled high by a pullup resistor and BOARD-TO BUS-L will not be asserted. If dip switch E3, which connects pin 11 to pin 6 is closed and the control latch asserts PROM-SEL-L then PHANTOM is asserted.

The EPROM socket has 28 pins so any EPROM from a 2716 to a 27128 can be used. So far a 2716 has proved adequate since it is used only to boot CP/M 68K. NOTE: The pin numbers in Figure 12 refer to the 28 pin socket.

The program in the present EPROM simply moves a loader program to RAM (beginning at $9000) and then jumps to $9000 to execute the loader. My present system uses two Digital Research 64K RAM cards. The low order board is set with half phantom and the high order with full phantom enabled. If more boards are added they should have full phantom enabled.

Adjustment procedure: The WD2797 requires some initial adjustments in order to operate.

1. Make sure switches E3 pin 6, 7 and 8 are all open. If you want to be more careful also remove the EPROM.
2. Do not connect cables to disk drives.
3. Place the board in an S-100 system with power off. Turn on power.
4. Press reset to force control latch outputs DEN-SEL and EN5-L low, which sets the DDEN input to the WD2797 low and the 5/8 input to 5 inch.
5. Momentarily short pins 19 and 20 to perform a master reset on the WD2797.
6. Close switch E3 pin 8 to enable TEST on the WD2797.
7. Set your scope to 0.1us/div and place probe on pin 31 of the WD2797. Adjust the 10K pot until you see a waveform 200ns wide.
8. Now put your probe on pin 29 of the WD2797. Adjust the 50K pot until you see a waveform 250ns wide.
9. Set your scope to 1us/div and adjust the 5-60pf variable capacitor until you see a squarewave with pulse width 2us i.e. a frequency of 250KHZ.
10. Open switch E3 pin 8 to disable TEST on the WD2797.

For more information on the WD2797 see reference 2.

The discussion below assumes a5 points to the base port of the controller board. For example, if the address selection switches are all on the base port is 0, so a5 must be set to $ff-

ff00. Also the equates

```
status    equ 0
fcmd      equ 0
```

are in effect.

Execution of a read: The instruction

```
move.b   status(a5),d1
```

reads the status register of the WD2797. The 68008 CPU board executes this instruction by placing the address $ffff00 on address lines A0 through A23 (see Figure 9). Address signals A3 through A7 are passed through E1 to E2, an 8131, which then asserts MATCH-H because BA3 through BA7 are all low (see Figure 1). Next the 68008 CPU board asserts SINP and PDBIN. The WD2797 board receives these signals through (the always enabled LS244) D2 so BIN-H and BDIN-H are asserted on the board. Now pins 3, 4 and 5 of A2 (see Figure 2) are all high so GET-L is asserted. This forces the assertion of GET-H which in turn results in the assertion of BOARD-TO-BUS-L. Now the WD2797 has a low on CS because A2 is 0 and a low on RE since GET-L is asserted. The lines A0 and A1 now select the register within the WD2797 whose contents will be sent to its DAL lines and therefore appear as data on BD0 through BD7. Finally, the data appears on the S-100 bus through the enabled driver C2 (see Figure 9).

Execution of a write: The instruction

```
move.b   d4,fcmd(a5)
```

sends a command to the WD2797.

The address information and the assertion of MATCH-H is the same for read and write. After the address information is setup the 68008 CPU board asserts SOUT and places d4.b on DO0 through DO7. Following a suitable delay the 68008 CPU board asserts PWR. The WD2797 board receives PWR and SOUT through D2 (see Figure 10) so BWR-H and BOUT-H are asserted. Now pins 1, 2 and 13 of A2 (see Figure 2) are all high so PUT-L is asserted. This in turn enables the buffer C1 (see Figure 9) so the data on DO0 through DO7 arrives on BD0 through BD7. Now the WD2797 has a low on CS because A2 is 0 and low on WE since PUT-L is asserted. The lines A0 and A1 now select the register within the WD2797 to which the data on its DAL inputs will be sent.

Software

The CP/M 68K BIOS is too long to fit into this article but you can follow the code to read and write sectors by assuming that a4 points to a RAM buffer, d6.w has the track number and d7.w has the sector number of the host disk.

The software discussion below is for 5 inch drives, but is almost identical for 8 inch drives. mdriv is the only routine which is different and it is simpler for the 8 inch drives.

```
read:
           clr.b      erflag
           move       #3,retry
read1:     movem.l    a1-a2/d6-d7,-(sp)
           bsr        rd_mini
           movem.l    (sp)+,a1-a2/d6-d7
           tst.b      erflag
           beq        read2
           subi       #1,retry
           bne        read1
           bsr        err_routine
read2:     rts
```

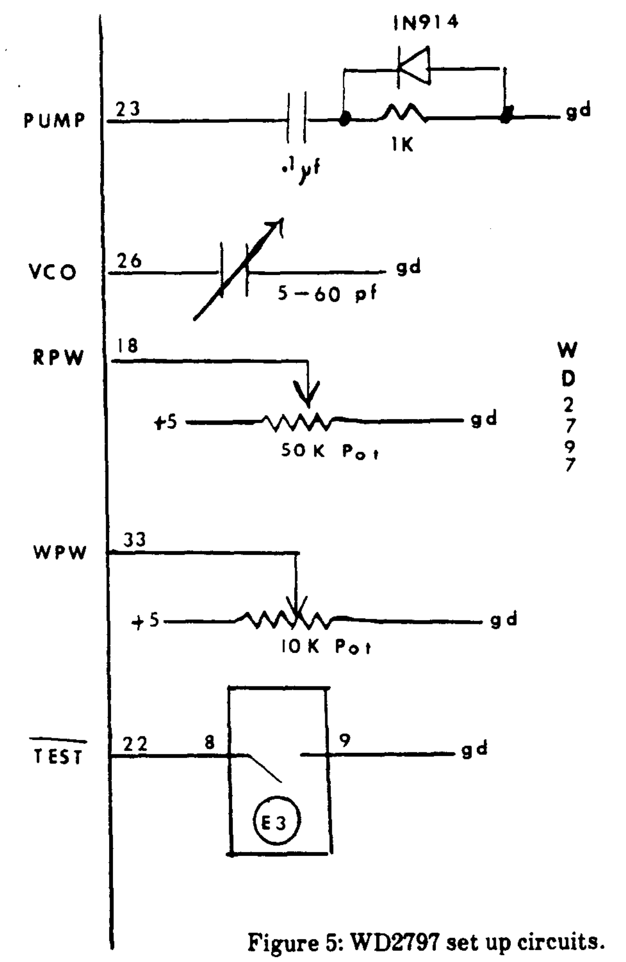
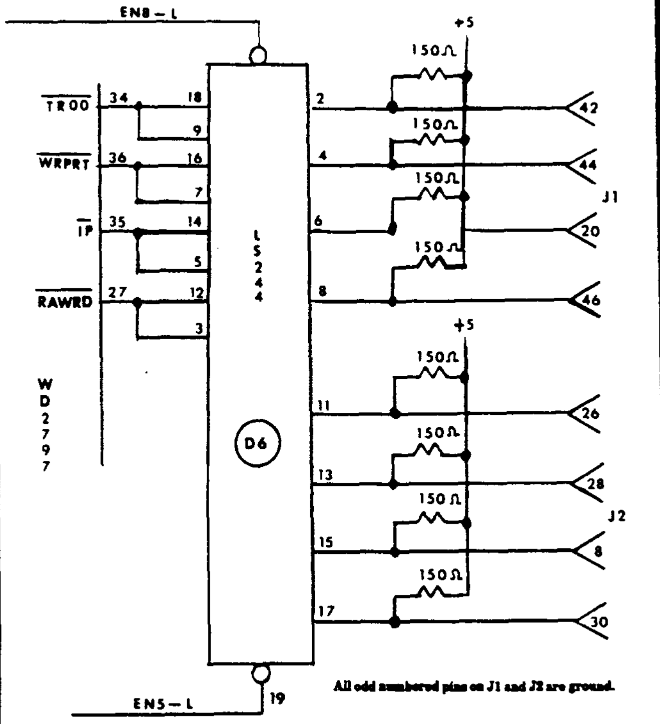


Figure 5: WD2797 set up circuits.



All odd numbered pins on J1 and J2 are ground.

Figure 6: 5 and 8 inch drive control.

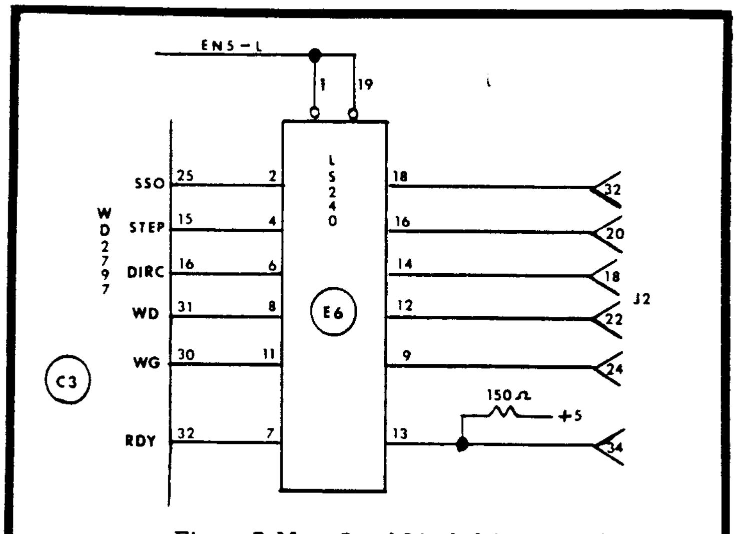


Figure 7: More 5 and 8 inch drive control.

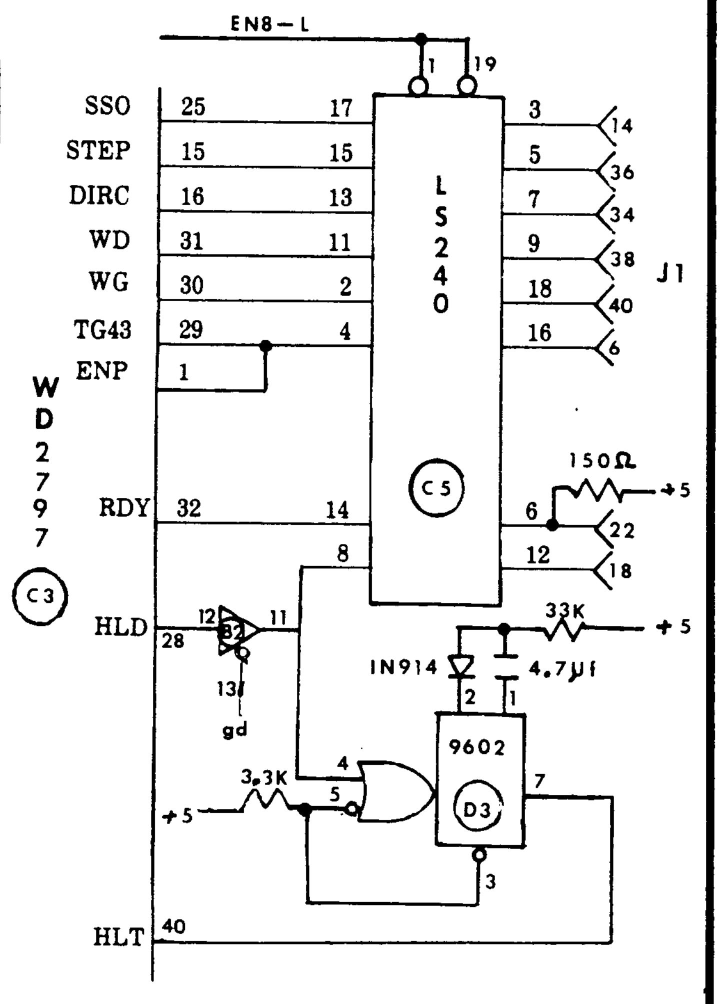


Figure 8 : Eight inch drive control.

The details of the err__routine, which just prints, are omitted.

rd__mini: At entrance to rd__mini register sp is saved so that in case an error occurs during processing the error routine can restore sp. This relieves the other routines of the burden of keeping track of sp. Upon return from rw__mini d4.w has the SSO bit set and all other bits are 0. The WD2797 read command is or'ed into d4.w and the command placed into d1.w for use by mrd, which returns, after executing the read, with the WD2797 status in d1.w. The relavent bits are checked for a read error. If a read error has occurred bit 6 of erflag is set. The final return is then made to read.

rwmini: The routine parmx merely sets a5 to point to the base port of the board and sets d5.w to the disk unit number. Each cylinder is divided into 2 tracks i.e. cylinder n of a diskette consists of tracks 2n and 2n. The track number is converted into a cylinder number, placed in d6.w, and a side select bit, placed in d4.w. mdriv is called to select the drive, get the drive up to speed and make sure it is ready. mdriv returns with the Z-bit clear if this drive is not the same as the previous one. In this case mrdid is called to locate the head of the newly selected drive. Next the byte count for a read or write command id placed in d2.w and the sector register of the WD2797 is loaded. The hardware details of this write to the sector register are almost identical to the move.b d4,f-cmd(a5). The addq ‡1,d7 is done because the BIOS starts sector numbering at 0 and the WD2797 starts it at 1.

mdriv: The first 3 instructions prepare data for the control latch so that MOTOR__ON is asserted, PROM is off, 5 inch is selected, double density is selected and the proper unit is selected. This information is sent to two places, one is the control latch and the other is dkctrl which holds the ram copy of the control latch. A call to mbusy is made to see if the drive is ready. If not, a delay loop is executed. A call to mbusy is again made to see if the drive went ready. The idea is to sample the ready line of the selected disk. If it is ready then exit, otherwise delay a short while and then sample again. After delay2 tries at this you can assume the drive is not going ready so error bit 3 is set and flow passes to merr, the error routine.

If the drive does go ready then prev__51, which keeps track of the current drive is updated. mdriv returns with the Z-bit clear if the current request is on a different drive from the previous request.

mrdid2: This routine begins by aiming a0 at a 6 byte area into which the disk id will be read. Next a byte count is placed in d2.w (1 less than the actual number of bytes transferred) and a readid command is placed in d1.b. mrd is called to tell the WD2797 to perform the readid. The WD2797 status is returned in d1.w following the command. The and instruction sets the Z-bit if there is no error, otherwise clears it.

mrdid A call to mrdid2 asks it to read an id field. If no error occurs the track register of the WD2797 is updated. If an error does occur mrdid steps out 1 cylinder and tries again. If all goes well this time the track register of the WD2797 is updated, otherwise bit 3 of erflag is set and the error exit is taken.

The stepout was chosen because once CP/M 68K is running cylinder 0 is not accessed. Also this seemed to work in most cases.

mseek: Recall that rwmini set d6.w to the desired cylinder. This is compared with the cylinder register of the WD2797. If they are equal no seek is necessary, else a seek command is sent to theWD2797. A call to mbusy is made to wait for completion of the the command and to get the status. The and instruction clears the Z-bit if an error occurred in which case bit 5 of erflag is set and the error return taken. If all is well mseek returns to the caller.

mrd: The call to mdisint saves the CPU status register, disables interrupts, sends the command in d1.b to the WD2797, aims a1 at the WD2797 data port and aims a2 at the wait flip-flop. The code which follows is the loop to transfer data from the WD2797 to memory. Execution of the move.b (a2),d1 merely sets the wait flip-flop, thereby stalling the CPU. When the WD2797 has a byte of data it will assert DRQ, clearing the wait flip-flop and waking up the CPU. The CPU then reads the data, placing it where a0 points, bumps a0 and decrements the byte count. This loop continues until the byte count number of bytes is transferred. If the WD2797 encounters a problem it asserts INTRQ, which also clears the wait flip-flop. INTRQ remains asserted until the status register of the WD2797 is read. In this event data is still transferred but it is meaningless. After the data is transferred the CPU status is restored.

mnbinit: This routine restores the CPU status.

mbusy: A short delay allows the WD2797 time to prepare its status register. Now the status register is continuously checked until the busy bit goes off. Finally, the WD2797 status is returned in d1.b.

You will notice 3 instructions are commented out. The reason for this is that they are priviliged instructions. CP/M 68K runs transient programs in user mode so when testing this program these instructions are not executed. However, the BIOS operates in supervisor mode so the *'s can be removed when the code is inserted into the BIOS.

mwr: This routine operates in essentially the same way as mrd except the data transfer is from RAM to the WD2797. The WD2797 asserts DRQ whenever it wants a byte of data. INTRQ operates here the same as it does for mrd.

merr: The control latch is set to deselect all drives and dkctrl is set to mirror the control latch. prev__51 is set so that mdriv will regard the next selection as new. Finally sp is restored and the return made to read (or write as the case may be).

wr-mini: This routine operates much the same way as rd__mini.

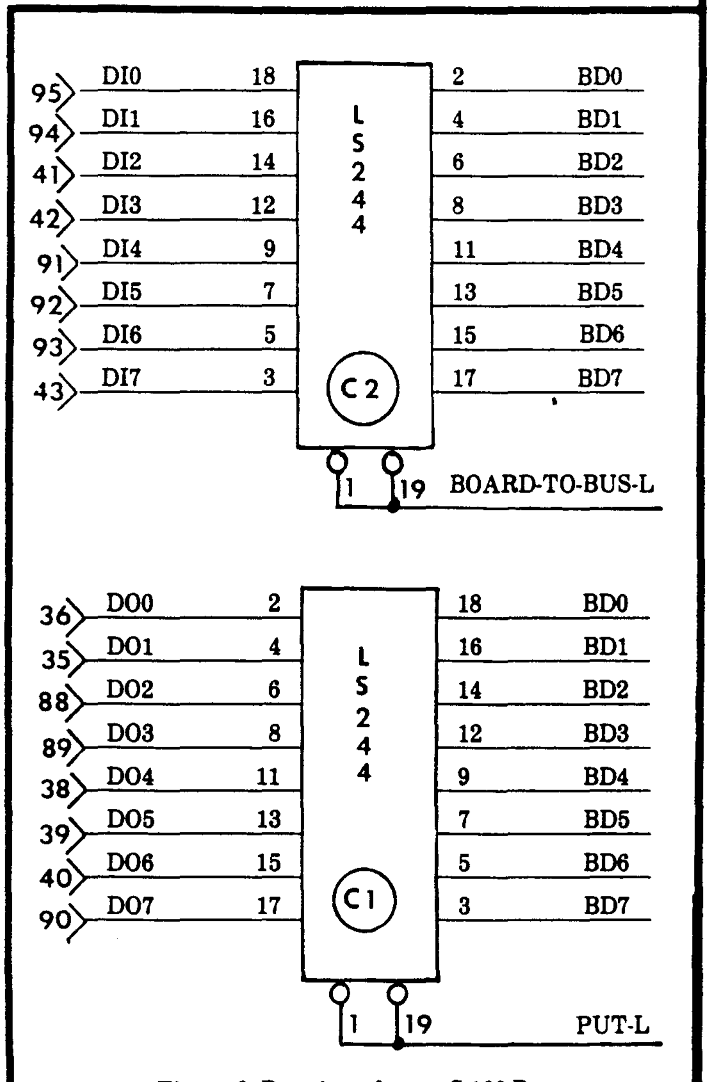


Figure 9: Data interface to S-100 Bus.

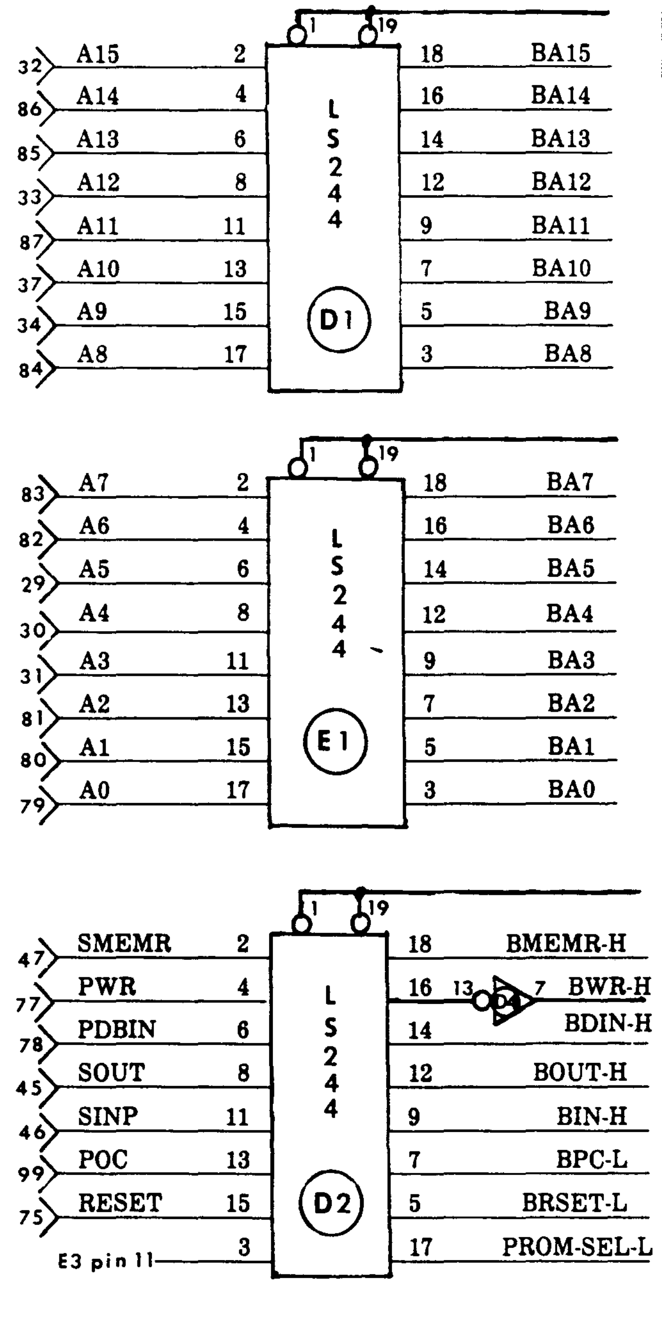


Figure 10: Address to Eprom and S-100 control bus connections.

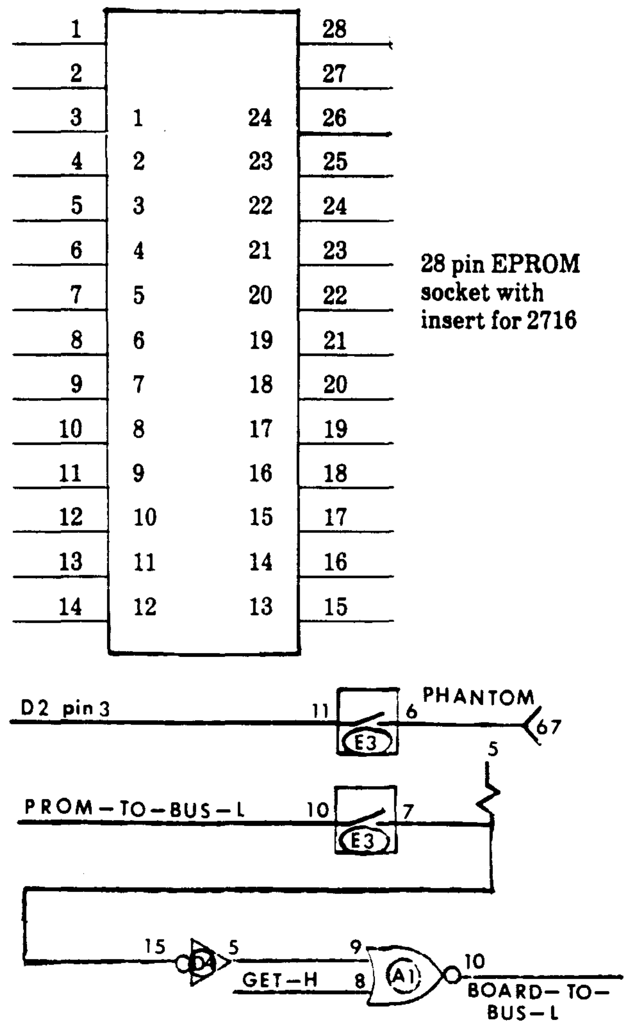


Figure 11: EPROM socket and PHANTOM.

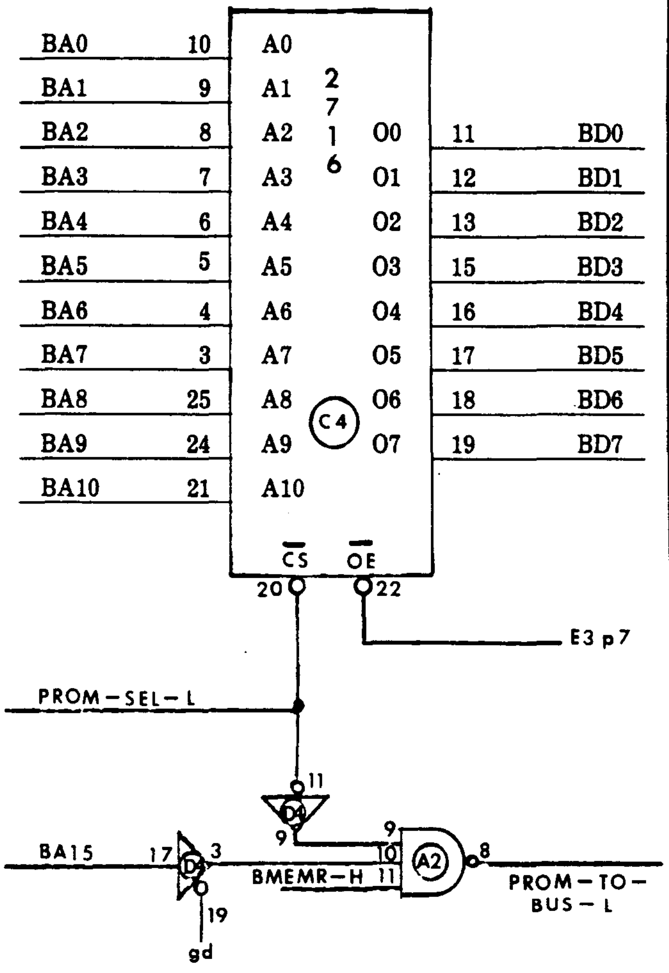


Figure 12: 2716 and EPROM control.

parmx: This routine references a data structure which holds information about the disk to be accessed. a3 is a pointer to this data structure. Details have been omitted. See listing.

```
Listing

parmx:
        move.b  phys_unit(a3),d5        * get physial unit number
        moveq.l #-1,d1                  * get all 1's
        move.b  base_port(a3),d1        * insert base port number
        move.l  d1,a5
        rts

delay   equ     20000   * loop delay to select different disk

*--------
* 2797 commands
*--------

rstor   equ     $08
readid  equ     $c4
writc   equ     $a8
readc   equ     $88
skcmd   equ     $1c
frcint  equ     $d0
stepout equ     $68

*--------
* port offsets from base port number in a5
*--------

status  equ     0
fcmd    equ     0
track   equ     1
sector  equ     2
ddata   equ     3
select  equ     4
ewait   equ     4
dens    equ     5
```

(continued on page 37)

Conclusions

We have been using this controller for 5 months with 5 inch TEAC and 8 inch Mitsubishi drives. With the 8 inch drives we have been using single sided disks in double sided double density mode with almost no errors. This can be done if the Mitsubishi drives are properly jumpered. On the 5 inch drives the error rate seems to be about the same as my friends who run IBM PC's or clones i. e. somewhat higher than on 8 inch drives. In all cases (both 5 and 8 inch disks) the errors seem to occur when a disk is first used. Once a disk has been formatted and used for a while it usually runs error free. Our department has two Stride computers (both run CP/M 68K) and we have had no trouble interchanging disks with these systems. At present our 68008 systems have a 68008 CPU card, this disk controller with 5 and 8 inch drives, two Digital Research 64K RAM cards, two Digital Research semi disks and a serial I/O card. Students then interface whatever other boards they need. ■

References:

1. *Build This Economy Floppy Disk Interface, by Dr. Kenneth Welles, Byte February 1977, page 34.*

2. *1983 Components Handbook, Western Digital Inc.*

3. *Using The S-100 Bus and the 68008 CPU, issue 16, The Computer Journal.*

Soldering
(continued from page 20)

give solder the ability to turn liquid at a workable temperature. There are different solders for specialized work and these are usuaully not suited for electronics because they turn liquid at higher temperatures. There is one type of solder that is 63% tin and 47% lead which is very good for electronics but it is harder to locate than the standard 60/40 type.

The only real choice in buying solder is to select the best gauge for the work at hand. Gauge refers to the thickness of the solder. Some common gauges are .032" which is very fine, .040" and .050" both good sizes for kit building, and .062" and up.

Actually gauge is not critical at all, if you happen to have a very fine gauge then you'll need to feed more into the joint to produce a good result, with heavier gauges just a little is all that's required. It becomes something of a personal preference as to what you like best. Just be sure to purchase in economical quantities, small packages can run out quickly and cost more to buy.

Hints And Tricks

Throughout we've talked about the theory of soldering, the tools used, the solder itself and a most exhaustive discussion of a sticky brown glob (rosin flux). Interspersed with these topics we've included some basic techniques. So to wrapup lets mention some odds and ends that can be helpful. If readers of this Journal know some more then please contribute, knowledge is only valuable by its sharing.

Cleaning Traces

Modern circuit boards are made with copper traces which are then plated with a nickel/tin alloy to keep them from corroding as rapidly as they would if left as bare copper, such traces are silver in color. Boards may also have a solder mask, this is a coating that repels solder, reducing the possibility of solder bridges, the mask is either a pale green or blue color. If a board has either or both of these then you don't want to clean it. It's not needed and would only damage the plating and the mask. However, some small boards may show copper colored traces. These should be lightly cleaned with fine steel wool or one of those nylon scouring pads until the traces just begin to shine. Do this once just prior to beginning work on the board. Even so you may find that such boards are more dificult to work on, a fine gauge solder may be helpful because it provides more flux.

The other item that may need cleaning is the leads of the parts to be installed. It's always a good idea to look at the lead and if particulary dull then lightly scrape it with a utility knife or pull it through the closed jaws of a needle-nose pliers. Ceramic disc capacitors are sometimes coated with a wax substance, this may extend down the lead far enough to serioulsy interfere with solering, it also keeps the lead shiny, for this reason I always clean these leads even if they don't seem to need it.

Parts Positioning

An experienced hand once said that soldering was easy, but keeping the parts from falling out before you can solder them is the real trick. In almost all instances you have to solder a PC board bottom side up, you insert the part and then turn the board over and all too often the part will either fall out or slip away from the board. In industrial situations the same problem has led to several solutions, most of which are not really useful to the casual builder.

One industial method involves cutting the lead just above the pad and then bending or staking it over. This works great, too great in fact. In the next article on unsoldering this becomes unsoldering nightmare number one, it's also more than is really needed for casual work. The second method involves bending or forming the leads so that when inserted into the PC board they press against the holes enough to hold the part. This is fine for kit builders but it requires special tools to always achieve it. Actually, half the time this will occur by chance and you'll find the part stays in place by simple friction.

A very simple variation of the formed lead method is to insert a part and if it is loose then hold it down with one hand and with a needle nose pliers in the other grasp the lead about a quarter inch above the board and pull it sideways parallel to the board for about a quarter or eighth of an inch. This will generate enough friction to hold the part until soldered. The real diference here is that the industrial method is done to the part prior to inserting it while this variation is done after the part is in place.

Sockets present a slightly diferent problem as they don't have enough lead length, again don't stake one or two pins as that makes unsoldering nearly impossible. Yet some people do on the grounds that unsoldering a socket is impossibly dificult anyways and extremely rare as well. My method is to insert the socket and then turn the board over onto the workbench, ideally onto a piece of cloth or cloth over foam rubber. Then while soldering I press down slightly to force the socket flush to the board. I only solder two diagonal pins and then I check to be sure the socket is flush and that the pin one end is correctly oriented before completing the other pins. Of course I often do several sockets at a time so this isn't as tedious to do as to describe. This doesn't always work because sometimes other parts extend above the socket, that's why the foam rubber but if that doesn't help I keep a few small blocks of wood around that can be strategically placed under the socket to push it up.

Wires and Lugs

Back when kits included lots of soldering of wires to lugs there were two problems that may have driven more than a few folks to swear off kit building. Both problems have reasonably good solutions and since there are still a few instances where they may be needed we'll mention them.

When soldering a wire to a switch lug most kits will tell you to make a loop of wire around the lug and then solder. This forms a mechanical connection prior to the soldering and it's alright to do it this way. But if you ever need to undo the connection you'll find that usually the wire must be cut and then the loop unsoldered. And guess what, the wire will then be too short to reattach, you'll have to splice it or take it off the other end and use a new length of wire. I generally don't make a loop but rather a 90 degree bend in the wire and then slip it through the lug, solder it and then trim away any excess wire. Sometimes if it holds I may just slip it through the lug with no bend at all. With one exception, I do make loop type connections on large wires or wires that will carry power.

The other problem occurs when you have to solder a wire to a lug that is buried within other lugs and wires.

What happens is that as you make the connection you notice a foul odor of burning wire. The cause is that one or more of the surrounding wires is being charred by the hot barrel of your soldering iron. One way to avoid this aggravation is to use a battery powered iron if you have one (they only get hot at the tip). The other solution is to keep the wires pulled back. Extra hands are great for this if you have them, your own that is. don't try to use someone elses, like a spouse or relation. You see the hot iron may burn the other person's fingers and then instead of charred wires you'll have a charred relationship.

A real easy way to hold back the wires is with hold-backs. What are they and from what obscure mail order firm do you buy them? You don't buy them, you make them. Just get a few paper clips some bigs ones and some little ones, add a few rubber bands some short ones and some long ones. Now take your wire cutters and cut the paper clips just around the bend of each loop. Tie a rubber band with a half hitch over one loop. To use them just grab the offending wires with the open loop and then gently stretch them away by fastening the rubber band to keep them out of the way while you solder. There will be always be something that the rubber band can loop over. You can make as many hold-backs as you need and feel free to modify them in any fashion.

Conclusion

That pretty much wraps up soldering, the only thing left is for you to go out and try it. If you're just begining start with a small project or with a scrap circuit board, some surplus firms sell such items. Then you'll be ready for the next feature, unsoldering.

Sources

The following companies publish catalogs that you may wish to send for. They are listed only for that reason and no assertion as to their worthiness or lack is implied. As always you should exercise reasonable care when making purchases by mail.

Radio Shack catalogs are available at most stores. Tools and parts, plus some simple kits.

The Heath Company, Benton Harbor, MI 49022, (800) 253-0570, many kits from simple to complex, including computer kits and training courses, test instruments, also some tools.

Edlie Electronics, 2700 Hempstead Turnpike, Levittown, NY 11756, (800) 645-4722, carries electronic parts and tools, for soldering and circuit repair, also some simple kits. Catalog is good reference for what is available.

Jameco Electronics, 1355 Shoreway Road, Belmont CA 94002, (415) 592-8097, many electronic parts, also tools and supplies and some kits. Again a good catalog to have for reference especially electronic spares etc.

Active Electronics, P.O. Box 8000, Westborough, MA 01581, (800) 343-0874, large stock of electronics parts and also lists tools, no kits though.

Priority One Electronics, 9161 Deering Avenue, Chatsworth, CA 91311, (800) 423-5922. Electronic parts and products, also soldering tools and circuit repair products.

The above is only a partial list of firms that supply tools and may also sell kits. Many others sell only kits, you should check the pages of this and other technically astute publications for the names and addresses of these firms. ■

OVERCOME
FORTRAN | PASCAL | C
LIMITATIONS WITH

$129
NO LICENSE FEE

NO LIMIT

Ver. 2.0 For:
MS FORTRAN/Pascal/C
IBM Professional
and R-M FORTRAN
SuperSoft FORTRAN

A library of over 90 Assembler routines transform FORTRAN, Pascal and C language compilers into the flexible, responsive, complete languages needed for the microcomputer environment. Hundreds of NO LIMIT owners are creating highly interactive software systems, often utilizing existing mainframe code and saving time and money in the process. Ver. 2.0 features include:

EXTENSIVE GRAPHICS	(Get, Put, Paint, Color, Dot, Line, Box, Circle, Ellipse, Large Characters)
FULL SCREEN CONTROL	(Windows, Cursor, Read/Write Screen)
STRING MANIPULATION	(Match, Compare, Concatenate/Extract, Pack, Justify, Zero Fill)
KEYBOARD CONTROL	(Read Key During Execution, String Read With Edit)
FILE MANAGEMENT	(Exist?, Rename, Delete)
COMMUNICATIONS	(Full Interrupt Driven to 9600 Baud, Set Com Line, Send/Receive, Line/Modem Status)
INTERRUPTS	DOS Execution, Program Execution
OTHER FEATURES	(Command Line Read, DOS 3.0 Directories, Peek, Poke, Random Numbers, System Status, etc.)

And to complement NO LIMIT, the I/O PRO development system allows creation and editing of FORTRAN/Pascal/C callable screens. This word processor type system pays for itself several times over on your first project. $260.

For immediate solutions to your programming needs call
(800) 562-9700
(512) 251-5543 (Texas)
M | E | F Environmental Inc.
P.O. Box 26537 Austin, Texas 78755

© Trademark of Microsoft Inc. Supersoft Inc. IBM Corporation Ryan-McFarland and Microsoft Inc. respectively

Surplus Parts Resource

Here's a catalog any serious computer tinkerer needs. It's a treasure-trove of stepper motors, gear motors, bearings, gears, power supplies, lab items, parts and pieces of mechanical and electrical assemblies, science doo-dads, goofy things, plus project boxes, lamps, lights, switches, computer furniture, and stuff you might have never realized you needed.

All at deep discounts cause they are surplus!

Published every couple of months, and consecutive issues are completely different. Send $1.00 for next three issues.

JERRYCO, INC. 601 Linden Place, Evanston, Illinois 60202

Apple II Graphics

(continued from page 13)

```
{*********************************************************}

PROCEDURE _setpartition(part:partition);
{switch between full screen graphics and mixed text/graphics}
BEGIN
  CASE part OF
    FULLSCREEN:  _wrhostbyte(_FULSCRN,Ø);
    MIXED:  _wrhostbyte(_MXD,Ø);
  END;
END;

PROCEDURE _selectpage(pagenum:integer);
{switch to specified graphics page}
BEGIN
  IF pagenum=1 THEN _wrhostbyte(_PG1,Ø)
  ELSE IF pagenum=2 THEN _wrhostbyte(_PG2,Ø)
  ELSE writeln('There is no page ',pagenum);
END;

PROCEDURE textscreen(pagenum:integer);
{switch to specified text screen}
BEGIN
  _selectpage(pagenum);
  _wrhostbyte(_TXT,Ø);
END;

FUNCTION _inrange(n,lolimit,hilimit:integer):boolean;
{test for value outside of limits--used to prevent drawing
 outside the screen boundaries}
BEGIN
  IF (n>=lolimit) AND (n<=hilimit) THEN _inrange:=TRUE
  ELSE _inrange:=FALSE;
END;

{end of APLGR/G}
```

LISTING 3. APLGR/L.INC

```
{APLGR/L.INC enables calling low resolution Apple graphics
routines from Turbo Pascal programs.
Requires the PCPI Z8Ø card (Applicard).
Copyright 1984 by N.T.Carnevale.
Permission granted for nonprofit use.}

{contains these routines:
PROCEDURE lorespatch;
  --installs the register-loading routines needed by setcolor,
    plot, hlin and vlin
PROCEDURE loresgr(pagenum:integer; part:partition);
  --invokes lores graphics
PROCEDURE clear_lores_screen(page:integer);
  --clears specified lores page
PROCEDURE setcolor(color:loreshues);
  --selects color for drawing
PROCEDURE plot(column,row:byte);
  --puts a point on the screen
PROCEDURE hlin(row,col1,col2:byte);
  --draws a horizontal line
PROCEDURE vlin(col,row1,row2:byte);
  --draws a vertical line

Some of these procedures call lores graphics routines
at the following ROM locations:
 SETCOL = ØF864H  set color
 PLOT   = ØF8ØØH  plot a point
 HLIN   = ØF819H  draw a horizontal line
 VLIN   = ØF828H   "      vertical line

This requires "poking" a few short machine language (65Ø2)
routines into the 65Ø2's RAM starting at location 9ØØ3H.
The parameters needed by these routines are "poked" into
locations 9ØØØ-9ØØ2H (bytes destined for the A and Y registers
and locations 2CH or 2DH).
}

TYPE
  loreshues=(BLACK,MAGENTA,DARKBLUE,PURPLE,DARKGREEN,GREY1,
             MEDIUMBLUE,LIGHTBLUE,BROWN,ORANGE,GREY2,PINK,
             LIGHTGREEN,YELLOW,AQUA,WHITE);  {lores colors}

CONST
  {low resolution constants}
  LOHRES=4Ø;            {# of pixels across the screen}
  LOVRES=4Ø;            {full screen vertical resolution}
  LOMIXVRES=4Ø;         {mixed mode vert res}
  _LORESPAGE1=$4ØØ;     {start of lores page 1}
  _LORESPAGE2=$8ØØ;     {but can't use page 2 with Applicard!
                         Overlaps with vital drivers!!!!}

  {easily accessible ROM routines for lores graphics}
  _LOCLRSCR=$F832;    {clears whole lores screen}
  _LOCLRTOP=$F836;    {spares four text lines at bottom}

  {The following addresses in the 65Ø2's RAM are used by


  setcolor, plot, hlin and vlin}
 _ASETCOL=$9ØØ3;  {set color}
 _APLOT=$9ØØA;    {plot a point at column,row}
 _AHLIN=$9Ø14;    {plot horiz. line at row v between col1 and col2}
 _AVLIN=$9Ø23;    {plot vert. line at col h between row1 and row2}

{*********************************************************}
{Next are routines to be patched into motherboard's RAM at
 $9ØØ3-$9Ø33 so that setcolor, plot, hlin and vlin can be used}


PROCEDURE lorespatch;
{installs the register-loading routines needed by
 setcolor, plot, hlin and vlin}
CONST
  LORESTUFF: array [$Ø1..$2F] of byte=(
    {_ASETCOL (9ØØ3-9ØØ9)--set color}
    $AD,$ØØ,$9Ø,$2Ø,$64,$F8,$6Ø,
    {_APLOT (9ØØA-9Ø13)--plot a point at column h, row v}
    $AD,$ØØ,$9Ø,$AC,$Ø1,$9Ø,$2Ø,$ØØ,$F8,$6Ø,
    {_AHLIN (9Ø14-9Ø22)--plot horiz. line at row v between
     col1 and col2}
    $AD,$Ø2,$9Ø,$85,$2C,$AD,$ØØ,$9Ø,$AC,$Ø1,$9Ø,$2Ø,$19,$F8,$6Ø,
    {_AVLIN (9Ø23-9Ø31)--plot vert. line at col h between
     row1 and row2}
    $AD,$Ø2,$9Ø,$85,$2D,$AD,$ØØ,$9Ø,$AC,$Ø1,$9Ø,$2Ø,$28,$F8,$6Ø
    );
   {Borland Pascal's "structured constants" feature is nonstandard.
    So, for that matter, is any hardware-specific code that might be
    generated even with standard syntax!  This just happens to be
    a quick and dirty way to define a table of bytes that represents
    65Ø2 instructions}
VAR source,dest,lnth:integer;
BEGIN
  source:=ADDR(LORESTUFF[1]);  {starting address of data to send}
  dest:=$9ØØ3;                 {where in the 65Ø2's RAM to put it}
  lnth:=$2F;                   {how many bytes to send}
  _wrhostdata(source,dest,lnth);
END;

{*********************************************************}


PROCEDURE loresgr(pagenum:integer; part:partition);
{switch to low resolution graphics on specified page}
BEGIN
  _selectpage(pagenum);
  _setpartition(part);
  _wrhostbyte(_LRS,Ø);
  _wrhostbyte(_GRFX,Ø);
END;

{*********************************************************}
{Elementary lores graphics procedures}

PROCEDURE clear_lores_screen;
BEGIN
  _callapl(_LOCLRSCR); {_callapl is in the file PCP.INC}
END;

PROCEDURE setcolor(color:loreshues);



{specify color to use for drawing}
VAR kolor:byte;
BEGIN
  kolor:=ORD(color);
  _wrhostbyte(_AREG,kolor);
  _callapl(_ASETCOL);
END;

PROCEDURE plot(column,row:byte);
{draw a point at specified location}
BEGIN
  IF _inrange(column,Ø,LOHRES) THEN
    IF _inrange(row,Ø,LOVRES) THEN BEGIN
      _wrhostbyte(_AREG,row);
      _wrhostbyte(_YREG,column);
      _callapl(_APLOT);
    END;
END;

FUNCTION _loresclip(n,lolimit,hilimit:integer):integer;
{called by hlin & vlin to prevent drawing outside screen margins}
BEGIN
  IF n<lolimit THEN _loresclip:=lolimit
  ELSE IF n>hilimit THEN _loresclip:=hilimit
  ELSE _loresclip:=n;
END;

PROCEDURE hlin(row,col1,col2:byte);
{draw horizontal line at "row" from col1 to col2}
VAR temp:byte;
BEGIN
  IF _inrange(row,Ø,LOVRES) THEN
    IF _inrange(col1,Ø,LOHRES) OR _inrange(col2,Ø,LOHRES)
    THEN BEGIN
      col1:=_loresclip(col1,Ø,LOHRES);
      col2:=_loresclip(col2,Ø,LOHRES);
      IF col1>col2 THEN BEGIN
        temp:=col1;
        col1:=col2;
        col2:=temp;
      END;
      _wrhostbyte(_AREG,row);
      _wrhostbyte(_YREG,col1);
      _wrhostbyte(_LOCXX,col2);
      _callapl(_AHLIN);
    END;
END;

PROCEDURE vlin(col,row1,row2:byte);
{draw vertical line at col from row1 to row2}
VAR temp:byte;
BEGIN
  IF _inrange(col,Ø,LOHRES) THEN
    IF _inrange(row1,Ø,LOVRES) OR _inrange(row2,Ø,LOVRES)
    THEN BEGIN
      row1:=_loresclip(row1,Ø,LOVRES);
      row2:=_loresclip(row2,Ø,LOVRES);
      IF row1>row2 THEN BEGIN
        temp:=row1;
        row1:=row2;
        row2:=temp;


      END;
      _wrhostbyte(_AREG,row1);
      _wrhostbyte(_YREG,col);
      _wrhostbyte(_LOCXX,row2);
      _callapl(_AVLIN);
    END;
END;

{end of APLGR/L}
```

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

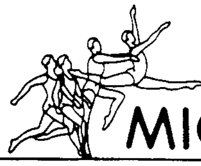
IBM™

MasterFORTH is a state-of-the-art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

CP/M™

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	125
CP/M 2.x (IBM 3740 8")	125
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$60
Graphics (Apple II series)	60
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 FORML Conference	25



MICROMOTION

12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

```
LISTING 4.  APLGR/H.INC
======================

{APLGR/H.INC enables calling hi resolution Apple graphics
 routines from Turbo Pascal programs.
 Requires the PCPI Z80 card (Applicard).
 Copyright 1984 by N.T.Carnevale.
 Permission granted for nonprofit use.}

{contains these routines:
PROCEDURE hirespatch;
  --installs the register-loading routines to be patched
    into motherboard's RAM at $9032-$9058 so ROM hires routines
    can be used
PROCEDURE hiresgr(pagenum:integer; part:partition);
  --invokes hires mode with specified page and partition
PROCEDURE clear_hires_screen(page:integer);
  --clears specified hires page
PROCEDURE hgrselect(scrn:integer);
  --select and clear specified page
PROCEDURE hgrclear;
  --clear hires screen
PROCEDURE hisetcolor(color:hireshues);
  --set color for drawing
PROCEDURE hplot(column,row:integer);
  --plot a point at specified location
PROCEDURE hline(destcol,destrow:integer);
  --draw from present cursor to destination
PROCEDURE setbackground(tint:hireshues);
  --specify color of background
PROCEDURE setcursor(column,row:integer);
  --put cursor at a location

Some of these procedures invoke some of the following ROM
hires graphics routines:
HGR    = 0F3E2H   invoke hires display page 1 with 4 text lines
HGR2   = 0F3D8H   invoke hires display page 2 (full screen)
HCLR   = 0F3F2H   clear current hires page
BKGND  = 0F3F4H   set background color
HCOLOR = 0F6F0H   set color for hires drawing
HPLOT  = 0F457H   position cursor & plot a point
HLINE  = 0F53AH   plot a line
HPOSN  = 0F411H   set cursor at h,v without plotting
                     --call before "draw"
SHPTR  = 0F730H   sets up shape pointers
(Reference:  pp.69-71 in Apple Graphics & Arcade Game Design,
by J.Stanton (The Book Co., Los Angeles) 1982.)

This requires "poking" a few short routines into the 6502's RAM
starting at location 9003H.  The parameters needed by these
routines are "poked" into locations 9000-9002H (bytes destined
for the A and Y registers and location 45H).
}

{------------Other Hires graphics locations------------------
COLRTBL  = 0F6F6H   start of color table
HLCOORD  = 00E0H    two byte horizontal coordinate
VCOORD   = 00E2H    vertical coordinate
CLRMASK  = 00E4H    color masking word from color table


 PAGENUM  = 00E6H    $20 for page 1, $40 for page 2
 SCALE    = 00E7H    scale factor for shape drawing
 SHAPTABL = 00E8H    two byte address of shape table
 ---------------------------------------------------------}

 TYPE
   hireshues=(BLACK1,GREEN,VIOLET,WHITE1,BLACK2,ORANGE,BLUE,WHITE2);

 CONST
   {hires constants}
   HIHRES=280;           {# of pixels across the screen}
   HIVRES=192;           {full screen vertical resolution}
   HIMIXVRES=160;        {mixed mode vert res}
   HIRESPAGE1=$2000;     {start of hires page 1}
   HIRESPAGE2=$4000;     {start of hires page 2}

   {easy ROM routines to call--no parameters needed}
   _HGR=$F3E2;     {invoke hires display page 1 with 4 text lines}
   _HGR2=$F3D8;    {invoke hires display page 2 (full screen)}
   _HCLR=$F3F2;    {clear current hires page}

 {*********************************************************}
 {The following 6502 RAM locations will be patched to hold
  routines that allow access to the ROM graphics functions,
  such as setcolor, hplot, hline etc.}
   _AHCOLOR=$9032; {PURPOSE:  set color for hires drawing
                    SETUP:    poke color into XREG}
   _AHPLOT=$9039;  {PURPOSE:  draw a point at location h,v
                    SETUP:    poke v into AREG, lo byte of h into
                                XREG, hi byte of h into YREG}
   _AHLINE=$9046;  {PURPOSE:  draw a line from initial cursor
                                location to specified point
                    SETUP:    poke v into YREG, lo byte of h into
                                AREG, and hi byte of h into XREG}
   _ABKGND=$9053;  {PURPOSE:  set background color
                    SETUP:    set color before calling, then poke
                                color mask into AREG}
   _AHPOSN=$9059;  {PURPOSE:  put cursor at location h,v
                                without plotting
                    SETUP:    poke v into AREG, lo byte of h
                                into XREG, hi byte of h into YREG
                                --same as for _AHPLOT}


 PROCEDURE hirespatch;
 {installs the routines to be patched into the 6502's RAM.}
 CONST
   {where this patch starts and how long it is}
   CODESTART=$9032;
   CODELENGTH=$34;
   {Borland Pascal's "structured constants" feature is nonstandard.
    So, for that matter, is any hardware-specific code that might
    be generated even with standard syntax!  This just happens to
    be a quick and dirty way to define a table of bytes that
    represents 6502 instructions}
   HIRESTUFF: array [$01..CODELENGTH] of byte=(
     {_AHCOLOR}
     $AE,$02,$90,$20,$F0,$F6,$60,
     {_AHPLOT}
     $AE,$02,$90,$AD,$00,$90,$AC,$01,$90,$20,$57,$F4,$60,
     {_AHLINE}
```

```
      $AE,$02,$90,$AD,$00,$90,$AC,$01,$90,$20,$3A,$F5,$60,
      (_ABKGND)
      $A5,$E4,$20,$F4,$F3,$60,
      (_AHPOSN)
      $AE,$02,$90,$AD,$00,$90,$AC,$01,$90,$20,$11,$F4,$60
      );
VAR source,dest,lnth:integer;
BEGIN
  source:=ADDR(HIRESTUFF[1]);
  dest:=CODESTART;
  lnth:=CODELENGTH;
  _wrhostdata(source,dest,lnth);
END;

(*************************************************************)

PROCEDURE hiresgr(pagenum:integer; part:partition);
(invoke hires mode with specified page and partition)
BEGIN
  _selectpage(pagenum);
  _setpartition(part);
  _wrhostbyte(_HRS,0);
  _wrhostbyte(_GRFX,0);
END;

(*************************************************************)

(Elementary hires graphics procedures)

PROCEDURE clear_hires_screen;
BEGIN
  writeln('dummy routine to clear hires screen');
END;

PROCEDURE hgrselect(scrn:integer);
(select and clear specified page)
BEGIN
  IF scrn=1 THEN _callapl(_HGR)
  ELSE IF scrn=2 THEN _callapl(_HGR2)
  ELSE writeln('There is no page ',scrn);
END;

PROCEDURE hgrclear;
(clear hires screen)
BEGIN
  _callapl(_HCLR);
END;

PROCEDURE hisetcolor(color:hireshues);
(set color for drawing)
BEGIN
  _wrhostbyte(_XREG,ORD(color));
  _callapl(_AHCOLOR);
END;

PROCEDURE hplot(column,row:integer);
(plot a point at specified locus)
BEGIN
  IF _inrange(column,0,HIHRES) THEN
    IF _inrange(row,0,HIVRES) THEN BEGIN
      _wrhostbyte(_AREG,lo(row));

      _wrhostbyte(_XREG,lo(column));
      _wrhostbyte(_YREG,hi(column));
      _callapl(_AHPLOT);
    END;
END;

PROCEDURE hline(destcol,destrow:integer);
(Draw from present cursor to dest.  Uses truly crude clipping!)
BEGIN
  IF _inrange(destcol,0,HIHRES) THEN
    IF _inrange(destrow,0,HIVRES) THEN BEGIN
      _wrhostbyte(_AREG,lo(destcol));
      _wrhostbyte(_XREG,hi(destcol));
      _wrhostbyte(_YREG,lo(destrow));
      _callapl(_AHLINE);
    END;
END;

PROCEDURE setbackground(tint:hireshues);
(specify color of background)
BEGIN
  hisetcolor(tint);
  _callapl(_ABKGND);
END;

PROCEDURE setcursor(column,row:integer);
(put cursor at a specific location)
BEGIN
  _wrhostbyte(_AREG,lo(row));
  _wrhostbyte(_XREG,lo(column));
  _wrhostbyte(_YREG,hi(column));
  _callapl(_AHPOSN);
END;

 (end of APLGR/H)
```

LISTING 5. LORES.A65
====================

```
;LORES.A65
;Purpose:  enable calling Apple low resolution graphics
;routines from CP/M using the PCPI Z80 card (Applicard).
;Copyright 1984 by N.T.Carnevale.
;Permission granted for nonprofit use.
;
;
;Assemble with A65, then use hex codes of the .PRN file to
;generate the code which will be written to the motherboard.
;
;---------------ROM Lores graphics routines-------------------
SETCOL:   .EQU 0F864H   ;set color
PLOT:     .EQU 0F800H   ;plot a point
HLIN:     .EQU 0F819H   ;draw a horizontal line
VLIN:     .EQU 0F828H   ; "      vertical line
;-------------Other Lores graphics locations------------------
H2:       .EQU 002CH    ;rightmost end of horizontal line
V2:       .EQU 002DH    ;bottom end of vertical line
;-------------------------------------------------------------
;
;
          .BLOCK 9000H  ;put this above the driver area
;
                        ;first, loci for temporary storage--
AREG:     .BLOCK 1      ;the scratchpad to which the Appli-
YREG:     .BLOCK 1      ;card writes data destined for
LOCXX:    .BLOCK 1      ;A, Y, and 2C or 2DH
;
CODESTART:              ;beginning of patch area
;
;************************
;ROUTINE:  ASETCOL
;PURPOSE:  set the color used for drawing
;SETUP:    poke "color" byte into AREG before calling
;************************
ASETCOL:
          LDA AREG
          JSR SETCOL
          RTS
;
;************************
;ROUTINE:  APLOT
;PURPOSE:  plot a point at column h, row v
;SETUP:    poke h into YREG, v into AREG before calling
;************************
APLOT:
          LDA AREG
          LDY YREG
          JSR PLOT
          RTS
;
;************************
;ROUTINE:  AHLIN
;PURPOSE:  draw a horizontal line at row v between
;             columns h1 and h2, where h1<h2
;SETUP:    poke v into AREG, h1 into YREG,


;             and h2 into LOCXX before calling
;************************
AHLIN:
          LDA LOCXX
          STA H2
          LDA AREG
          LDY YREG
          JSR HLIN
          RTS
;
;************************
;ROUTINE:  AVLIN
;PURPOSE:  draw a vertical line at column h between
;             rows v1 and v2, where v1<v2
;SETUP:    poke h into YREG, v1 into AREG,
;             and v2 into LOCXX before calling
;************************
AVLIN:
          LDA LOCXX
          STA V2
          LDA AREG
          LDY YREG
          JSR VLIN
          RTS
;
;
FINISH:
;
;how long the whole works is:
LENGTH:   .EQU FINISH-CODESTART
;
          .END
;
;end of LORES.A65
```

LISTING 6. LOWRES.PAS
====================

```
PROGRAM lowres; (lores routine test)

 (Copyright 1984 by N.T.Carnevale.
  Permission granted for nonprofit use.)

(*I PCP.INC)
(*I APLGR/G.INC)
(*I APLGR/L.INC)

VAR
  ans:char;           (for keyboard responses)
  scrn:integer;       (which lo-res graphics screen to use)
  h,v:integer;        (horizontal and vertical coordinates
                       --top left = 0,0)
  tint:loreshues;     (color used for drawing)

PROCEDURE delay;  (about 4 second delay)
VAR i,j:integer;
BEGIN
  FOR i:=0 TO 500 DO
    FOR j:=1 TO 500 DO;
END;

BEGIN
  lorespatch;   (install the register-loading routines)
  writeln('Low-resolution graphics exerciser');
  write('Press return to clear and fill screen 1: ');
  readln(ans);  (can't use lores screen 2)
  loresgr(1,FULLSCREEN);      (display lores screen 2)
  clear_lores_screen;         (clear it)
  tint:=BLACK;                (start with black)
  FOR v:=0 TO LOVRES-1 DO BEGIN
    setcolor(tint);           (use specified color)
    hlin(v,0,LOHRES-1);       (draw horiz line across screen)
    IF tint=WHITE THEN tint:=BLACK
    ELSE tint:=SUCC(tint);  (next color to use)
  END;
  delay;
  FOR h:=0 TO LOHRES-1 DO BEGIN
    setcolor(tint);
    vlin(h,0,LOVRES-1);  (draw vert line down screen)
    IF tint=WHITE THEN tint:=BLACK
    ELSE tint:=SUCC(tint);
  END;
  delay;
  textscreen(1);  (return to the text display before exit)
END.  (end of PROGRAM lowres)
```

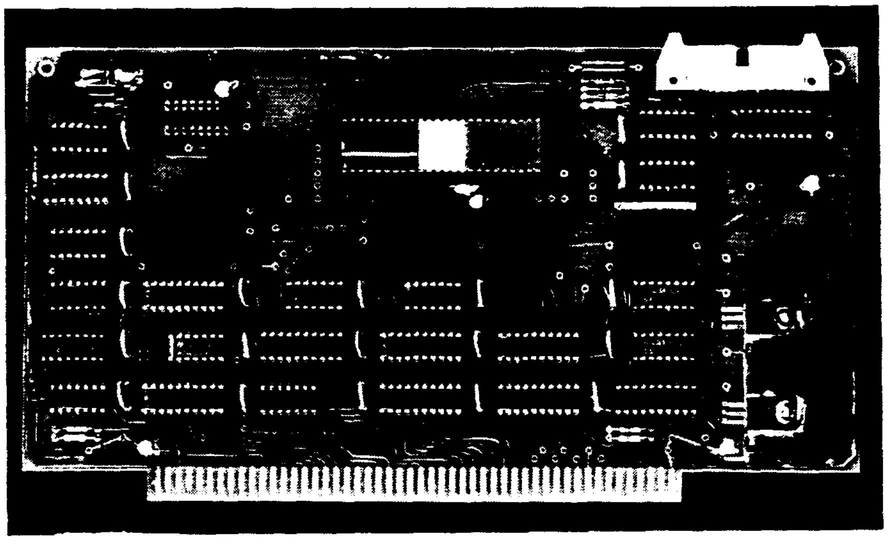
LISTING 7. SINES.PAS

```
PROGRAM sines; {demonstrates plot of sine function}

{Copyright 1984 by N.T.Carnevale.
 Permission granted for nonprofit use.}

CONST
  GRAFSCREEN=2; {use only hires screen 2 with PCPI v.2 CP/M}
  BELL=7;

TYPE
  {these are used to map the "real world" onto the display}
  realdata=RECORD
      x,y:real; {x&y world coordinates, that is, "real data"}
    END;
  screendata=RECORD
      x,y:integer; {x&y display coordinates}
    END;
  realscalefactors=RECORD
      mx,my,bx,by:real; {used to map world into display}
    END;

{$I PCP.INC}
{$I APLGR/G.INC}
{$I APLGR/H.INC}

VAR
  ans:char;
  frameloc,framesize:screendata;
  lowerleft,upperright:realdata;
  frame:realscalefactors;
  hue:hireshues;

{$I PLOTTER.INC}
{PLOTTER.INC contains the following:
 PROCEDURE setframe--sets up the coefficients ("magnifications"
   and "shifts") that are used to transform or map "real data"
   to the display.  Parameters are:
     lowerleft,upperright:realdata--the opposite corners of
       a rectangular area that contains the range of "real
       data" to be plotted ("corners of the real world").
     frameloc:screendata--left upper corner of area on the
       screen where the data is to go (where to put the
       picture).
     framesize:screendata--dimensions of the area on the
       screen where the data is to go (how big to make the
       picture).
     VAR frame:realscalefactors--this record contains the
       coefficients (calculated by setframe) that will be
       used by other procedures to map "real data" to the
       display.

 PROCEDURE plot--draws a point on the hires page using
   specified scale factors.  Parameters are:
     point:realdata--x,y coordinates of the point in the
       "real world."
     frame:scalefactors--the coefficients used to map


        the point onto the display.

 PROCEDURE plotline--starting from present cursor location,
   draws a line to the point on the screen that corresponds
   to a specified endpoint in the "real world," using
   specified scale factors.  Parameters are:
     endpoint:realdata--x,y coordinates of the end of the
       line in the "real world."
     frame:scalefactors--the coefficients used to map the
       point onto the display.
}


PROCEDURE genplot;
{generate and plot one cycle of a sine wave}
CONST PI=3.1415926;
VAR
  i:integer;
  point:realdata;
  dx:real;
BEGIN
  point.x:=0.0;
  dx:=0.02*pi;
  point.y:=sin(point.x);
  plot(point,frame);            {plot the first point}
  FOR i:=1 TO 100 DO BEGIN
    point.x:=point.x+dx;        {calculate the next point}
    point.y:=sin(point.x);
    plotline(point,frame);      {and draw a line to it}
  END;
END;

BEGIN
  hirespatch;  {install register-loading routines}
  writeln('Sine plotter');
  write('First, screen ',GRAFSCREEN,
        ' will be cleared--press return to proceed');
  readln(ans);
  hgrselect(GRAFSCREEN);              {select screen to use}
  hiresgr(GRAFSCREEN,FULLSCREEN);     {  and clear it}
  textscreen(1);                      {restore text display}
  writeln;
  writeln('Press return to plot sine function.');
  writeln('After the bell rings, press return again');
  writeln('  to leave graphics mode.');
  readln(ans);
  {specify limits of "real world" data}
  lowerleft.x:=0.0;     lowerleft.y:=-1.0;
  upperright.x:=2*PI;   upperright.y:=1.0;
  {set up size of display area}
  framesize.x:=HIHRES - 90;  framesize.y:=HIVRES DIV 2;
  {put first frame at top left-hand corner of display}
  frameloc.x:=0;  frameloc.y:=0;
  hiresgr(GRAFSCREEN,FULLSCREEN);  {go back to graphics}
  hue:=BLACK1;                     {first "color" to use}
  REPEAT
    hue:=succ(hue);              {advance to the next color}
    hisetcolor(hue);
    frameloc.y:=frameloc.y+10;  {  and shift the frame}
    frameloc.x:=frameloc.x+12;
    setframe(lowerleft,upperright,frameloc,framesize,frame);
    genplot;                     {plot one sine wave}
  UNTIL hue=WHITE2;
  writeln(chr(BELL));  {ring the bell}
  readln(ans);         {wait until return key is pressed}
  textscreen(1);       {restore text display before exit!}
END.  {end of PROGRAM sines}
```

LISTING 8. PLOTTER.INC

```
{PLOTTER.INC--what it takes to set up a frame
 and plot data into it.  Written for floating point data.
 Copyright 1984 by N.T.Carnevale.
 Permission granted for nonprofit use.}

(*This file must be included after PCP, APLGR/G and APLGR/H.
 The following types (and corresponding variables) must be
 defined in the main file before PLOTTER is included:

TYPE
  realdata=RECORD
      x,y:real; {x&y world coordinates, that is, "real data"}
    END;
  screendata=RECORD
      x,y:integer; {x&y display coordinates}
    END;
  realscalefactors=RECORD
      mx,my,bx,by:real; {used to map world into display}
    END;

PLOTTER contains the following procedures:
  setframe--sets up the coefficients ("magnifications"
    and "shifts") that are used to transform or map "real data"
    to the display.  Parameters are:
      lowerleft,upperright:realdata--the opposite corners of
        a rectangular area that contains the range of "real
        data" to be plotted ("corners of the real world").
      frameloc:screendata--left upper corner of area on the
        screen where the data is to go (where to put the
        picture).
      framesize:screendata--dimensions of the area on the
        screen where the data is to go (how big to make the
        picture).
      VAR frame:realscalefactors--this record contains the
        coefficients (calculated by setframe) that will be
        used by other procedures to map "real data" to the
        display.

  plot--draws a point on the hires page using
    specified scale factors.  Parameters are:
      point:realdata--x,y coordinates of the point in the
        "real world."
      frame:scalefactors--the coefficients used to map
        the point onto the display.

  plotline--starting from present cursor location,
    draws a line to the point on the screen that corresponds
    to a specified endpoint in the "real world," using
    specified scale factors.  Parameters are:
      endpoint:realdata--x,y coordinates of the end of the
        line in the "real world."
      frame:scalefactors--the coefficients used to map the
        point onto the display.

 Procedures not in this file that would be nice to have:
   --"moveto" a specific location without drawing a point


    (unlike plot, which moves the cursor to a point and
    draws a point there)
  --"relative" cursor moves (plot and plotline put the
    cursor at a specific or "absolute" location on the
    display
  --true clipping, so that, if one or both endpoints of a
    line lies outside the defined frame, only the portion
    of it that is within the frame will be drawn
  --a circle drawing procedure
*)


{sets up the scale factors used by the plot routines}
PROCEDURE setframe
  (lowerleft,
   upperright:realdata;  {data limits}
   frameloc:screendata;  {left upper corner of display area}
   framesize:screendata; {dimensions of display area}
   VAR frame:realscalefactors {calculated by setframe}
  );
BEGIN
  WITH frame DO BEGIN
    mx:=(framesize.x-1)/(upperright.x-lowerleft.x);
    bx:=frameloc.x-mx*lowerleft.x;
    my:=(framesize.y-1)/(lowerleft.y-upperright.y);
      {note:  Apple's screen is "upside-down"}
    by:=frameloc.y-my*upperright.y;
  END;
END;

{put cursor and plot a point at a specified location}
PROCEDURE plot(point:realdata; frame:realscalefactors);
VAR h,v:integer; {actual display coords}
BEGIN
  WITH frame DO BEGIN
    h:=round(mx*point.x+bx);
    v:=round(my*point.y+by);
    hplot(h,v);
  END;
END;

{draw a line from present cursor location to specified endpoint}
PROCEDURE plotline(endpoint:realdata; frame:realscalefactors);
VAR h,v:integer; {actual display coords}
BEGIN
  WITH frame DO BEGIN
    h:=round(mx*endpoint.x+bx);
    v:=round(my*endpoint.y+by);
    hline(h,v);
  END;
END;

{end of PLOTTER.INC}
LISTING 9.  DUMPSCRN.PAS
========================


PROGRAM dumpscrn; (dumps a hires screen to the printer.
Assumes the printer card is software-compatible with
the GRAPPLER.
Copyright 1984 by N.T.Carnevale.
Permission granted for nonprofit use.)

CONST GRAFSCREEN=2; (use only hires screen 2 with PCPI v.2 CP/M)

($I PCP.INC)
($I APLGR/G.INC)
($I APLGR/H.INC)

TYPE string70=string[70];

VAR
  size:(single,double);      (specifies 1:1 or 2:1 screen dump)
  ans:char;
  controlstring:string[4];  (used for printer card commands)
  i,numlf,copynum:integer;
  scrn:integer;

PROCEDURE delay;
VAR i,j:integer;
BEGIN
  FOR i:=0 TO 500 DO
    FOR j:=1 TO 500 DO;
END;

FUNCTION promptans(prompt:string70):char;
(display the prompt on the console,
 get a single uppercase response from the keyboard)
VAR ans:char;
BEGIN
  write(prompt);
  readln(ans);
  promptans:=upcase(ans);
END;

PROCEDURE dumpit;  (tell Grappler to do the screen dump)
VAR i:integer;
BEGIN
  ans:=promptans('Adjust top edge of paper, then press RETURN');
  FOR i:=1 TO numlf DO writeln(lst);  (blank lines for centering)
  writeln(lst,chr(0),chr(25),controlstring); (null is for safety's sake)
  FOR i:=1 TO numlf DO writeln(lst);  (more blank lines after dump)
END;

BEGIN
  textscreen(1);  (insure text display at start of program)
  hirespatch;     (install register-loading routines)
  REPEAT
    write('Dumping screen ',GRAFSCREEN,'--');
    scrn:=GRAFSCREEN;
    hiresgr(scrn,FULLSCREEN); (shows the screen without clearing it)
    delay;
    textscreen(1);  (return to text display)
    ans:=promptans('P)roceed or Q)uit? ');
  UNTIL ans IN ['P','Q'];
  IF ans='P' THEN BEGIN
    ans:=promptans('D)ouble or S)tandard size? ');
    IF ans='D' THEN BEGIN
      numlf:=9;
      controlstring:='GDR2';  (command for magnified screen dump)
    END ELSE BEGIN
      numlf:=19;
      controlstring:='GR2';  (standard screen dump)
    END;
    writeln;
    write('Number of copies to make: ');
    readln(copynum);
    FOR i:=1 TO copynum DO dumpit;  (do the screen dump)
    writeln('Check top edge of paper and reset printer');
  END;
END.  (of PROGRAM dumpscrn)
```

```
LISTING 10.  SAVSCRN.PAS
========================


PROGRAM savscrn; (saves a hi res screen to disk.
Copyright 1984 by N.T.Carnevale.
Permission granted for nonprofit use.)

CONST GRAFSCREEN=2; (use only hires screen 2 with PCPI v.2 CP/M)

($I PCP.INC)
($I APLGR/G.INC)
($I APLGR/H.INC)

TYPE
  string70=string[70];
  byte=char;
  screenline=array [1.._BPL] of byte;  (_BPL is defined in APLGR/G)
  figfile=FILE of screenline;

VAR
  ans:char;
  scrn:integer;

PROCEDURE delay;
VAR i,j:integer;
BEGIN
  FOR i:=0 TO 500 DO
    FOR j:=1 TO 500 DO;
END;

FUNCTION promptans(prompt:string70):char;
(display prompt on monitor,
 get uppercase single character from keyboard)
VAR ans:char;
```



INTELLICOMP
Introduces
Inexpensive
S-100 68008
CPU Board

The card pictured above is $65 for the bare board, $210 for the kit, or $265 assembled and tested. It uses only standard parts. A sample BIOS for CP/M 68K is available on disk for $20. The board works fine with Digital Research Computers 64K RAM boards and semi disks. A detailed description of the board appears in Issue 16 of The Computer Journal.

For additional information call Intellicomp, Inc. at (614) 846-0216 (evenings best time) or write to:
Bruce Posey
Intellicomp, Inc.
292 Lambourne Ave., Worthington, OH 43085

```
BEGIN
  write(prompt);
  readln(ans);
  promptans:=upcase(ans);
END;

FUNCTION rowstart(row,page:integer):integer;
(calculate the starting address corresponding a line or row number)
VAR pagebase:integer;
BEGIN
  IF page=1 THEN pagebase:=HIRESPAGE1 ELSE pagebase:=HIRESPAGE2;
  rowstart:=pagebase + $28*(row SHR 6) + (((row SHR 3) MOD 8) SHL 7)
           + ((row MOD 8) SHL 10);
END;

PROCEDURE doit; (simple read and save a screen to disk)
VAR
  filnam:string[12];
  f:figfile;
  linenum:integer;
  temp:screenline; (temporary array to hold a line from the screen)
BEGIN
  write('File to receive picture: ');
  readln(filnam);
  assign(f,filnam);

  rewrite(f);
  FOR linenum:=0 TO (HIVRES-1) DO BEGIN
    (read _BPL bytes from the display memory, starting at
     the address that corresponds to the line number,
     into the array temp[])
    _rdhostdata(rowstart(linenum,GRAFSCREEN),addr(temp[1]),_BPL);
    (save the array of bytes in the file)
    write(f,temp);
  END;
  close(f);
END;

BEGIN
  textscreen(1);  (guarantee text display at program start)
  hirespatch;     (install register-loading routines)
  REPEAT
    write('Saving screen ',GRAFSCREEN,'--');
    scrn:=GRAFSCREEN;
    hiresgr(scrn,FULLSCREEN); (shows the screen without clearing it)
    delay;
    textscreen(1);  (return to text display)
    ans:=promptans('P)roceed or Q)uit? ');
  UNTIL ans IN ['P','Q'];
  IF ans='P' THEN doit;
END.  (end of PROGRAM savscrn)
```

FREE SOFTWARE

RENT FROM THE PUBLIC DOMAIN!

User Group Software isn't copyrighted, so there are no fees to pay! 1000's of CP/M and IBM software programs in .COM and source code to copy yourself! Games, business, utilities! All FREE!

CP/M USERS GROUP LIBRARY

Volumes 1-92, 46 disks rental—$45

SIG/M USERS GROUP LIBRARY

Volumes 1-90, 46 disks rental—$45

Volumes 91-176, 44 disks rental—$50

SPECIAL! Rent all SIG/M volumes for $90

K.U.G. (Charlottesville) 25 Volumes—$25

IBM PC-SIG (PC-DOS) LIBRARY

Volumes 1-200, 5¼" disks $200

174 FORMATS AVAILABLE! SPECIFY.

Public Domain User Group Catalog Disk $5 pp. (CP/M only) (payment in advance, please). Rental is for 7 days after receipt, 3 days grace to return. Use credit card, no disk deposit. Shipping, handling & insurance—$7.50 per library.

(619) 914-0925 information,(9-5)

(619) 727-1015 anytime order machine

Have your credit card ready! VISA, MasterCard, Am. Exp.

Public Domain Software Center

1533 Avohill Dr.

Vista, CA 92083

```
LISTING 11.  GETSCRN.PAS
========================

PROGRAM getscrn; (fills a hi res display with data from
a file that was saved to disk by savscrn.
Copyright 1984 by N.T.Carnevale.
Permission granted for nonprofit use.)

CONST GRAFSCREEN=2; (use only hires screen 2 with PCPI v.2 CP/M)

($I PCP.INC)
($I APLGR/G.INC)
($I APLGR/H.INC)

TYPE
  string70=string[70];
  byte=char;
  screenline=array [1.._BPL] of byte;
  figfile=FILE of screenline;

VAR
  ans:char;
  scrn:integer;

PROCEDURE delay;
VAR i,j:integer;
BEGIN
  FOR i:=0 TO 500 DO
    FOR j:=1 TO 500 DO;
END;

FUNCTION promptans(prompt:string70):char;
VAR ans:char;
BEGIN
  write(prompt);
  readln(ans);
  promptans:=upcase(ans);
END;

FUNCTION rowstart(row,page:integer):integer;
(calculate the starting address corresponding a line or row number)
VAR pagebase:integer;
BEGIN
  IF page=1 THEN pagebase:=HIRESPAGE1 ELSE pagebase:=HIRESPAGE2;
  rowstart:=pagebase + $28*(row SHR 6) + (((row SHR 3) MOD 8) SHL 7)
           + ((row MOD 8) SHL 10);
END;

PROCEDURE doit; (simple read a screen from disk
  & write to specified screen)
VAR
  filnam:string[12];
  f:figfile;
  linenum:integer;
  temp:screenline; (temporary array to hold a line from the screen)
BEGIN
  write('File to read: ');
  readln(filnam);
  assign(f,filnam);

  reset(f);
  hiresgr(scrn,FULLSCREEN); (shows the screen without clearing it)
  FOR linenum:=0 TO (HIVRES-1) DO BEGIN
    (read _BPL bytes from the file into temporary storage)
    read(f,temp);
    (write the bytes to the display memory, starting at
     the address that corresponds to the line number)
    _wrhostdata(addr(temp[1]),rowstart(linenum,GRAFSCREEN),_BPL);
  END;
  close(f);
END;

BEGIN
  textscreen(1);        (start in text display)
  hirespatch;           (install register-loading routines)
  scrn:=GRAFSCREEN;
  REPEAT
    hiresgr(scrn,FULLSCREEN); (shows the screen without clearing it)
    delay;
    textscreen(1);     (return to the text display)
    ans:=promptans('Replace that with data from a file? ');
    IF ans='Y' THEN BEGIN
      doit;
      delay;
      textscreen(1);
      ans:=promptans('Do it again? ');
    END;
  UNTIL ans<>'Y';
END.  (end of PROGRAM getscrn)
```

For those who would rather not key in these listings, they are available on disk in Apple CP/M format for $10.00. Send payment (VISA or MasterCard accepted) with your name and address to **The Computer Journal,** 190 Sullivan Crossroad, Columbia Falls, MT 59912.

■

Z SETS YOU FREE!

Free to create computer environments right for you . . . free to automate repetitive tasks . . . free to increase your productivity. **Z-System**, the high-performance 8-bit operating system that flies! Optimized assembly language code — full software development system with linkable libraries of often needed subroutines —relocating (ROM and RAM) macro assembler, linker, librarian, cross-reference table generator, debuggers, translators, disassembler — ready to free you!

TERM III New generation communications package provides levels of flexibility, functionality, performance not available until now. Replaces BYE and XMODEM . . . master/server local area network capability . . . public or private bulletin board and electronic message handling are integral features . . . auto-dial/answer, menu install . . . XMODEM (CRC/Checksum), MODEM7 Batch, Kermit, CIS, and XON/XOFF protocols . . . 100-page manual **$99.00**

Z-MSG Rolls Royce of message handling systems . . . mates with TERM III or BYE for most advanced overall electronic mail/file transfer capabilities . . . menu installed . . . extreme configurability . . . many levels of access and security . . . word, phrase editor, field search . . . complete message manipulation and database maintenance **$99.95**

DISCAT Elegant, menu and command-line driven file and disk catalog manager. Generates and controls multiple master catalogs, working catalog used for update quickness. Nine flexible modules easily altered by user for custom requirements. Works with Z shells (VMENU, VFILER, MENU), aliases, and multiple commands per line **$39.99**

ZCPR3: The Manual Bound, 350 pages, typeset book describes features of ZCPR3 command processor, how it works, how to install, and detailed command usage. Bible to understand Z-System **$19.95**

ZCPR3 and I/OPS Loose-leaf book, 50 pages, 8-1/2" by 11", describes ins-and-outs of input/output processing using Z-System. Shows how to modify your BIOS to include I/O redirection . . . complements **The Manual** **$9.95**

More missing links found — Z Application Progams! Fly with eagles! Our programs promote high performance through flexibility! Productivity results from dynamically changeable work environments, matching operator to tasks and machines.

Above programs require 48K-byte memory, ZCPR3, Z-Com, or Z-System, and Z80/NSC800/HD64180-based computer. Shipping from stock. State desired disk format, plus two acceptable alternatives. As payment, we accept Visa, Mastercard, personal checks, money orders, and purchase orders from established companies. We also ship UPS COD.

Call or write to place order or to obtain literature.

Echelon, Inc. 101 First Street • Suite 427 • Los Altos, CA 94022 • 415/948-3820

Editor
(continued from page 1)

mapped I/O transfer capability; two-channel, full duplex asnychronous serial communications with programmable baud rate generator and modem control signals; 12-source vectored programmable interrupt controller; two channel 16-bit programmable reload timer; clocked serial I/O port; programmable dynamic RAM refresh and timing; wait state generator for slow memory and I/O devices; and dual-bus interface compatibility with 68xx and 80xx families. It is currently available in a 6MHz version and an 8 MHz version is promised. The higher clock speed plus instructions which operate in fewer clock cycles than the Z-80 and a hardware 8-bit multiply with 16-bit results have produced some impressive benchmark results. We are currently working on a construction series for a HD64180 Single Board Computer, and are looking for articles on programming it. Micro Mint is announcing a very reasonable priced SBC, Echlon has their enhanced Z-System and an assembler ready, and Byte will be breaking a feature article. Keep your eye on this chip, it's hot and we will be carrying more information when available.

WD65802 CPU for Apple Upgrades

Western Design is finally delivering its WD65802 which is a plug-in replacement for the 6502, and provides 16-bit accumulator, ALU, X and Y index registers, and stack pointer, and can run existing 6502 and 65C02 code. The S-C Macro, ORCA/M, and Merlin Pro assemblers are ready for the 65802 (they'll also handle the 65816), and it is rumored that Apple will bring out a new model II based on one of these chips.

This chip allows Apple II users to upgrade to 16-bit registers for about $50 and still continue to use their existing hardware and software. Now if Apple would only release 800K high performance drives, the model II's would really hum.

Reasonably Priced Software

Many of the new programs are available for $100 or less, and the unrealistically high software prices may be a thing of the past. Some companies try to defend their high prices by talking about the man-years of development time and the high promotional and support costs, while they drive BMW's and throw $50,000 parties. Eventually the marketplace will decide the proper price, and it is interesting to note that some of the best support comes from the lower priced software companies where you can talk to the original programmer and people care about their product and their customers. These companies have lower advertising budgets (no ten million dollar promos) which will raise hell with the slick four-color magazines which depend on these budgets. We should help support reasonably priced software by passing the word and telling others about programs that work and provide good support at low cost.

Hacker's Haven

If my workshop was in the attic I could call it *Art's Attic*, but it's in the basement and if my name was Dave I'd call it *David's Dungeon.* Since I can't think of anything better, I'll call it *Hacker's Haven* for now and hope that you will all understand that I'm using the original definition of hacker and NOT someone who breaks into systems. In fact, right now I don't even have a modem!

I have added a Vista A800 8" DSDD drive interface to my Apple II+, it sure is nice to have 1.2 Megabytes of high speed disk available. It's not a hard disk, but I have removable media and I picked up the interface for $50 and used it with one of my existing drives. Under CP/M it can write either standard IBM format single density or double density, and using single density I can read and write to the disk with both the Apple and S-100 CP/M systems, which will make it easy to transfer files between the systems. I'll also run some benchmarks on this drive against the Apple drives. I received patch disks for DOS 3.3, CP/M 2.2, and Apple Pascal. If anyone has experience with this interface I'd like to hear from them, especially if they have any information on patching ProDOS to run on this interface.

The operating system is the most important part of a system, and CP/M-80 does have its shortcomings. I have acquired the ZCPM3 core and utilities from Echelon and will be bring it up this month (I hope). From reading the literature it will do a lot of the things that I missed in CP/M-80 2.2, and I'll keep you posted as I learn more about it.

If I add the 65802 to my Apple with the 8" drive, and get a HD64180 running with ZCPR3, I'll be content for some time while I learn how to use their new features (and still be able to run all my existing software). But this magazine isn't just about what I am doing—it's also about what you're doing. Drop us a line and tell the readers what you are working on, and what you'd like to do. If you have any questions or problems, let the other hackers help you find an answer. ■

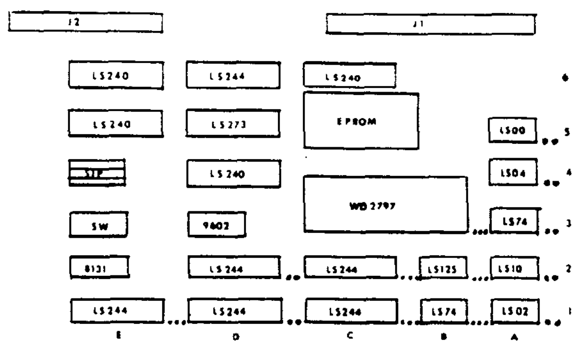
NEW FREE CATALOG

CP/M MSDOS **AFFORDABLE ENGINEERING SOFTWARE** TRSDOS PCDOS

- **LOCIPRO** Root Locus — **$69.95**
- **ACTFIL** Active Filter Design/Analysis — **$69.95**
- **STAP** Static Thermal Analysis — **$69.95**
- **MATRIX MAGIC** Matrix Manipulation — **$69.95**
- **RIGHTWRITER** Proofreader & Writing Style Analyzer — **$74.95**
- **ACNAP2** AC Circuit Analysis — **$69.95**
- **DCNAP** DC Circuit Analysis — **$69.95**
- **SPP** Signal/System Analysis — **$69.95**
- **PLOTPRO** Scientific Graph Printing — **$69.95**
- **PCPLOT2** High Resolution Graphics — **$69.95**

BV Engineering
Professional Software
2200 Business Way Suite 207 • Riverside CA 92501 • **(714) 781-0252**
VISA MasterCard

Floppy Disk Controller
(continued from page 26)



Board Layout

```
mwr_mini:
        move.l  sp,spsv
        bsr     rwmini
        ori     #writc,d4
        move    d4,d1
        bsr     mwr
        and.b   #$fc,d1
        beq     wr_mn2
        bset    #7,erflag
        bra     merr
wr_mn2: rts

rd_mini:
        move.l  sp,spsv
        bsr     rwmini
        ori     #readc,d4
        move    d4,d1
        bsr     mrd
        and.b   #$fc,d1
        beq     rd_mn2
        bset    #6,erflag
        bra     merr
rd_mn2: rts

rwmini:
        bsr     parmm
        move    d6,d4           * track to d4.w
        and     #1,d4           * even or odd
        lsl     #1,d4           * position bit for 880
        lsr     #1,d6           * cylinder to seek
        bsr     mdriv
        beq     rwmini1
        bsr     mrdid
rwmini1: bsr    mseek
        move.l  a4,a0
        move    #511,d2         * byte count
        addq    #1,d7           * sekhst starts at 0
        move.b  d7,sector(a5)
        rts

*---------
* can't use clr.b select(a5) or sf select(a5)
* because both do a read before write which
* sets the WAIT ff
*---------
merr:   andi.b  #$f0,dkctrl     * deselect drive
        move.b  dkctrl,select(a5)
        st      prev_51         * prev := -1
        move.l  spsv,sp         * restore stack ptr to starting
value
        rts

mseek:
        cmp.b   #79,d6          * don't damage drives
        bgt     mbad_trk
        cmp.b   track(a5),d6    * right track ?
        beq     mseek1
        move.b  d6,ddata(a5)    * tell 2797 new position
        move.b  #skcmd,fcmd(a5) * go find it
        bsr     mbusy           * wait for status
        and.b   #$98,d1
        bne     mseek_err       * if error
mseek1: rts

mbad_trk:
        bset    #2,erflag
        bra     merr

mseek_err:
        bset    #5,erflag       *    set seek error message
        bra     merr            *    and error

mrdid2:
        lea     ctrak,a0        * read 6 byte address id into
cctrak
        moveq   #5,d2
        move.b  #readid,d1      * read addr id command
        bsr     mrd             * read it
        and.b   #$9f,d1         * if no errors
        rts
```

```
mrdid:
        bsr     mrdid2
        beq     mrdid1
        move.b  #stepout,fcmd(a5)
        bsr     mbusy
        bsr     mrdid2
        bne     mid_err
mrdid1: move.b  ctrak,track(a5) * set track reg in 2797
        rts

mid_err:
        bset    #4,erflag       * else set read id message
        bra     merr            * and error

mrd:
        bsr     mdisint         * do set up and disable
interrupts
mrd1:   move.b  (a2),d1         * any read from EWAIT ok to stop
cpu
        move.b  (a1),(a0)+      * send a byte
        dbra    d2,mrd1         * loop for count
        bra     mnbint          * common exit code and enabe
interrupts

mwr:
        bsr     mdisint         * do set up and disable
interrupts
mwr1:   move.b  (a2),d1         * any read from EWAIT ok to stop
cpu
        move.b  (a0)+,(a1)      * get a byte
        dbra    d2,mwr1         * loop for count

mnbint:
*       move d0,sr              * enable interrupts

mbusy:
        moveq   #10,d1          * status not available for a
short bit
mbusy1: dbra    d1,mbusy1
mbusy2: move.b  status(a5),d1   * get status
        btst    #0,d1
        bne     mbusy2          * loop while the busy bit set
        rts

mdisint:
*       move    sr,d0           * save status
*       ori     #$0700,sr       * disable interrupts
        move.b  d1,fcmd(a5)     * send command to 2797
        lea     ddata(a5),a1    * aim a1 at DATA
        lea     ewait(a5),a2    * aim a2 at EWAIT
        rts

*---------


* LS273 meanings are given below
*
* motor_on      1       motor_off       0       bit7
* prom_off      1       prom_on         0       bit6
* 5 inch        0       8 inch          1       bit5
* double_dens   0       single_dens     1       bit4
*
* 5 inch_sel_1          bit3
* 5 inch_sel_0          bit2
* 8 inch_sel_1          bit1
* 8 inch_sel_0          bit0
*---------

delay1  equ     80
delay2  equ     10000

mdriv:
        move.b  #$c0,d1
        bset    d5,d1           * d5.b has unit number
        move.b  d1,dkctrl
        move.b  d1,select(a5)   * select drive
        move    #delay1,d2
mdriv2: bsr     mbusy
        and.b   #$80,d1
        beq     mdriv3
        move    d2,-(sp)
        move    #delay2,d2
mdriv1: dbra    d2,mdriv1
        move    (sp)+,d2
        dbra    d2,mdriv2
        bset    #3,erflag       * set err bit
        bra     merr
mdriv3: cmp.b   prev_51,d5
        beq     mdriv4          * eq means can skip rdid:
        move.b  d5,prev_51      * set prev to current disk
mdriv4: rts
```

New Products

Root Locus Analysis

BV Engineering has added LOCIPRO to its line of engineering software. LOCIPRO provides control system and electronic engineers a simple means to quickly determine closed loop system stability from open loop transfer functions. LOCIPRO is a stand-alone computer program which quickly solves the locus of roots for systems up to 26th order and ten loop elements. Output data can be vectored to a line printer or data files. All program inputs are free format and menu driven. Output files are compatible with other BVE products adding transient analysis and high resolution graphics. LOCIPRO is available for $72.95 under PC/MSDOS, CP/M-80, and TRSDOS in 121 different disk formats. For additional information and a free catalog call or write BV Engineering, 2200 Business Way ‡207, Riverside, CA 92501 , phone (714) 781-0252.

C Development System

AGS has announced Smart/C, a fully integrated, pre-compilation development environment for the C language. Smart/C, which incorporates many elements from artificial intelligence research, is a productivity enhancing software tool. It allows the user to create, edit, test, and debug C programs, all before any compilation step.

They claim that Smart/C supplys a complete set of tools for eliminating all syntax errors and most logic errors before any compilation is done, and that its interpretation is not incremental compilation, but true interpretation. Smart/C consists of the Environment, which provides syntax directed editing and interpretation of C source; and the Migrator, which allows C programs not created with Smart/C to take advantage of the editing and debugging capabilities of the environment. There is a Verbose capability available which prompts the novice user as to the next appropiate syntax element when the program is being created.

Smart/C runs on several machines, ranging from the IBM PC, to the AT&T 3B series, to the VAX 11/780 tm and runs on a variety of operating systems, including MS-DOS, UNIX tm System V release 2, XENIX and Berkeley BSD 4.2. Pricing ranges from $500 for the IBM PC under MS-DOS to $10,000 for the DEC VAX 11/780 running Ultrix. For more information, contact AGS Computers, Inc., Advanced Products Division, 1139 Spruce Drive, Mountainside, New Jersey 07092.

Assembler and Debugger for Hitachi HD64180 Super Chip

Echelon, Inc has announced two programs to support Hitachi's HD64180 high-integration 8-bit microprocessor chip: ZAS, a machine code relocating macro assembler and ZDM, a debugger and monitor.

ZAS produces Intel compatible HEX as well as Microsoft REL files. It's compatible with Digital Research's ASM, MAC and RMAC, with Microsoft's MACRO-80, and Xitan's TDL assemblers.

Considered a universal assembler, ZAS converts Hitachi HD64180 instructions (IN0, MLT, OTDM, OTIM, OUT0, SLP, TST, and TSTIO) into machine operation codes. ZAS also handles the complete Zilog Z80 instruction set. Nestable conditionals and full expression handling, relocation by absolute, code, common, and data criteria, and complete macro expansion and library insert capabilities. ZAS creates 8-bit standard SYM tables used by DSD, SID and ZSID debuggers. Fifty pseudo-ops are provided making for effective and efficient Assembly Language code writing, for both ROM and RAM usage.

ZAS comes complete with Microsoft compatible REL file linking loader (ZLINK), Intel-to-Zilog mnemonic translator (ZCON), relative code file librarian (ZLIB), and source listing symbol-to-line cross reference generator (ZREF).

ZDM, the dynamic debugger and monitor, provides quick assembly language code development and maintenance. Twenty-one commands permit complete object code debugging and hardware port exercising. Beyond-normal commands: string search in hex and ASCII, send and receive I/O port bytes, verify if two blocks of memory are identical, show only branch statements, send all screen output to printer, examine prime and alternate registers. Plus math in hex, enable/disable interrupts and regular functions such as fill, go, input, move, display, read, set, trace, and untrace complete the command set.

ZAS sells for $69.00 including a 70 page loose-leaf manual, and ZDM sells for $50.00 including a 20 page manual. Both programs work with CP/M, MP/M, and Z and run on Z80, NSC800 and HD64180 based microcomputers. For OEM's and VAR's, Echlon offers ZAS and ADM with economical pricing and multi-year, volume buying arrangements. Echlon also produces the high-performance 8-bit operating system called Z-System (combined ZCPR3 and ZRDOS) which is fully upward compatible with CP/M plus utilities and subroutines.

Contact Echlon, Inc., at 101 First Street, Los Altos, CA 94022, or phone (415) 948-3830.

Data Acquisition Network

Strawberry Tree Computers now has a network that links up to 64 computers running its Analog Connection PC data acquisition system, to a master computer up to 3500 feet away. The DNA network manufactured by Network Development Corporation transfers data at up to 1 million bits per second over a 4-wire cable between IBM PC, XT, AT or compatible computers. Remote computers can log data on the master computer or print on a shared printer while monitoring and controlling temperature, pressure and other inputs. Any computer, including the master, can run MS DOS programs such as Lotus 1-2-3 or dBase III for data analysis and process supervision. The slave can operate without disk drives in dirty environments. Operation of the network is not impared by failure of any remote computer.

The prices are $695 for the Master Interface Card and Software, $395 for the Slave Interface Card, with the Analog Connection PC data acquisition interface starting at $690 for eight channels. For more information contact Walter Maclay at Strawberry Tree

Computers, 949 Cascade Drive, Sunnyvale, CA 94087, phone (408) 736-3083.

ProDOS Assembler for Apple IIe and IIc

The S-C Macro Assembler Version 2.0 is now available in both DOS 3.3 and ProDOS versions. They both support the full instruction set and addressing modes of the 6502, 65C02, 65802, and 65816 processors, as well as Steve Wozniak's SWEET-16 pseudo-processor.

The ProDOSS version includes all the familiar S-C Macro features, and adds some new ones. One new directive implements a form of text compression for programmer messages that squeezes text to an average of less than five bits per character. It also supports a new "blocked include" directive, which allows even larger programs to be assembled than was possible with the DOS-based version.

The S-C Macro Assembler is NOT copy protected, and is compatible with the Apple II, II Plus, IIe, and IIc. Only 64K RAM is required, but if more is available the ProDOS /RAMdisk may be used. Videx Videoterm and Ultraterm cards, standard IIe and IIc 80-column hardware, and hard disks such as the Sider and Corvus are fully supported.

S-C Macro Assembler is priced at $100 for either the DOS 3.3 or the ProDos versions, or $120 for both purchased at the same time. An upgrade price is available for owners of previous versions. It is available from S-C Software Corporation, 2331 Gus Thomasson Road, Suite 125, Dallas, TX 75228, phone (214) 324-2050

Data Logging Software for IBM PC, XT, & AT

Lawson Labs has added the PC64 data logging software which allows up to 64 channels of analog input to be defined and scanned at a preset interval. Each channel has its own label, units label, offset and scaling factors, and high and low alarm limits. 16-channel strip chart recorder software is included, and the strip chart will run on an IBM graphics printer or any Epson compatible graphics printer. Data can be stored on disk automatically between scans, and the data files are fully compatible with Lotus 1-2-3 which can be used for data manipulation. Sampling can be done at rates of 1 scan per second to 1 scan per day. The software is not copy protected and is fully listable. A 128K IBM PC or equivalent with one disk drive and one Lawson Labs Model 140 A/D card is required. The price including manual is $150. Contact Lawson Labs, Inc., 5700 Raibe Road, Columbia Falls, MT 59912, phone (406) 387-5355

SBC180 Computer/Controller

Micromint has developed a single board computer using the powerful Hitachi HD64180. The SB180, only 4 × 7½" offers a Z-80 compatible CPU running at 6MHz, 256 bytes of RAM, up to 32K bytes of ROM, two serial ports, a parallel port, expansion bus, and an industry standard 756A-compatible disk controller for up to four disk drives (any combination of 3½", 5¼", or 8" drives. It can be used for a disk based computer system, or a battery-powered dedicated controller with 32K of ROM space, and can run standard 8080/8085 and Z-80 software at up to twice the speed of a 4MHz Z-80. The SB180 can run CP/M 2.2, CP/M Plus, Z-System, MP/M II, TurboDOS, and Oasis operating systems.

The SB180 with 256K bytes RAM and a ROM monitor is $369.00. A boot disk with the Z-System, limited utilities, and Super BIOS scource listings is $49.00. To order, or for more information, contact Ken Davidson at Micromint, 25 Terrace Drive, Vernon, CT 06066, phone 1-800-635-3355.

■

**DON'T PAY $700
FOR DATA ACQUISITION
SOFTWARE**

We offer full-featured, reasonably priced data logging software. Our software is well documented, unprotected and fully listable.

64 - CHANNEL CAPACITY DATA LOGGING PACKAGE for IBM PC® $150

128 - CHANNEL CAPACITY DATA LOGGING PACKAGE for APPLE II® $100

HARDWARE SUPPORTED:

IBM:	Model 140 15-bit A/D 7.5 sam/sec	$265
APPLE:	Model 34 12-bit A/D 40,000 sam/sec	$325
	Model 38 8-bit A/D 111,000 sam/sec	$165
	Model 14 13-bit A/D 15 sam/sec	$175
	Model 40 Timer/Clock 1 ms resolution	$175
BOTH:	Model 20 Differential Thermocouple Thermometer with true cold junction compensation	$175
	Model 17 16-channel differential Multiplexer	$165

**LAWSON LABS, INC.
5700 Raibe Road
Columbia Falls, MT 59912**
Phone: 406 387-5355

"BMON"
Software In-Circuit Emulator

Links your CP/M computer with any Z80 based computer or controller that you may develop. All that is needed is BMON, 8K of ROM space, and a handshakeable bi-directable I/O port (either RS232 or Parallel).

Features:

—Full program development debugger with Breakpoints, Snaps, Stops, & Waits.

—Single Step program execution.

—Download file from CP/M system to development RAM.

—Upload Memory from development RAM to CP/M disk.

—Two versions: Master BMON runs in your CP/M system, Slave BMON runs in your target system.

Note: Requires Microsoft's M80 & L80 assembler & linker to setup Slave BMON.

8" SSSD Disk containing Master BMON, Slave BMON, CONSOL, BMONIO, CONSOLIO, and Users Manual $49.95

Shipped Via prepaid UPS
—No COD or P.O. Box—
Check or Money Order to:

Barnes Research & Development
750 W. Ventura St.
Altadena, CA 91101
(818) 794-1244

*CP/M is a trademark of Digital Research Inc.
M80 & L80 are trademarks of Microsoft Inc.*

ORDER FORM

Enter my subscription to The Computer Journal for the period checked. Payment in U.S. funds is enclosed.
☐ one year (6 issues) $14 in U.S. ☐ two years (12 issues) $24 in U.S. ☐ new subscription ☐ renewal

Back Issues #'s ____________________ @$3.25 ea. __________

☐ Check enclosed ☐ VISA ☐ MasterCard Card # ____________________

Expiration date ________ Signature ____________________

Name ____________________

Address ____________________

City ____________________ State ________ ZIP ________

The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

Classified

***The Computer Journal* will carry Classified Ads. The rate is $.25 per word. All Classified Ads must be paid in advance, and will be published in the next available issue. No checking copies or proofs are supplied, so please type your ad or print legibly.**

KEYBOARDS FOR COMPUTER BUILDERS. Full ASCII, numeric pad, UC/lc, CAPS-LOCK, REPEAT, SELF-TEST! Brand new, hundreds sold to builders of Apples, Xerox 820s, Big Boards, etc. Parallel TTL output, strobe. 5 volts/100 ma. Custom case available. Keyboard $35. Documentation (21 pgs.)/cable pkg. $5. Spare CPU/ROM $4. UPS included. Detailed specs on request. Electrovalue Industrial Inc., Box 376-CJ, Morris Plains, NJ 07950. (201)-267-1117.

Voice Processor for the KAYPRO Computer. Unlimited speech contains all software. Call or write Busch Computer, PO Box 412, West Haven, CT 06516. Phone (203)484-0320.

S-100 68008 CPU BOARD. Detailed description in issue 16 of The Computer Journal. A&T $260, Kit $210, Bare Board $65. Prices include shipping. INTELLICOMP, INC., 292 Lambourne Ave., Worthington, OH 43085, Phone (614)846-0216 after 6 p.m.

S-100 Bus IEEE-488 Interface Card with cable, manuals, and software for the North Star Horizon. Purchased new from Pickles and Trout in 1979 and used once. $100. Call Phil Wells at (406) 755-1323 days of (406) 257-5326 evenings.

Corvus 10MB Hard Disk for the Apple II plus, $888.00; Apple III Second Disk Drive, $199.00; Apple III SOS Device Driver Writer's Guide $19.95; BPI Accounting for Apple III (Requires Hard Disk) $99.00; Apple Writer 1.1, 16 Sector, $8.88; Apple DOS User's Manual (II, II plus, IIe), $8.88; Apple DOS Programmer's Manual (II, II plus, IIe), $6.88; KAYPRO-Home Accountant by Continental, $49.00; Soroc IQ 130 Terminal, $399.00. All plus shipping. The Computer Place, 36 2nd Street East, Kalispell, MT 59901, Phone (406)755-1323.

THE SECURITY DISK. PROTECTED VS UNPROTECTED. At last, the best of both worlds. Here is software designed to PROTECT YOUR PRIVATE FILES. SIMPLE PROTECTION TO MULTI-LEVEL CRYPTOLOGY. Plus "DYSUN::THE DISK UNLOCKER" WILL ANALYZE & UNLOCK COMMERCIAL "COPY-PROTECTED" DISKS, then recopy them to standard DOS 3.3 format. "DYSUN" will also RECOVER LOST DATA & REPAIR BLOWN DISKS. A special "SECRETS-TIPS-TECHNIQUES" program is also included. Not locked-up. Listable. Machine Language Source Codes included. Supports Apple II, II Plus, IIc, IIe. To order send $29.95 CHECK/MO to B.M.E. Enterprises, Box 191-J, Kila, MT 59920.

Morrow Decision I S-100 system with MPZ-80 CPU, DJ/DMA floppy disk controller, 256K static ram, Wonderbus I/O on mother board, Disk Jockey Hard Disk (HDCA) Controller, 801 floppy drive, 10MB hard disk, CP/M, Micronix Multiuser system, unconfigured MP/MII, dBase II, Wordstar, Accounting Plus. Excellent condition. $3500, some trades considered. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406)257-9119.

Book Sale—These books are offered at this price while the supply lasts.
Zilog Z80-CPU Technical Manual, $1.50
Real Time Programming—Neglected Topics
  by Caxton C. Foster, $8.00
CBASIC Users Guide
  by Osborne, Eubanks, and McNiff, $14.00
Introduction to FORTH
  by Ken Knecht, $9.00
FORTH Programming
  by Leo J. Scanlon, $13.00
These prices are postpaid in the U.S. only. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

Books of Interest

Apple II/IIe
Robotic Arm Projects
by John Blankenship
Published by Prentice-Hall
149 pages, 7 × 9″

Robotics is an interesting subject, and as Blankenship states, "More than ever before, there is a need for anyone interested in computers and/or electronics to learn about motor control, artificial intelligence, and other robotic-related subjects." While there are many robotics books on the market; the serious ones assume that you have a Ph.D and are so theoretical that they don't help in average practical applications, and the hobbyist books have little or no useful information.

This book is a pleasant combination of fundamental information and simple demonstration programs for those who want to get started in this interesting field. The author's goal is to enable you to understand the basic principles through simple projects, so that you can create your own designs and not just imitate his work. The contents of the book are as follows:

•**Chapter 1 Introduction.** Industrial vs. hobby robots, constructing the robot, what is a robot? software and interfacing, robots can be fun.
•**Chapter 2 A Simulated X-Y Arm.** Why simulation? simple videobot three dimensional video simulation, Using the simulator, BASIC X-Y simulator program.
•**Chapter 3 Intelligent Control of the X-Y Simulator.** Defining the goal, command syntax, using the flowchart, designing by defining, logical control structures, folwcharting the modules, multipurpose modules, coding the program, BASIC Blocks World Simulation program.
•**Chapter 4 A Mechanical X-Y Arm.** Construction of a physical arm, using surplus parts, motor control, electronic noise problems, optical isolation, open vs closed loop systems, eliminating bounce, software counters, Assembly Rev Counter program, input/Output ports, the 6522 VIA, plotting with the arm, BASIC X-Y Plot Demo program.
•**Chapter 5 Vision Systems.** Types of Vision, restrictions and limitations, objectives, a simple example BASIC vision 1 program, practical uses, scaling factors, BASIC Vision Algorithm program, centroids and vectors, condensing the data, vision hardware, a HIRES SCRN function, HIRES SCRN function assembly program.
•**Chapter 6 A Simulated Jointed Arm.** Why joints? trigonometry needed for joint simulation, coding the simulator, manual control of the simulator, BASIC Manual Jointed program.
•**Chapter 7 Intelligent Control of the Jointed Simulator.** Manual vs computer control, converting X-Y coordinates to joint angles, converting to steps, algorithm to ensure that all joints will finish at the same time, BASIC Intelligent Control of a Simulated Jointed Arm program.
•**Chapter 8 Control of Stepper Motors.** How a stepper motor works, step variations, half stepping, interfacing, bipolar and unipolar motors, program control, integrated circuit controllers, sample BASIC Stepper Motor Control program.
•**Chapter 9 Building a Jointed Arm.** Flexibility, construction, attaching motors, the software, BASIC Intelligant Control of a Stepper Motor Arm program, controlling the motor.
•**Chapter 10 Positional Control of DC Motors.** Why DC motors? comparision of Stepper and DC motors, overshoot, a test platform, the computer interface, designing the program, coding the program, BASIC Motor Positional Control System program.
•**Chapter 11 Hobby Servo Motors.** Using radio controlled model servos, generating the pulses from the game port, the interrupt program, Quad Servo Drive System assembly program, BASIC Servo Control Demo program.
•**Chapter 12 An Arm for a Hand Puppet.** constructing a robot puppet, interfacing the motors, the software, Interrupt Timing System assembly program, BASIC Puppet Controller program.
•**Chapter 13 Computerizing a Toy Arm.** Why use toys? The mechanical connection, controlling the solenoids, sensory feedback, designing the software, BASIC Armatron Controller program.
•**Chapter 14 Man-Sized Arms.** Degrees of freedom, making compromises, designing the manipulator, building working models, designing the wrist.
•**Appendix A Sources for Robot Parts.**
•**Appendix B Optional Diskette Information.**

Although the book is rather thin (146 pages of text), it is tightly written and contains a lot of useful information. Some of the subjects are covered in a mere pharagraph or two, which was done to avoid overwhelming the beginner with more information than he needs. Other books can be consulted for more details on specific subjects after the fundamentals are absorbed. The author has used BASIC for most of the programs, not because it is necessarily the best-suited language for robotic experimentation, but because the majority of the readers will be familiar with it. The programs in the book are also available on disk for $19.95 for those who would rather not key the listings.

This book should be useful for someone's first indroduction to building and controlling robotic devices using low-cost junk box and surplus parts. ■

Apple II and IIe are registered trademarks of the Apple Computer Corporation.

CTM

"...received my moneys worth with just one issue..."

—J. Trenbick

"...always stop to read CTM, even though most other magazines I receive (and write for) only get cursory examination..."

—Fred Blechman, K6UGT

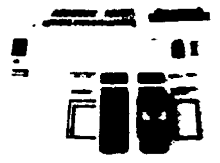
U.S.A.	$15.00 for 1 year
Mexico, Canada	$25.00
Foreign	$35.00(land) - $55.00(air)
(U.S. funds only)	
Permanent (U.S. Subscription)	$100.00
Sample Copy	$3.50

CHET LAMBERT, W4WDR
1704 Sam Drive • Birmingham, AL 35235
(205) 854-0271

Advertiser's Index

Alliance Computers	20
Apropos	43
Arctec	4
Barnes Research	39
John Bell	8
Bersearch	15
Business Utility Software	5
BV Engineering	36
Classifieds	37
Computer Trader	43
Davidge	43
Echelon, Inc.	19, 35
Educational Microcomputer Systems	20
Intellicomp	33
Jerryco	28
Lawson Labs	39
Miller Microcomputer Services	11
Next Generation Systems	17
Public Domain Software	34

APROTEK 1000™ EPROM PROGRAMMER



TECHNICAL BREAKTHROUGH NOW ALLOWS A PRICE BREAKTHROUGH

only
$250.00

A SIMPLE, INEXPENSIVE SOLUTION TO PROGRAMMING EPROMS

The ***APROTEK 1000*** can program 5 volt, 25XX series through 2564, 27XX series through 27256 and 68XX devices plus any CMOS versions of the above types. Included with each programmer is a personality module of your choice (others are only $10.00 ea. when purchased with ***APROTEK 1000***) Later, you may require future modules at only $15.00 ea., postage paid. Available personality modules: PM2716, PM2732, PM2732A, PM2764, PM2764A, PM27128, PM27256, PM2532, PM2564, PM68764 (includes 68766) (Please specify modules by these numbers).

APROTEK 1000 comes complete with a menu driven BASIC driver programmer listing which allows READ, WRITE, COPY, and VERIFY with Checksum. Easily adapted for use with IBM, Apple, Kaypro, and other microcomputers with a RS-232 port. Also included is a menu driven CPM assembly language driver listing with Z-80 (DART) and 8080 (8251) I/O port examples. Interface is a simple 3-wire RS-232C with a female DB-25 connector. A handshake character is sent by the programmer after programming each byte. The interface is switch selectable at the following 6 baud rates: 300, 1.2k, 2.4k, 4.8k, 9.6k and 19.2k baud. Data format for programming is "absolute code" (i.e., it will program exactly what it is sent starting at EPROM address 0). Other standard downloading formats are easily converted to absolute (object) code.

The ***APROTEK 1000*** is truly universal. It comes standard at 117 VAC 50/60 HZ and may be internally jumpered for 220-240 VAC 50/60 AZ. FCC verification (CLASS B) has been obtained for the ***APROTEK 1000***.

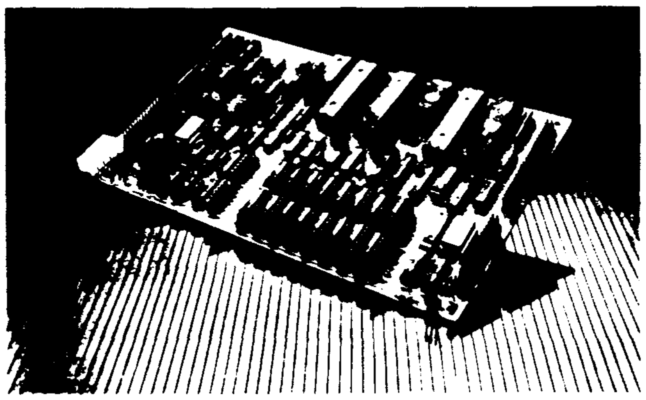
APROTEK 1000 is covered by a 1 year parts and labor warranty.

FINALLY — A Simple, Inexpensive Solution To Erasing EPROMS

APROTEK-200™ EPROM ERASER
Simply insert one or two EPROMS and switch ON. In about 10 minutes, you switch OFF and are ready to reprogram.
APROTEK-200™ only ***$45.00.***

APROTEK-300™ only ***$60.00.***
This eraser is identical to ***APROTEK-200™*** but has a built-in timer so that the ultraviolet lamp automatically turns off in 10 minutes, eliminating any risk of overexposure damage to your EPROMS.
APROTEK-300™ only ***$60.00.***

APROPOS TECHNOLOGY
1071-A Avenida Acaso, Camarillo, CA 93010
CALL OUR TOLL FREE ORDER LINES TODAY:
1-(800) 962-5800 USA or 1-(800) 962-3800 CALIFORNIA
TECHNICAL INFORMATION: 1-(805) 482-3604
Add Shipping Per Item: $3.00 Cont. U.S. $6.00 CAN, Mexico, HI, AK, UPS Blue



POWER THAT GOES ANYWHERE!

Single Board Computer

FAST	— 6MHz Z80B* CPU
POWERFUL	— 64K to 256K RAM, 2K to 64K ROM
	— 5¼" and 8" Floppy Controller, SASI
	— 2 RS-232, Centronics Port
FLEXIBLE	— 50-pin I/O Expansion Bus
SMALL	— 5¾" x 10"

DAVIDGE CORPORATION
292 East Highway 246
PO Box 1869
Buellton, CA 93427
(805) 688-9598

*Z80 is a registered trademark of Zilog

THE COMPUTER CORNER

A Column by Bill Kibler

It is the hot time of the summer and computing is starting to take a back seat to other forms of recreation. I have had a chance to reflect and review my computing needs while my systems were shut down during the last few weeks of moving to a new location. My vacation is about to start (sure need one after moving all those boxes) and the industry as a whole is doing a two step, and I thought this would be a good time to review pending projects and ideas for them.

Some months back I stated an interest in doing machine to machine linking and have since found several products already available. I usually use public domain software as my source for most of my projects, and BYE and XMODEM will allow most users to handle their system to system needs. Packet radio links have still got my interest and you will see something later from me for sure.

I received one inquiry on my putting together a FORTH ROM based system and all the problems that it implies. This idea has also become my next project and hopefully will be my next special article for *The Computer Journal.* The idea here is first to check out the public domain disks for source code, go after the bugs and unknown problems, followed by implementation and uses. Now FORTH has some good points and some bad points, but for robotics or solar system control the idea of being able to tailor an operating system to consist of just the code needed for the project (and maybe portable to other systems) is hard to beat. If someone out there has done this already and would like to write about it, give the editors here at TCJ a call and I will go on to some other projects.

I haven't got my hands on anything new lately, as the market for used systems is pretty bad. My old units will not sell and so I haven't any money for the new (used) stuff. With the market changes and the maturity of the systems, I am rethinking all my efforts. I am still planning on buying one of the 68000 machines later this year, and maybe use it for the FORTH system. I'll be attending Micro Cornucopia's SOG IV (unofficial get-together) and should have some thoughts on my projects (and comments on the SOG too). What this is all coming down to is the state of the industry, pretty shakey.

There has been a real shakeout in computer companies, but a most interesting fact surfaced last month. Apple reported that over 60% of it's sales were for the TEN year old Apple II's. This amounts to about 450,000 systems by Apple (plus who knows how many copies) which is still a considerable amount of units. The software is not quite the same story, as very few new programs have been released. I liken this whole problem to autos, especially antiques, which are still rebuilt and sold and even driven a lot, with some units that are over 40 years old. Now computers aren't quite 40 years old but there are still a lot of us that will keep using our old systems, no matter what, and I expect to see some support to spring up in this area soon (other than magazines like TCJ).

A few final comments before I grab my backpack and head out the door. My Remex drives are still working, the modified units have only an occasional error, while the unchanged one gets a lot. I have heard of others dying for ever, but have not gotten any information from our readers as to why, yet. The cost of print spoolers is dropping and for those who didn't build the one in TCJ I'll have some words about an S-100 unit next month. For you clone builders, check the layouts well as some bare boards need by-passes and tuning caps in some places to make them work. Seems some of the timing windows were rather poorly done and may need some slowing down just to work. These may be undocumented spots for capicators, and require trial and error work to get the correct value.

Had some troubles with other mini drives, and remembered to look for inverse logic. It never fails to catch you when you're sure you know what you're doing. A TEK drive wouldn't work right until all the switches were turned on and the drive selected switch turned off. So when faced with a series of switch positions and the unit doesn't work, try reversing all logic, you will be surprised how often the documentation is wrong about this item. Also had a drive motor sieze up on me, seems the Oillite bearings had no oil. If you decide to take it apart, it is not for the bumble fingers, these are small and will give you problems getting the brushes altogether again. Try unbending the tabs at the back and putting the brushes in after partial assembly. Soak those bearings in a good light oil **not WD-40 types** (they will strip out the good oils). On some of the double sided drives, watch out when cleaning the heads. The floating head is just that, a floating head mounted by thin strips of metal. I got to rebend the metal gimbal of one that had been caught by a cotton swab, and bent badly out of shape. The unit now has a crease in it, but checked out OK after realignment.

Now where is my freeze-dried food??? bye.... ■

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used marks are acknowledged, and we apologize for any we have overlooked.

Apple II, II + , IIc, IIe, Macintosch, DOS 3.3, ProDOS; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. MBASIC; Microsoft. Wordstar; MicroPro International Corp. IBM-PC, XT, and AT; IBM Corporation.

Where these terms (and others) are used in The Computer Journal they are acknowledged to be the property of the respective companies even if not specifically mentioned in each occurence.