THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Vol. II, No. 4 $2.50 US

Editor's Page

The Customer is Tired of Being Ignored!

Many of the companies in the microcomputer industry have developed a very simple customer relations policy—they just ignore correspondence and phone calls. This may have been a cost-effective way of handling the problem in an expanding market filled with eager and unsophisticated buyers, but experienced business people will not put up with this shoddy treatment in the current buyer's market.

The microcomputer industry is still structured around the pioneer companies started by a technically oriented entrepreneur with little cash and no business experience who ran the business out of a spare bedroom. These undercapitalized, understaffed enterprises made millions because there was no competition, and because the owners were too busy taking money to the bank to worry about the customer. Their problem was how to expand fast enough to meet the unfilled demands. It didn't matter if the customer was unsatisfied because the product was already paid for (they don't give refunds if the product doesn't work), and there were dozens of new buyers to replace every lost customer.

Good documentation and customer support are extremely important for business equipment and software. With the large number of vendors competing for your dollar today, your decision to buy any micro product should be largely based on how you will be treated by the company after the purchase. In order to help you compare customer support policies, *The Computer Journal* is surveying vendors and will publish the customer relations policy with the addresses and phone numbers. We will also encourage them to submit bug fixes, customization notes, and interfacing information for publication.

Realizing that what happens in the real world is often different than the published policy, we will also publish the experiences of our readers, after giving the vendors the opportunity to respond. We do need your experiences with vendors (both the good and the bad) to help others avoid the companies with poor products and miserable customer relations.

Our loyalty is to our readers, and not to any possible advertisers. We will not hesitate to report bad products and poor customer support, but we will need copies of correspondence and records of phone calls which we can present to the vendors prior to publication.

We have had a bad experience with a computer system we purchased recently. We wrote twice for information on an upgrade, but they have not answered our letters. We liked their product, but will not purchase any other equipment from them or recommend their equipment to anyone else. We are sending copies of the letters to the vendor for their reply before publishing the details.

This magazine is published for you, and it will only be as good as you make it. Take time to tell us about your vendor experiences, and share your bug fixes, customization patches, interfacing information and technical tips with the other readers. ■

Editor/Publisher.....*Art Carlson*
Art Director.....*Joan Thompson*
Technical Editor.....*Lance Rose*
Production Assistant.....*Judie Overbeek*
Contributing Editor.....*Ernie Brooner*

The Computer Journal® is published 12 times a year. Annual subscription is $24 in the U.S., $30 in Canada, and $48 airmail in other countries.

Postmaster: Send address changes to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquires to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

BUILD A VIC-20 EPROM PROGRAMMER

by Neil Bungard

Introduction

About a month ago I was presented with an interesting problem. I had acquired a Z-80 based S-100 bus microcomputer which had no operating software installed on it. I had an operating system stored on paper tape, but I needed the operating system on an EPROM before my system would run it. The EPROM programmers that I looked at were too expensive to justify buying simply to program one EPROM, so I looked for an alternative solution. It so happened that I had recently purchased a VIC-20, so I decided to design an EPROM programmer attachment for the VIC-20 and use the VIC-20 to transfer the Z-80 operating software from the paper tape to the EPROM. The VIC-20 solved my EPROM programming problem with exceptional grace. With the VIC-20 and ten dollars worth of additional hardware, I was able to enter, inspect, program, and verify data on both 2716 (2K by 8) and 2732 (4K by 8) EPROMs. In addition, the VIC-20 EPROM programmer has features not found on even the most expensive EPROM programmers. Data to be programmed into the EPROM can be entered from cassette tape, 5" diskettes, any RS-232 device, from the VIC-20 keyboard, or even from another EPROM.

Circuit Description

The VIC-20 EPROM programmer circuit diagram is shown in Figure 1. This circuit was designed to program 2716 (2K by 8) EPROMs, but with minor circuit modifications the VIC-20 can program 2732 (4K by 8) EPROMs as well. These modifications will be explained in detail, and can be hard wired or wired so that "personality modules" can be inserted to select between programming 2716s or 2732s.

The VIC-20 EPROM Programmer performs four basic functions:

1. It can set the address of where data trasfers are to begin on the EPROM.
2. It can read data from a "read only" EPROM. This is used in the EPROM duplicate mode.
3. It can read data from a "programable " EPROM. This is used in the program verification mode.
4. It can program an EPROM. This, of course, is used in the program mode.

To obtain an understanding of the operation of the EPROM programmer circuit in each mode, let us look at these four basic functions separately.

The first function, address selection, is accomplished by loading three presettable counters with the beginning address of where you wish to store or retrieve data on the EPROM. The counters are 74LS193s (IC 1, 2, and 3 in Figure 1). The inputs of the counters are tied through buffers to the data bus of the VIC-20. The outputs of the counters are connected to the address lines of the 2716s so that the values loaded into the 74LS193s will be present on the address lines of the EPROMs. With an address value loaded into the 74LS193s and present on both EPROMs, a memory read from either EPROM or a memory write to the programable EPROM can be accomplished. Once the data transfer operation is accomplished, the selected address can either be advanced in increments of one, or a new address can be loaded into the counters.

Values are loaded into selected counters by the VIC-20 via an Input/Output (I/O) device code generator (IC4). The device code generator (a 74LS138) is a three line to eight line decoder. Its three input lines are tied through buffers to the three lowest order address bits of the VIC-20 address bus. When the VIC-20 wants to accomplish an information transfer (either Input or Output) it places the address of where the data exchange is to take place onto pins 1, 2, and 3 of the 74LS138. The VIC-20 then pulses "BLK1" to a logic 0. This generates a device code pulse on one of eight output pins on the 74LS138 which accomplishes a data transfer to/from one of eight unique address locations defined by the values of A0, A1, and A2. Table 1 shows the eight data transfer pulses generated on the EPROM programmer. Figure 2 is the timing diagram of the device code pulse generated.

With reference to Table 1, the following software is required to operate the device select hardware and set a given address location for a memory transfer:

```
10 POKE 8195,0
20 POKE 8192,0
```

This program addresses memory location 0 (decimal) on the EPROMs. 10 POKE 8195,0 loads the four higher order bits of the 12 bit address into address counter IC3 by placing the high order address value on data bus bits D0 through D3 and pulsing pin 11 of IC3 to a logic 0. 20 POKE 8192,0 loads the eight low order address bits into counters IC1 and IC2

Device Code Address	Output Pin of 74LS138	Use of Device Code Pulse
8192	15	loads the 8 lower order bits of the address counter
8193	NA	NA
8194	13	transfers data from "programmable" EPROM to the VIC 20
8195	12	Loads the 4 higher order bits of the address counters
8196	11	transfers data from "read only" EPROM to the VIC 20
8197	10	programs EPROM and increments the address counters
8198	NA	NA
8199	NA	NA

Table 1

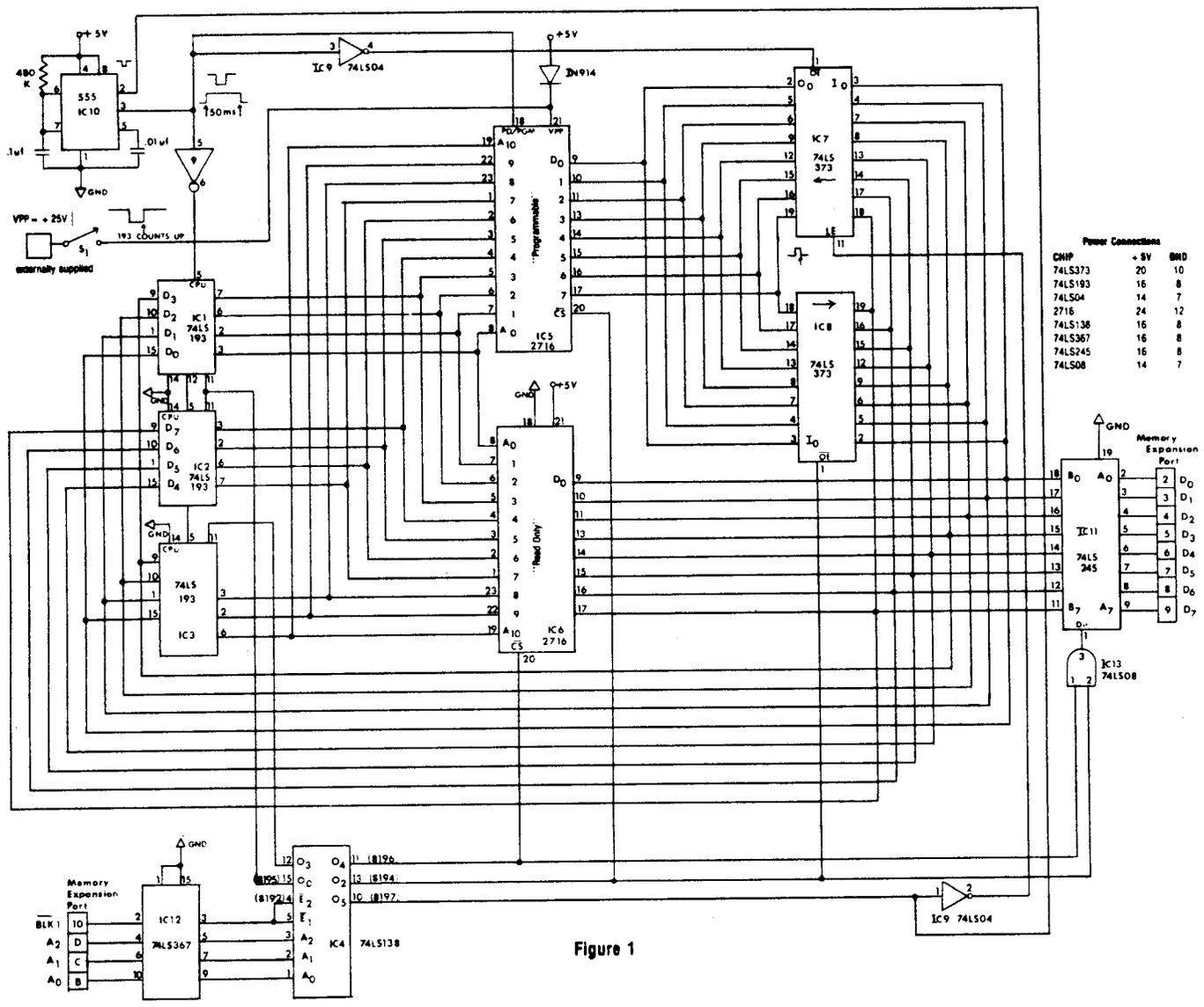


Figure 1

by placing the EPROM address value on the data bus and pulsing pin 11 of IC1 and IC2 to a logic 0. To advance the counters, which increments the selected address, the following instructions are required:

10 POKE 8197,0

20 FOR I = 0 TO 50: NEXT I

POKE 8197,0 generates a device code pulse on pin 10 of the 74LS138 (IC4). This low going pulse triggers a monostable multivibrator (IC10), which is configured to generate a 50ms high-going pulse. This 50ms pulse is required in the EPROM programming sequence which will be discussed later. The output of the monostable (pin 3, IC10) is connected, through an inverter, to the "up-count" input of IC1. If the READ/PROGRAM switch (S1, Figure 1) is in the READ position (open), the selected address will be advanced by one when the monostable times out, but no EPROM memory location will be programmed. However, if the READ/PROGRAM switch is in the PROGRAM (closed) position when the POKE 8197,0 instruction is executed, address location 0 (decimal) on the programmable EPROM will be loaded with a 0 before the address counter (IC1) is advanced. Details of this sequence will be discussed further in the explanation of the programming mode.

The important thing to remember in our discussion thus far is that for the purpose of reading either EPROM, the READ/PROGRAM switch must be in the READ position. The second instruction 20 FOR I = 0 TO 50: NEXT I is a software time delay to allow the monostable to time out before any other instructions are executed. Even though the 50ms pulse is only used in the programming mode it is still generated in the read and verify modes to advance the counters. It is the trailing edge of this pulse that always advances the address counters and thus must be accounted for in all the modes of operation.

The second basic function performed by the EPROM programmer is reading data from the "read only" EPROM. In order to do this, the memory address of where the read operation is to occur is loaded into the memory address counters as previously explained. Next, a memory read operation is performed by the VIC-20 which generates an input device code pulse on pin 11 of the device code pulse generator IC4. Pin 11 of IC4 is connected to the chip select

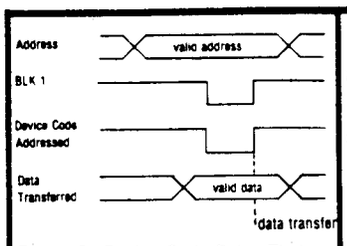


Figure 2: Device Code Pulse Timing.

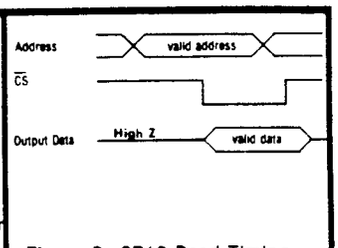


Figure 3: 2716 Read Timing.

(CS) input of the read only 2716 (IC6). When the CS input (pin 20) is pulsed low (with pin 18 tied low and pin 21 held high) data flows from the memory location addressed on the EPROM into the VIC-20. Figure 3 shows the timing diagram of the 2716 read operation. The following software is required to perform the read only EPROM read operation:

```
10 POKE 8195,0
20 POKE 8192,0
30 A = PEEK(8196)
```

The BASIC instructions in lines 10 and 20 set the memory address of where the read operation will occur. These two instructions have been previously explained. The instruction 30 A = PEEK(8196) generates a device code pulse on pin 11 of IC4 which inputs the data from the addressed memory location, and assigns the data to the variable name "A".

The third basic function performed by the EPROM programmer, reading data from the programmable EPROM, is accomplished in exactly the same manner as reading from the read only EPROM. The only difference is that a device code pulse is generated on pin 13 of IC4 instead of pin 11 as in the case of the read only EPROM. Pin 13 of IC4 is connected to the CS input of the programmable EPROM (pin 20, IC5) and to the output enable (OE) of an octal tristate buffer (pin 1, IC8). Data flows out of this EPROM and through the buffer when pin 13 of the 74LS138 is pulsed low. For the read operation to be accomplished, pin 18 of the 2716 must be at a logic 0 and pin 21 at +5 volts. The software required to read data from the programmable EPROM is:

```
10 POKE 8195,0
20 POKE 8192,0
30 A = PEEK(8194)
```

The fourth and last function performed by the EPROM programmer is programming an EPROM. To enter the program mode, the READ/PROGRAM switch must be placed in the PROGRAM position. This applies +25 volts to pin 21 of the EPROM (IC5), and configures the EPROM to be programmed. Be sure that the 25 volt supply is stable and at no time fluctuates higher than 26 volts. **A voltage greater than 26 volts on pin 21 of the EPROM will immediately destroy the EPROM.** Next, as was the case when reading the EPROMs, the memory address of where the data transfer is to take place must be loaded into the address counters. By applying an address to the EPROM, placing pin 21 at +25 volts, and insuring that the CS input is at a logic 1, the EPROM is ready to be programmed. Applying a data byte on D0 through D7 of the EPROM, and pulsing pin 18 (PD/PGM) to a logic 1 for 50ms completes the programming operation. The timing diagram for the 2716 programming sequence is shown in Figure 4. The data is applied to the EPROM by the VIC-20 through a memory write operation to location 8197 (decimal). This memory write operation generates a device code pulse on pin 10 of IC4 which latches the outputted data byte into IC7 and triggers the monostable multivibrator (IC10). The monostable applies a 50ms high-going pulse to pin 18 of the EPROM and programs the applied memory address with the data byte latched in IC7. The falling edge of the 50ms programming pulse automatically advances the counters and prepares the next memory address for programming. The following software is required to program address location 0 with decimal value of 100:

```
10 POKE 8195,0
20 POKE 8192,0
30 POKE 8197,100
```

Application Software for the VIC-20 EPROM Programmer

The software required to read, verify, and program 2716 EPROMs will be presented in the following discussion on application software. To perform an EPROM duplication, the contents of a previously programmed EPROM is transferred into the VIC-20's RAM memory. Here the data can be inspected and/or modified if desired. The data is then transferred to a blank EPROM, and a verification is accomplished to ensure that no error occured in the data transfer. The following software reads the 2048 memory locations of a 2716 and places the data into the VIC-20's RAM memory between locations 5000 (decimal) and 7048(decimal). Insure that the READ/PROGRAM switch is in the READ position, and that the EPROM to be read is inserted into the read only EPROM socket:

```
10 PRINT ""                      /clear screen
20 PRINT "SET SWITCH TO"
30 PRINT "READ POSITION"         /ensure that
40 PRINT                         /READ/PROGRAM
50 PRINT "HIT RETURN"            /switch is in
60 PRINT "TO START"              /correct position
70 INPUT X
80 POKE 8195,0                   /load high-order 74LS193
90 POKE 8192,0                   /load low-order 74LS193
100 FOR I = 5000 to 7048         /set 2048 RAM spaces
110 A = PEEK(8196)               /read EPROM
120 POKE I,A                     /store value in RAM
130 POKE 8197,0                  /advance counters
140 FOR T = 0 to 50:NEXT T       /wait for 50ms pulse
150 NEXT I                       /to read next location
160 PRINT"READ COMPLETE"         /prompt user
170 STOP
```

*Do not enter comments in right hand column as part of program.

Once the data is in RAM memory, the user can check the code to ensure that the correct 2716 is being duplicated, and/or make minor modifications to the code. Once satisfied with the data in RAM, the data is transferred to a blank EPROM which has been inserted into the programmable EPROM socket. Before the transfer takes place, the READ/PROGRAM switch must be placed in the

PROGRAM position. The following software loads the data which is in the VIC-20's RAM memory space, between address 5000 (decimal) and 7048 (decimal), into a 2716 EPROM:

```
200 PRINT ""                          /clear the screen
210 PRINT"SET SWITCH TO"
220 PRINT"PROGRAM POSITION"           /ensure that
230 PRINT                             /READ/PROGRAM
240 PRINT"HIT RETURN"                 /switch is in
250 PRINT"TO START"                   /correct position
260 INPUT X
270 POKE 8195,0                       /load high-order 74LS193
280 POKE 8192,0                       /load low-order 74LS193
290 FOR I = 5000 to 7048              /set 2048 RAM spaces
300 A = PEEK(I)                       /read data from RAM
310 POKE 8197,A                       /write data to EPROM
320 FOR T = 0 to 50:NEXT T            /wait for 50ms pulse
330 NEXT I                            /go read next location
340 PRINT"PROGRAMMING COMPLETE"       /prompt user
350 STOP
```

When the programming is complete, you should run the following verification program to ensure an accurate EPROM copy:

```
400 PRINT ""                          /clear the screen
410 PRINT"SET SWITCH TO"
420 PRINT"READ POSITION"              /ensure that
430 PRINT                             /READ/PROGRAM
440 PRINT"HIT RETURN"                 /switch is in
450 PRINT"TO START"                   /correct position
455 INPUT X
460 POKE 8195,0                       /load high-order 74LS193
470 POKE 8192,0                       /load low-order 74LS193
480 FOR I = 5000 to 7048              /set 2048 RAM spaces
490 A = PEEK(8194)                    /read programmable EPROM
500 B = PEEK(I)                       /read data from RAM
510 IF A<>B THEN 570                  /if no match then stop
520 POKE 8197,0                       /if match, advance counters
530 FOR T = 0 to 50:NEXT T            /wait for 50ms pulse
540 NEXT I                            /go do next comparison
550 PRINT"DUPLICATION GOOD"           /prompt user
560 STOP
570 PRINT"DUPLICATION BAD"
580 PRINT"ERROR IN"                   /prompt user
590 PRINT"ADDRESS";I
```

As mentioned in the introduction, EPROMs programmed by the VIC-20 EPROM programmer can receive their data from a number of sources. The only modifications necessary to input data from other sources would be between lines 80 and 160 of the BASIC code. These lines of code move data from some source (another EPROM in the above case) and place the data in the VIC-20's RAM memory. Using the EPROM programmer for my own application, I was required to take data from the VIC-20's RS-232 interface port. The following software accomplishes a data transfer from the RS-232 serial port and places the data into RAM memory locations 5000 (decimal) through 7048 (decimal):

```
80 OPEN 2,2,3 CHR$(163)+CHR$(160)   /open the channel, 110 baud,
   2 stop bits
90 FOR I = 5000 to 7048             /set 2048 RAM spaces
100 GET 2,A$                        /turn on receiver channel
110 IF A$ = "" THEN 100             /ignore a null
120 A = PEEK(667)                   /find serial byte in buffer
130 B = A + 7423                    /define buffer location
140 C = PEEK(B)                     /get data from serial port buffer
150 POKE I,C                        /put data in RAM memory
155 NEXT I                          /go get next data byte
```

Another possibility for entering data is through the VIC-20's keyboard. This is not practical for larger data bases, but in many cases the data is less than a few hundred bytes. For that amount of data the keyboard is not a bad option. The following software will load data into RAM memory starting at location 5000 (decimal), via the vic-20's keyboard:

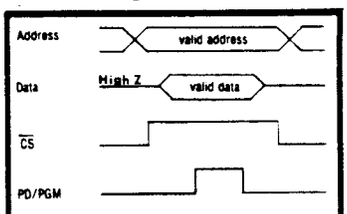


Figure 4: 2716 Program Timing.
*voltage on pin 21 of the 2716 must be + 25Volts

Enter an "S" to terminate the entry program.

```
80 PRINT ""                           /clear the screen
85 FOR I = 1 to 2048                  /input up to 2048 times
90 PRINT"ENTER DATA"                  /prompt user
95 PRINT"BYTE";I                      /to enter
100 INPUT A$                          /data byte
105 IF A$ = "S" THEN 160              /enter S to stop
110 A = VAL(A$)                       /convert input string to a number
115 J = I + 5000                      /define RAM storage location
120 POKE J,A                          /put data in RAM memory
125 PRINT ""                          /clear the screen
130 NEXT I                            /go input next data value
135 PRINT"NO MORE ROOM"
140 PRINT"ON EPROM"                   /prompt user
145 STOP
```

A Word About Bus Buffering

My original VIC-20 EPROM programmer design did not have Data and Address bus buffers, but I encountered some problems when inserting and removing the EPROMs. When I inserted or installed the "read only" 2716 EPROM, the VIC-20 operating system would crash. I installed the 74 245 data bus buffer and the 74LS367 address bus buffer and the problem disappeared. In general it is a good idea to buffer all of your interface projects. This ensures that you have good "drive" from your computer, and buffering isolates the CPU and other ICs in your computer from your circuit, which could avoid damage to the computer as a result of wiring errors.

Programming 2732s On the VIC-20 EPROM Programmer

So far only 2716 (2K by 8) EPROMs have been mentioned with reference to the VIC-20 EPROM programmer. The programmer can also program 2732 (4K by 8) EPROMs. There are only three pins which differ between the two EPROMs. These are pins 18, 20, and 21. Figure 5 shows the connection changes necessary to use the VIC-20 EPROM programmer with 2732 EPROMs. These connections can be hard wired to configure the EPROM programmer for 2732s only, or a 14 pin wire wrap socket can be placed on the EPROM programmer board and wired so that jumper wires

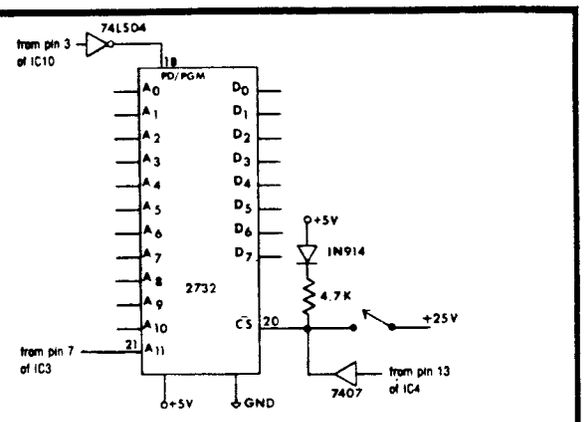


Figure 5: 2716 to 2732 connection changes.

can be used to choose between programming 2716s and 2732s. Using this method, *personality modules* can be constructed which plug into the 14 pin socket. These modules determine whether the programmer is configured for 2716 or 2732 EPROMs by providing the correct jumper wires across the 14 pin socket. Figure 6 shows how the socket is wired to achieve the EPROM selection using the personality modules. Also in Figure 6 are the wiring configurations necessary to construct both the 2716 and the 2732 personality modules. The modules themselves are constructed from 14 pin DIP headers, with jumper wires soldered in place on the header to configure the module.

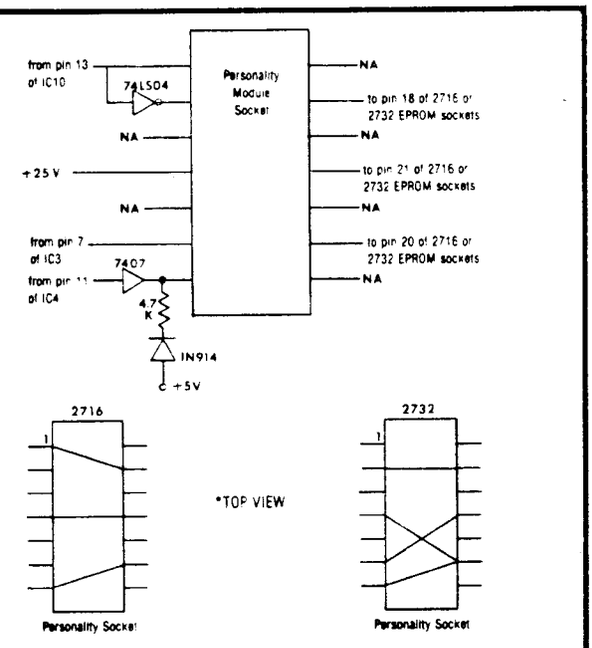


Figure 6: Personality Modules and Socket

Construction of the VIC-20 EPROM Programmer

The VIC-20 EPROM programmer can easily be constructed on a perforated Vectorboard equipped with edgeboard fingers. The edgeboard fingers must be double sided and the pin spacing must be 0.156 inches. The Vectorboard is standard equipment, and can be purchased from most electronic parts distributors.

I used wire wrap techniques to construct the EPROM programmer. There are no critical wiring considerations, and the project took less than five hours to construct. A photograph of the VIC-20 EPROM programmer is shown in Figure 7, and a pin assignment of the VIC-20 expansion port is shown in Figure 8 to aid in construction of the programmer.

It should be noted that an external 25 volt power supply is required to program the 2716 and 2732 EPROMs.

The software routines used to program the 2716s will also work with the 2732s. The only changes required in the software routines are in the instructions which account for the 2716's 2048 memory locations. To program 2732s on the VIC-20 without expanding the VIC RAM memory, the 2732's 4K memory should be considered as two 2K blocks. Each 2K block would be programmed separately, which would require two passes through the software routines. On the first pass the address counters should be loaded with a starting address of zero and the first 2K locations would be programmed the same as a 2716. On the second pass the address counters should be loaded with a starting address of 2048 and the next 2K locations would be programmed.

Conclusion

The software routines presented in this article are what I would consider utility routines. They make the VIC-20 EPROM programmer work, but they are neither instructive nor "friendly" to the user. Providing only the required utility routines allows you to develop an "operating system" to suit your particular needs.

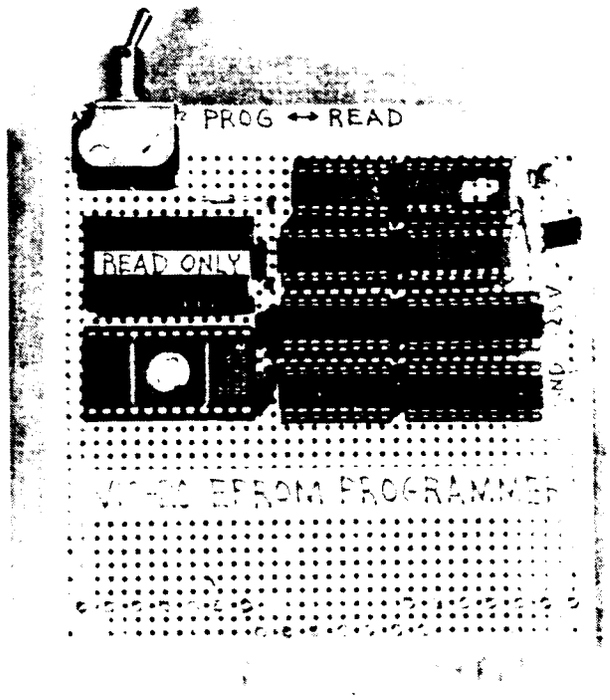


Figure 7

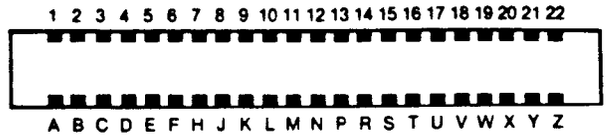
If you want to use the VIC-20 EPROM programmer as a general purpose programming tool for both 2716 and 2732 EPROMs, I would suggest a "prompt and queue" operating system. By "prompt and queue" I mean that the computer prompts the user and asks the user what he/she would like to do. The user in turn enters the proper queue which instructs the computer to perform a specific task. This type of operating system can be implemented nicely with the use of menus. For example, you might have the VIC-20 ask the following question:

WHICH TYPE OF EPROM DO
YOU WISH TO CONSIDER?

If you input "2716", the VIC-20 might print the following menu:

WHAT FUNCTION DO YOU WISH TO PERFORM?
1) EXAMINE CODE ON A 2716 EPROM
2) PROGRAM A 2716 EPROM
3) DUPLICATE A 2716 EPROM
4) ENTER CODE THROUGH KEYBOARD
5) ENTER CODE FROM TAPE
6) ENTER CODE FROM RS232 PORT
7) ENTER CODE FROM DISK
8) QUIT
INPUT YOUR SELECTION?

This menu allows you to perform all the functions which are possible on the VIC-20 EPROM programmer. Selecting a desired function might result in another menu being printed which would help the user choose the proper parameters used in a utility routine. In addition to helping with parameters, the VIC-20 could instruct the user as to which "personality module" to use and when to insert it. Your imagination is your only limitation when designing an operating system for the VIC-20 EPROM programmer. Have fun with it and drop us a line to let us know what you come up with. ■



PIN #	TYPE
1	GND
2	CD0
3	CD1
4	CD2
5	CD3
6	CD4
7	CD5
8	CD6
9	CD7
10	$\overline{BLK1}$
11	$\overline{BLK2}$

PIN #	TYPE
12	$\overline{BLK3}$
13	$\overline{BLK5}$
14	$\overline{RAM1}$
15	$\overline{RAM2}$
16	$\overline{RAM3}$
17	VR/W
18	$\overline{CR/W}$
19	$\overline{IRQ}$
20	NC
21	+5V
22	GND

PIN #	TYPE
A	GND
B	CA0
C	CA1
D	CA2
E	CA3
F	CA4
H	CA5
J	CA6
K	CA7
L	CA8
M	CA9

PIN #	TYPE
N	CA10
P	CA11
R	CA12
S	CA13
T	I/02
U	I/03
V	$\overline{S02}$
W	$\overline{NMI}$
X	$\overline{RESET}$
Y	NC
Z	GND

Figure 8

Multi-user

A Column by E.G. Brooner

CP/NET is Digital Research's answer to the networking problem. It is in some ways not a complete network (in fact, it is only the software for one), but it is probably the cheapest way for a clever computer user to "get into" networking. CP/NET can be thought of as an extension of CP/M and MP/M. It takes advantage of the existing features in these very popular operating systems. Using CP/NET is just like using either of its parent systems—all of the same commands are used along with a few new ones to take care of network communication tasks. Basing the network software on what is already there greatly simplifies the entire process.

The latest brochure listed the cost of the CP/NET software at around $200 for each "workstation," which can be almost any micro equipment running CP/M. Actually, one station has to be MP/M equipped—there is no way around that restriction. And if only one station is so equipped, you really have something similar to a multiprocessor or a very good timesharing system. CP/NET becomes more "network-like" as more MP/M workstations are added.

In talking about other milti-user systems, especially networks, we have made mention of the centrally located disk and printer peripherals that can be accessed by the secondary users. In network terminology these "intelligent" devices are referred to as *servers*; i.e., disk servers and print servers. A server is usually a dedicated microcomputer handling the communication to and from the peripherals, or between the peripherals and the workstations.

In CP/NET, the server function is performed by part of the additional network software. The MP/M equipped micro, which must be part of any CP/NET, acts as a server for the rest of the network in addition to performing any other functions assigned to it. The MP/M equipped micro can accomodate up to a total of 16 users. If they are terminals, you have a time sharing system; if they are micros with their own processors, the system can be a network.

Let's take a look, first, at MP/M. A time sharing system would consist of one micro running MP/M and some number of terminals sharing its one CPU and its memory, as well as the peripherals. You might find that as you add more than the first two or three terminals, the speed of each user begins to suffer, especially if all of them are quite busy.

If you replace the terminals with individual complete micros, each running CP/M, the situation can be somewhat better. It will not be any better when they are working through or with the main computer, but if they were to disconnect themselves temporarily, they would have the ability to function independently. This is what happens in the simplest possible configuration of CP/NET. The replacement of terminals with complete computers is basically the dividing point between time sharing and networking.

At this point the CP/NET resembles its more sophisticated brethren in that the users are capable of working by themselves when not accessing the central peripherals. In an ordinary MP/M system, the additional users have to share the same CPU and memory system as well as the peripherals.

A CP/NET with only one shared MP/M server will display many desirable network features and lack some others such as a high speed communication system (communication is by way of ordinary I/O ports) or a sophisticated error-detection and correction capability. It can be favorably compared to a multi-processor system, since the users operate independently when not accessing the host system's facilities.

More network features appear when more than one of the workstations is equipped with an MP/M operating system rather than ordinary CP/M. Two or more such installations can access each others' peripherals, while communication between one MP/M and one CP/M station is more of a one-way operation. However, the communication in either case is at the inherently slow speed of an I/O serial port, and you might have to write or alter part of the software to have the kind of communication you really want.

How it Works

Knowing CP/NET's terminology is important in understanding the system. Any MP/M equipped machine (or machines) rate the title of master and the CP/M workstations are called *slaves*. This fairly well describes their relationship as far as sharing facilities is concerned.

Essential elements of any CP/M system are the BIOS (I/O system) and BDOS (disk operating system.) When we graduate to MP/M we find that the BIOS is replaced by an XIOS (eXtended I/O) to handle multiple users, and that the BDOS is augmented by an "eXtended" XBDOS. This is the part that handles the time sharing chores. Figure 1 shows the relationship of these elements to system memory and the rest of the MP/M system. The configuration shown here would permit several terminals to share the MP/M-equipped computer. An MP/M system would normally share a single computer with several terminals.

CP/NET allows the slave users to be complete computers rather than simply terminals—thus, they are spared from having to all share the same CPU. The CP/NET operating system is a logical extension of the elements just discussed. Figure 2 shows the added elements and their relative position in the CP/M system. One of these new elements is

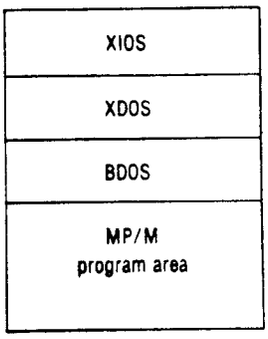


Figure 1

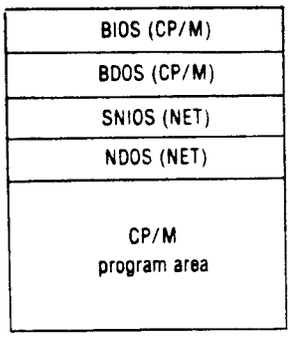


Figure 2

the NDOS, or network system; another, the SNIOS is the slave I/O software.

Notice now that the application program communicates directly with the NDOS; this is shown in flowchart form in Figure 3. The NDOS "decides" whether a particualr system call should be routed to the resident BDOS (as it ordinarily would be) or to the network, via the SNIOS. If the user has addressed one of his local peripherals, the I/O is handled by his internal CP/M system. If he has addressed a remote device, the I/O is routed to the network and thus to its ultimate destination. The slave user operates independently of the MP/M master except when accessing its peripherals.

Using CP/NET

If you're familiar with CP/M you already know about the software linkage of "physical" and "logical" devices; these include the CON: or console, and the LST: or printer, among others. You also have available, in CP/M, several disk drive assignments. These various devices can be reassigned (within limits) by commands from the console.

CP/NET extends this capability. In CP/NET you can reassign some of the master's devices in a similar way. For example, NETWORK H: = C:(01) assigns disk drive H: of the master for use as drive C: by user number 1. Drive H: might be a single floppy or a segment of a shared hard disk. Similarly, a printer driven (served) by the master can be assigned to be the LST: device of one or more users. Following these assignments the user simply calls for drive C: or LST: just as he would if they were a physical part of his own installation. In a moment we'll look at how this is actually handled by the software.

The access of a master facility may not be instantaneous, because of the sharing that might be taking place. These delays, which hopefully will not be excessive, are handled by a "queuing" arrangement in the master's network software. User requests line up and wait their turn, if necessary.

Implementation

There is some software modification to be made whenever a new CP/NET is placed in service. User and device assignments have to be made, to name the most important task. These assignments are entered into the software and become part of *status tables*. These tables have to be accessed by the individual user operating systems, to guide the routing of communication from one point to another.

Another important software task is the automatic

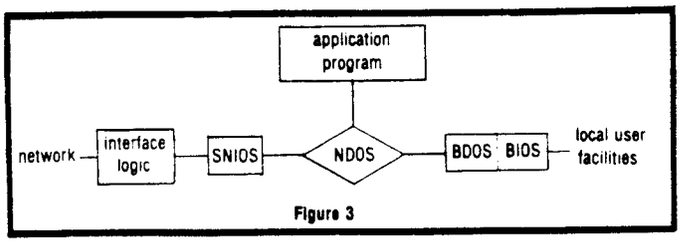


Figure 3

construction of "messages," within which the actual data or command (being transmitted from-here-to-there) is only one section. The source and destination addresses and some other control information are also appended. Figure 4 outlines a typical CP/NET message. Remember that you don't have to put this message together—the system does it for you. The real "data," which may be just a CP/M command line, is all that the user has to provide.

You might also wish to have some sort of customized hardware interface at each machine to permit the use of a communication protocol other than what is a part of your "normal" I/O ports. None of this should be very terrifying to experienced persons, but neither can the installation be considered just a "plug it in, turn it on" kind of operation.

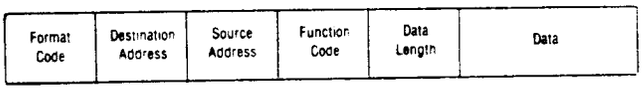


Figure 4

Evaluation

I must confess to having never used either MP/M or CP/NET, so this report is based on information released by Digital Research as well as reports found in other publications. It has been reported by others that implementing MP/M, either as a timesharing system or as part of CP/NET, is not a trivial task.

Based on my own experience with CP/M, and the general reputation of Digital's products, I would expect the system to be all that is claimed, and perhaps more. One expectations would be that current revisions might differ from, and be superior to, the original release. The price is attractive. I would expect that any CP/M compatible software would work with either MP/M or the network, which could be a valuable asset if you are already using such software.

My best guess is that "network" is stretching the term a bit if you compare this system to some of the more expensive ones such as Ethernet, but that it is a worthwhile compromise for the small-budget user of desktop computers. Because of rapid developments in networking technology, too, some of my source material might be obsolete by the time this column is printed. To be on the safe side, if you are interested, please research the matter in detail and make your own evaluation of whatever version is currently available. ■

For more information on MP/M and CP/NET, contact Digital Research Inc., Pacific Grove, CA 93950.

Build a High-Resolution S-100 Graphics Board

Part Three: Construction

by Lance Rose, Technical Editor

In the first two parts of this series, we described what is needed to display a video image, and discussed the present circuit in depth—now it's time to get down to the actual construction details.

Figure 1 shows a photograph of the front side of the graphics board. Figure 2 shows a view of the back side which should help suggest some wire routings. These are not all that critical and if you prefer a different routing, go ahead and use it. Figure 3 is a drawing of the front side of the circuit board identifying each of the ICs on it along with the pin #1 locations.

The first obvious thing is that there are plenty of chips to find a place for, so there is no wasted space. Although the layout shown here is not the only possible one, I did experiment with several other variations and this one came out on top. The rationale behind the layout is as follows: The state ROMs and clock oscillator are all located near each other to minimize long connections. Most of the small LSTTL chips are also placed to minimize long connections and reduce clutter. The crystal is mounted near the top of the board to facilitate changing. Although this probably won't be a factor for you, during development I tried a couple of different crystals and wanted to be able to access it easily. Also, the crystal is mounted in a piece of socket cut from a DIP wire-wrap socket. This can be done with a small hacksaw or coping saw. Alternatively, you can solder it in if you're sure you won't have to change it. Personally, I am a fan of flexibility.

The DIP switch for address selection is also at the top of the board where it can be reached when the board is in the card cage. Once again, if you know you'll be setting the switches once and never again, you can move it down. You can even go further and eliminate the switch and wire wrap the inputs to the open-collector NOR gates to either + 5V or ground. It saves one IC position but makes changing the address a bit of work.

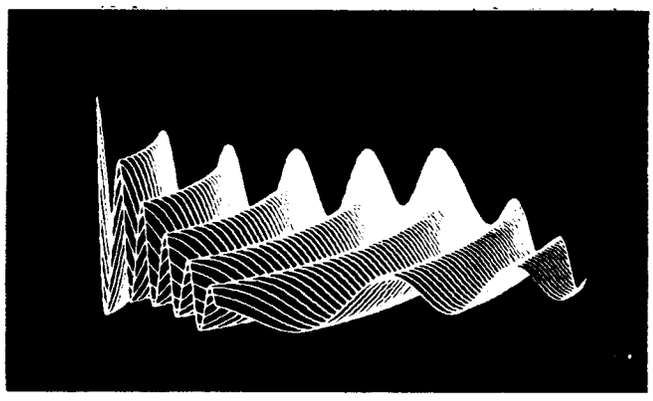
All the bus interface chips are mounted near the bottom of the board (near the bus, naturally). The address decoders and some of the other chips are in something of a compromise location. Since the RAM chips take up a lot of space, choosing a layout for them took quite a while. I tried layouts with the 24-pin sockets in a horizontal position but they didn't work out as efficiently.

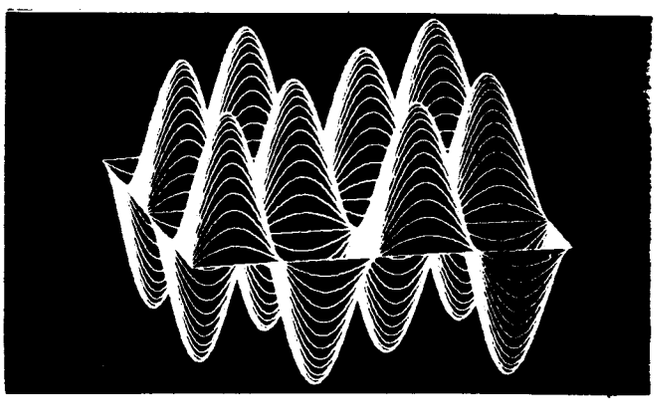
If you have had some experience in board layout or just want to strike out on your own, by all means do so. There may be other layouts that are a hair better but I think most of you will be quite satisfied with this one.

You can see that the TO-3 type regulator takes up some room, but I'm a believer in adequate power supplies. I wouldn't feel good about using only a single TO-220 5-volt regulator even though on paper it has the same current capacity as the larger metal one. And two of the TO-220s take up as much space as one of the TO-3s when you include heat sinks for them, so....

The complete parts lists for the board is found in Table 1. As I stated in Part I of this series, you should be able to buy all the parts for around $200, or even a bit less since the price of 6116 RAM chips has come down a little since I bought mine. If you were to try using soldertail sockets instead of wire-wrap and solder all the connections, the price would be even lower. However, being conservative, I wouldn't recommend that approach.

The basis of any layout you choose is, of course, the prototype board itself. There are a number of these to choose from. Some have pre-etched traces for power and ground, some have one pre-etched pad per hole, some have only an area for a voltage regulator (usually a TO-220) and some have nothing at all but the holes. The problem with any of those that have traces or pads laid out on them is that most of them assume you'll be using only 14 and 16-pin IC packages. Designs with a lot of 24-pin sockets don't seem to be able to take as much advantage of these pre-etched prototype boards. The good thing about them is that they usually have heavy traces for running power and ground





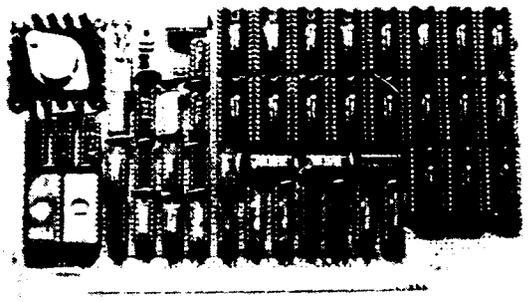


Figure 1

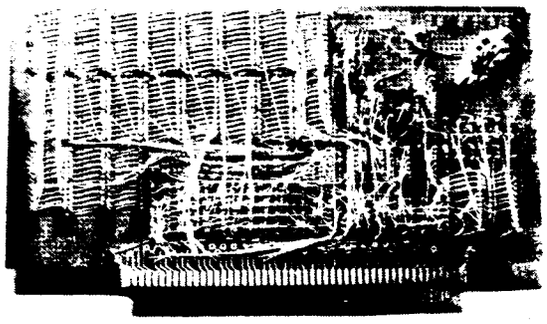


Figure 2

connections.

I normally use the completely blank boards, not just because they are the least expensive but because they offer the most flexibility in layout. Using one of these does require that you pay due attention to providing adequate power and ground "busses". A single strand of 30 gauge wire-wrap wire is a bit marginal in my experience. All power and ground leads should be doubled if at all possible. Alternatively, you can use some heavier wire for the power and ground leads and connect all the sockets to these busses with short lengths of 30-gauge wire. This approach is probably the best but requires more soldering, at least until you get the power and ground leads laid out. If you choose to distribute power and ground with wire-wrap wire, try to lay them out in a grid pattern to approximate a ground plane.

Another thing to watch is bypass capacitors. Be sure to use enough here. There are a lot of high-frequency signals running around, and one bypass cap per chip is not too much. This can be stretched a little in the RAM portion of the board since CMOS is more tolerant of noise. The RAM chips are crammed together pretty tightly and you might not be able to fit one in for each chip.

Before beginning the actual construction, let me warn you about the 24-pin sockets. Some brands are sized such that they can be placed side-by-side on a board with no blank rows of holes between them. Others require a small gap. You'll have to obtain the former kind in order to get everything to fit on this board. I got mine from Jameco, but I'm sure they are available from other sources as well. These particular ones have a "CA" stamped on the top of the socket. I don't know if this is the brand name or not, but if you order sockets remotely, be sure to ask for ones that will fit side-by-side. This only applies to the 24-pin sockets. With the others, there is always at least one row's spacing between them.

As a general procedure, I think it's best to begin by mounting the voltage regulator and tantalum electrolytics along with the connections to the S-100 8-volt and ground fingers. At the same time, you can do any other mechanical work necessary with a drill or file. The RCA phono jack shown here is mounted on a small L-bracket that has been drilled out to size. If you use a different type of connecter, do the appropriate work here so that when you get the sockets on the board you won't have to be drilling, sawing, filing and sanding around them.

With a hefty TO-3 regulator you can afford to use one of the low-profile heat sinks. This will make it easier to get the finished board into your card cage. Once this is done, you can plug the board in and check out the regulator for proper voltage.

Following this, my own habit is to place all the sockets on the board and fasten them down by connecting the power and ground leads to them. At the same time I solder in the bypass caps. Powering up at this point will detect any shorted caps and it will be easier to find them without all the signal leads running around the board.

Perhaps I ought to say a word or two about color coding. I've seen projects built with only a single color of wire-wrap wire but debugging has to be more difficult that way. Even if all you have nearby is a Radio Shack store, you can still get 4 different colors - red, blue, yellow and white. If you're willing to mail order, you can get about another 4 or 5 colors. That may be going overboard. I usually use red for power supply, blue for ground, yellow for address lines and white for data and signal lines (actually five colors would be just right). This is just a personal preference on my part and if you would rather use all one color, by all means go ahead. As long as you can find your way around the board, that's all that matters.

After all the sockets are located on the board you can go ahead and start connecting the signal leads. The suggested order of wiring is as follows.

First wire the oscillator circuit, which includes the 7404, the crystal, and the necessary resistors and capacitor. You can then power up and check for oscillation. You should be able to detect this with a reasonably good scope or logic probe. Once oscillation is verified, the next section to wire is the state machine. This includes the 2732 EPROMs, all the 74LS193 counters, and two of the 74LS393s—U8 and U9, and the 74LS174.

If you are going to program your own EPROMs, the program listing in Figure 5 is the one to use. It requires the use of either Digital Research's MAC® macro assembler or a recent version of Microsoft's M80 assembler. If you don't have either one of these or can't get access to one, you can either rewrite the program in BASIC or order a set of ROMs from *The Computer Journal*. The price is $25 per pair preprogrammed. If you have the facilities for programming and would like a machine-readable copy of the program, we

can provide one on 8" single density CP/M for $15. This also includes the test program and sample use program whose listings appear in Figures 6 and 7 here. We may also be able to provide some 5.25" formats but you'll have to check with us and see if it's one that we can make.

Once this is done and you have either programmed your own EPROMs or obtained them from us, you can plug the board in and look at the outputs of the 74LS174 to see if the timing is about what you would expect. This would mean blanking and sync outputs at approximately the horizontal sweep frequency and RAM counter reset (pin 15) once every 1/60 second. If these outputs look about right, the state machine is probably working fine.

Next go ahead and wire up the 74LS165 shift register and any necessary gates associated with it. Wire the transistor and the rest of the output circuit as well. Although I used a 2N5450 transistor in my version, it was mostly because I happened to have a bunch of them in my junk box. Almost any small-signal transistor with a decent gain and bandwidth should work, so try using what you have handy.

You can now connect the board to your video monitor for further troubleshooting. With the power on, the video screen should display a uniform bright raster with no tearing or lack of synchronization. If you can't get the image to lock in with the monitor's horizontal and vertical hold controls, something is out of whack. Try looking at the composite video output with an oscilloscope. The waveform should appear much as in Figure 2 in Part I of this series. Check both voltage levels and timing. The signal level can be off quite a bit and still work by adjusting the brightness and contrast accordingly, but the timing should be very close. If it isn't, stop and check all connections in the timing circuit U3-U5 as well as the state ROM address counters U8 and U9. Also check the dot clock and byte clock outputs for proper frequency.

Once you get a display that will lock in on the screen, go ahead and, using a clip lead, ground each of the parallel data inputs to the 74LS165 in turn. This should result in a series of thin, dark vertical lines in the bright field. Grounding two adjacent data inputs will produce wider dark lines. If this is what you get, so far so good.

Next try wiring up the local RAM address counters U16 and U17, buffers U18 and U19 and any necessary gates that go with them. Wire up at least one of the 6116 sockets, preferably one adjoining U18 or U19. This will let you display information from the video RAM without having to have all the sockets wired. You should also wire up the address decoders U27, U28 and U29a and U20d, but only connect the chip select line for the one RAM socket you have in at the present. You'll also need U14b and U12d (you can leave pin 9 on the latter open for now).

On powering up you should get an all-white (green) display except for a small area where the single 6116 is. In that area the screen should show random dots both on and off. In all likelihood, the pattern will be somewhat regular and probably look like alternating white and black bars about 8 dots wide each. This is simply due to the power-up characteristics of the 6116 and has no other significance. There will be a few random areas anyway. If you don't get alternating bands, don't worry. The important thing is to identify at least a few features attributable to random bits.

If things are working here, you're over the hump. Go ahead and make the connections for all the bus interface logic. This means wire everything else except the remaining

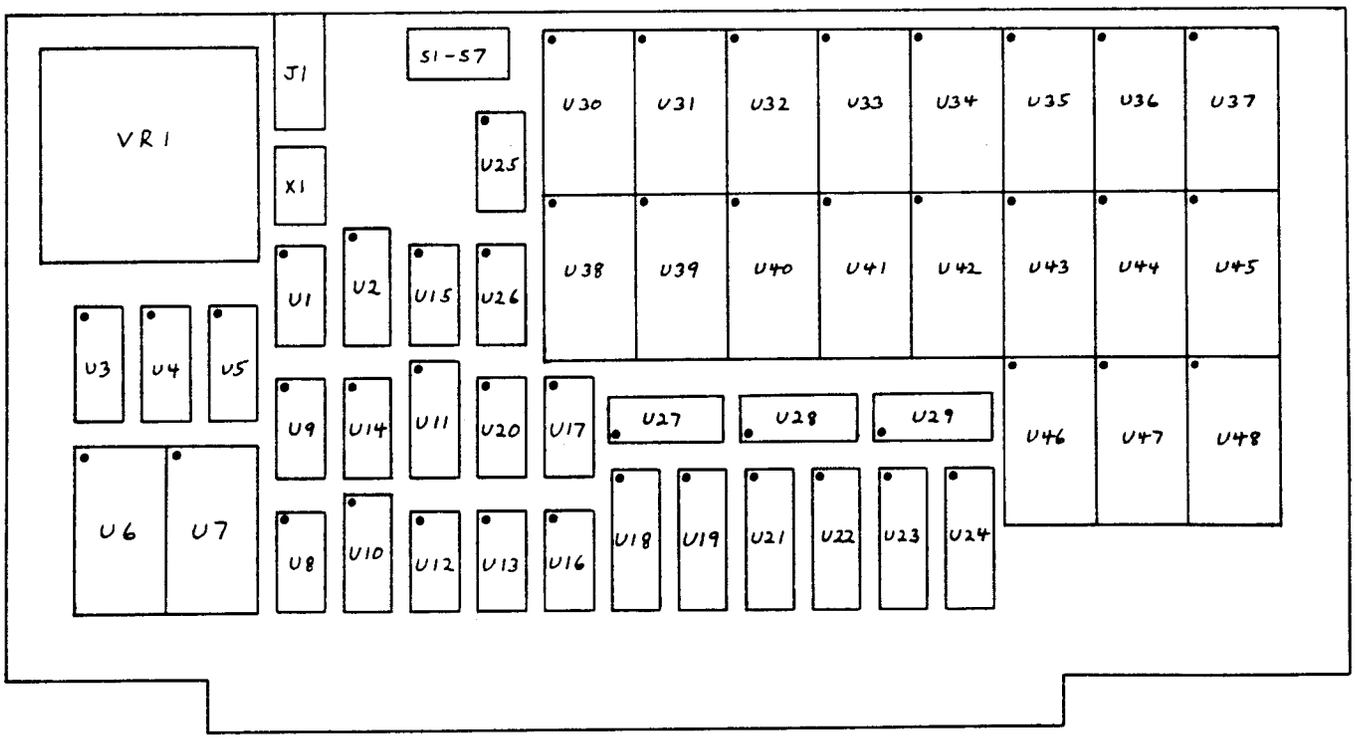


Figure 3: Component layout for board.

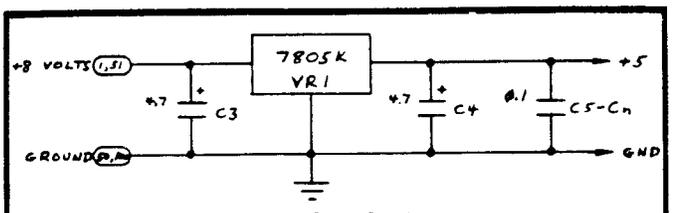


Figure 4:Power Supply

RAM chips. Also, put in the DIP switch if you're going to use one. If you like you can place it in a socket, as I did. Double check all but the remaining RAM connections and you're ready for an actual test.

Plug the board into your bus and power up. Of course if your machine won't operate at this point, you have a problem. Most likely you made a connection to a wrong bus pin or accidentally grounded a bus line or something of that nature. I have found that about 90% of all errors in prototype circuits can be found by visual inspection, checking the wiring against the schematic. Another thing you should be checking on a regular basis here is the output from the regulator. If it is more than about a quarter of a volt off, you probably should try another regulator.

After you get the board in and the machine running, the easiest way to check the board out is from BASIC. The general procedure for writing a byte to the video RAM is as follows:(Assume the board is addressed at locations 100, 101, 102 decimal)

(1) Output the low byte of the desired video RAM address to port 100.

(2) Output the high byte of the desired video RAM address to port 101.

(3) Output the desired data byte to be placed in video RAM to port 102.

You should be able to tell what addresses are active by which RAM socket you wired up. Outputting a byte of 0 will make a dark line 8 dots wide appear at that location. A byte of 255 will make a bright line of the same size. Intermediate values will turn on or off all other possible combinations. If it doesn't appear to work right away here, double check to see that the single RAM chip you have wired up is at the address you think it is. If you're not sure, you can always go ahead and wire up all the rest of the RAM sockets and install the chips. However, that's a lot of connections and if you know you already have a working board and just have to mechanically run the wires, it seems to be psychologically easier.

If you just can't get any effect from trying to write into the video RAM, you'll need a logic probe to do some tracing. Check for the WE* pulse on pin 21 of the RAM during the write operation. If you think it will help, you can disable the arbitration circuitry and allow constant access to the video RAM from the bus by temporarily removing the wires from pins 10 and 12 of U10 and grounding them. This should constantly select the bus address latches as the address generators for the RAM chips. Just don't forget to change this back after you find the problem. Make sure that U18 and U19 are not being enabled at the same time as U21 and U22. Check for the usual things like bent-under pins on

C1	220 pf ceramic disk
C2	470 pf ceramic disk
C3,C4	4.7 uf/25V tantalum electrolytic
C5-Cn	0.1 uf ceramic disk (bypass)
J1	RCA phono jack
Q1	2N5450 or equivalent
R1,R2	470 ohm, 1/4 watt
R3	100 ohm, 1/4 watt
R4	220 ohm, 1/4 watt
R5,R6	150 ohm, 1/4 watt
R7-R14	2.2 kohm, 1/4 watt
S1-S7	7-position DIP switch
U1	7404 hex inverter
U2-U5	74LS193 presettable binary counter
U6,U7	2732 4Kx8 EPROM
U8,U9,U16,U17	74LS393 dual binary counter
U10	74LS174 hex D flip-flop
U11	74LS165 shift register
U12	74LS02 quad NOR gate
U13	74LS32 quad OR gate
U14	74LS08 quad AND gate
U15	7406 hex driver, open collector
U18,U19,U23,U24	74LS244 octal buffer
U20	74LS00 quad NAND gate
U21,U22	74LS374 Octal D flip-flop, 3-state
U25,U26	74LS266 quad exclusive NOR, open collector
U27,U28	74LS138 one of eight decoder
U29	74LS139 dual one of four decoder
U30-U48	6116-4 2Kx8 CMOS RAM
VR1	7805K 5 volt regulator, TO-3
X1	16 MHz crystal

Table 1: Parts List

chips, bad chips (very unlikely) or wiring errors (much more likely). It's hard to give specific instructions here since there are so many variations. As I said above, most of my own errors have been due to accidentally leaving a connection out or making it to the wrong pin. When you're staring at about a thousand pins upside down and backwards, it's not too hard to do.

After debugging the write portion, try reading the byte back. You can skip steps 1 and 2 above if you're accessing the same location. For step 3, input a byte from port 102 and see if it's the same as what you output. If not, use similar debugging techniques on the read portion of the bus interface logic. Note here that you can do either 1 or 2 above without the other. You only need to change the portion of the address that is different from the last address accessed.

The final step, once the board is working with a single chip, is to add the other RAMs. It just takes some time. With all the RAM chips plugged in, you're ready for final checkout. A simple test program in Microsoft BASIC is shown in Figure 6. It writes random bytes to random screen locations and then attempts to read them back. Make sure the DIP switch on the board is set for 100 decimal (64H) or else change the port numbers in the test program to coincide with what you have the switches set for. If an error is detected, the program reports it. When the program is running the screen should display a speckled random pattern that changes slowly, one speckle at a time. No hash should be seen in the display at all. If there is hash, suspect the interlock circuit that is supposed to keep the processor waiting during scan lines. Also check the wait state circuitry on the CPU board to make sure it's functioning properly. Most newer memories don't require any wait states so yours

Figure 5: Program for T.V. signal ROMs.

```
;       2732 ROM program to generate TV waveforms
;       Used for I/O-mapped graphics board
;       Version of 2/3/84
;
;
BLANK   EQU     00000100B       ;Generate blanking level
SYNC    EQU     00001000B       ;Generate sync level
COUNT   EQU     00010000B       ;Count up RAM pointer with dot clock
WAIT    EQU     00100000B       ;Generate wait state
DISABL  EQU     01000000B       ;Maintain wait state and disable MWRITE
LOCL    EQU     01100000B       ;Keep wait, MWRITE disabled, use RAM pointer
RESET   EQU     10000000B       ;Reset RAM pointer
TOP     EQU     27              ;Scan lines in blanked top of frame
VERT    EQU     480             ;Scan lines in view area
LINE    EQU     1016            ;Dots in line
HSYNC   EQU     80              ;Dots in horizontal sync pulse
LEFT    SET     174             ;Dots in left border
HORIZ   SET     640             ;Dots in view area
RIGHT   SET     122             ;Dots in right border
EQPLSE  EQU     48              ;Dots in equalizing pulse
FINISH  EQU     10              ;Dots to finish current access
WRTOFF  EQU     10              ;Dots to turn off MWRITE
SWITCH  EQU     10              ;Dots to change multiplexer inputs
;
;
;       Low byte ROM
;
        REPT    TOP/2           ;Blanked lines at top of first field
        DB      HSYNC-1
        DB      LEFT-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      RIGHT-1
        ENDM
;
        REPT    VERT/2          ;Visible lines in first field
        DB      HSYNC-1
        DB      LEFT-FINISH-WRTOFF-SWITCH-1
        DB      FINISH-1
        DB      WRTOFF-1
        DB      SWITCH-1
        DB      HORIZ-1 AND 0FFH
        DB      SWITCH-1
        DB      RIGHT-SWITCH-1
        ENDM
;
        DB      HSYNC-1         ;Blanked half line
        DB      LINE/2-HSYNC-1 AND 0FFH
;
        REPT    6               ;Equalizing pulses
        DB      EQPLSE-1
        DB      LINE/2-EQPLSE-1 AND 0FFH
        ENDM
;
        REPT    6               ;Vertical retrace
        DB      LINE/2-HSYNC-1 AND 0FFH
        DB      HSYNC-1
        ENDM
;
        REPT    5               ;Equalizing pulses
        DB      EQPLSE-1
        DB      LINE/2-EQPLSE-1 AND 0FFH
        ENDM
        DB      EQPLSE-1        ;Full equalizing line
        DB      LINE-EQPLSE-1 AND 0FFH
;
        REPT    TOP/2           ;Blanked lines at top of second field
        DB      HSYNC-1
        DB      LEFT-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      HORIZ/4-1
        DB      RIGHT-1
        ENDM
;
        REPT    VERT/2          ;Visible lines in second field
        DB      HSYNC-1
        DB      LEFT-FINISH-WRTOFF-SWITCH-1
        DB      FINISH-1
        DB      WRTOFF-1
        DB      SWITCH-1
        DB      HORIZ-1 AND 0FFH
        DB      SWITCH-1
        DB      RIGHT-SWITCH-1
        ENDM
;
        REPT    6               ;Equalizing pulses
        DB      EQPLSE-1
        DB      LINE/2-EQPLSE-1 AND 0FFH
        ENDM
;
        REPT    6               ;Vertical retrace
        DB      LINE/2-HSYNC-1 AND 0FFH
        DB      HSYNC-1
        ENDM
;
        REPT    5               ;Equalizing pulses
        DB      EQPLSE-1
        DB      LINE/2-EQPLSE-1 AND 0FFH
        ENDM
        DB      EQPLSE-1        ;Half equalizing line
        DB      LINE/2-EQPLSE-1 AND 0FFH
;
;
;       High byte ROM
;
        REPT    TOP/2           ;Blanked lines at top of first field
        DB      BLANK+SYNC
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        ENDM
;
        DB      BLANK+SYNC      ;Visible lines in first field
        DB      BLANK
        DB      BLANK+WAIT
        DB      BLANK+DISABL
        DB      BLANK+LOCL
        DB      LOCL+(HORIZ-1)/256
        DB      BLANK+DISABL
        DB      BLANK
        REPT    VERT/2-1
        DB      BLANK+SYNC+COUNT
        DB      BLANK
        DB      BLANK+WAIT
        DB      BLANK+DISABL
        DB      BLANK+LOCL
        DB      LOCL+(HORIZ-1)/256
        DB      BLANK+DISABL
        DB      BLANK
        ENDM
;
        DB      BLANK+SYNC      ;Blanked half line
        DB      BLANK+(LINE/2-HSYNC-1)/256
;
        REPT    6               ;Equalizing pulses
        DB      BLANK+SYNC
        DB      BLANK+(LINE/2-EQPLSE-1)/256
        ENDM
;
        REPT    6               ;Vertical retrace
        DB      BLANK+SYNC+(LINE/2-HSYNC-1)/256
        DB      BLANK
        ENDM
;
        REPT    5               ;Equalizing pulses
        DB      BLANK+SYNC
        DB      BLANK+(LINE/2-EQPLSE-1)/256
        ENDM
        DB      BLANK+SYNC      ;Full equalizing line
        DB      BLANK+RESET+(LINE-EQPLSE-1)/256
;
        REPT    TOP/2           ;Blanked lines at top of second field
        DB      BLANK+SYNC      ;(Includes blanked half line)
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        DB      BLANK
        ENDM
;
        REPT    VERT/2          ;Visible lines in second field
        DB      BLANK+SYNC+COUNT
        DB      BLANK
        DB      BLANK+WAIT
        DB      BLANK+DISABL
        DB      BLANK+LOCL
        DB      LOCL+(HORIZ-1)/256
        DB      BLANK+DISABL
        DB      BLANK
        ENDM
;
        REPT    6               ;Equalizing pulses
        DB      BLANK+SYNC
        DB      BLANK+(LINE/2-EQPLSE-1)/256
        ENDM
;
        REPT    6               ;Vertical retrace
        DB      BLANK+SYNC+(LINE/2-HSYNC-1)/256
        DB      BLANK
        ENDM
;
        REPT    5               ;Equalizing pulses
        DB      BLANK+SYNC
        DB      BLANK+(LINE/2-EQPLSE-1)/256
        ENDM
        DB      BLANK+SYNC      ;Half equalizing line
        DB      BLANK+RESET+(LINE/2-EQPLSE-1)/256
;
;
        END
```

may not be working and you won't have known it. If every byte comes back an error, something is wrong in the read portion of the bus interface logic. Check wiring. If only occasional errors occur, check for good grounding and power connections. Also, make sure there are enough 0.1uf bypass caps.

When the board passes the GTEST program, you can then try out the sample plotting program included here in Figure 7, also in Microsoft BASIC (but easily convertible to another BASIC). A screen menu lets you choose a variety of functions to test the board. The scale is assumed to be linear in both directions and the left, right, upper and lower limits can be changed with the "Set Limits" option. After that you can set an individual point, draw a straight line, an arc or a sine curve (arguments in degrees).

The program is designed mostly for fooling around and as an example of how to use the facilities of the graphics board. The painful slowness of BASIC is amply evident here. Any truly useful graphics programs simply must be written in assembly language to operate with any kind of reasonable speed. Compiled BASIC or FORTRAN are improvements but still not as good as assembly. If you aren't doing at least some assembly language programming yet, this may be the impetus you need.

Figure 6: BASIC test program.

```
1000 A=INT(38399!*RND)
1010 OUT 100,A-INT(A/256)*256
1020 OUT 101,INT(A/256)
1030 D%=256*RND MOD 256
1040 OUT 102,D%:E%=INP(102)
1050 IF D%=E% THEN 1000
1060 A$=HEX$(A)
1070 IF LEN(A$)<4 THEN A$="0"+A$:GOTO 1070
1080 D$=HEX$(D%):IF LEN(D$)=1 THEN D$="0"+D$
1090 E$=HEX$(E%):IF LEN(E$)=1 THEN E$="0"+E$
1100 PRINT "ERROR AT "A$;", ";D$;" READ AS ";E$
1110 GOTO 1000
```

Figure 7: BASIC program to draw some shapes with the board.

```
1000 REM  DEFINE I/O PORT ADDRESSES
1010 P0=100:P1=P0+1:P2=P0+2
1020 REM  SET DEFAULT LIMITS FOR SCREEN
1030 X1=-1.33333:X9=1.33333:Y1=-1:Y9=1
1040 REM  CLEAR SCREEN, DISPLAY MENU AND INPUT CHOICE
1050 PRINT CHR$(27);"* (1) CLEAR SCREEN":PRINT " (2) REVERSE VIDEO"
1060 PRINT " (3) SET POINT":PRINT " (4) DRAW LINE"
1070 PRINT " (5) DRAW ARC":PRINT " (6) SET LIMITS"
1080 PRINT " (7) LOAD SCREEN":PRINT " (8) SAVE SCREEN"
1090 PRINT " (9) DRAW SINE CURVE":PRINT "(10) EXIT":PRINT:INPUT O:PRINT
1100 REM  EXECUTE PROPER SUBROUTINE AND THEN RETURN TO MENU
1110 ON O GOSUB 1140,1170,1200,1270,1370,1450,1480,1550,1610,1630
1120 GOTO 1050
1130 REM  CLEAR SCREEN
1140 FOR H%=0 TO 149:OUT P1,H%:FOR L%=0 TO 255
1150 OUT P0,L%:OUT P2,0:NEXT L%:NEXT H%:RETURN
1160 REM  REVERSE VIDEO TOGGLE
1170 FOR H%=0 TO 149:OUT P1,H%:FOR L%=0 TO 255:OUT P0,L%
1180 OUT P2,NOT INP(P2) AND 255:NEXT L%:NEXT H%:RETURN
1190 REM  SET A POINT ON THE SCREEN
1200 INPUT "COORDINATES(X,Y): ",X,Y
1210 IF X<X1 OR X>X9 OR Y<Y1 OR Y>Y9 THEN RETURN
1220 H%=(X-X1)/(X9-X1)*639
1230 V%=(Y-Y9,/(Y1-Y9)*479:M=V%*80!+H%\8:B=2^(H% MOD 8)
1240 OUT P0,M-INT(M/256)*256:OUT P1,INT(M/256)
1250 OUT P2,INP(P2) OR B:RETURN
1260 REM  DRAW LINE
1270 INPUT "STARTING POINT(X,Y): ",X2,Y2
1280 INPUT "ENDING POINT(X,Y): ",X8,Y8
1290 DX=X8-X2:DY=Y8-Y2:IF DX=0 AND DY=0 THEN X=X2:Y=Y2:GOTO 1210
1300 IF DX<>0 THEN S=DY/DX ELSE S=0:GOTO 1340
1310 IF S*(X9-X1)*479/(Y9-Y1) 639) THEN S=(Y8-Y1)/479:GOTO 1340
1320 FOR X=X2 TO X8 STEP SGN(X8-X2)*(X9-X1)/128
1330 Y=Y2+(X-X2)*S:GOSUB 1210:NEXT X:RETURN
1340 FOR Y=Y2 TO Y8 STEP SGN(Y8-Y2)*(Y9-Y1)/958
1350 X=X2+(Y-Y2)*S:GOSUB 1210:NEXT Y:RETURN
1360 REM  DRAW ARC
1370 INPUT "CENTER(X,Y): ",XC,YC
1380 INPUT "RADIUS: ",R
1390 INPUT "DEGREE LIMITS(BEGIN,END): ",A1,A2
1400 A1=A1*.0174533:A2=A2*.0174533
1410 S=(X8-X1)/639:IF (Y9-Y1)/479>S THEN S=(Y9-Y1)/479
1420 S=S/2*R):FOR A=A1 TO A2 STEP SGN(A2-A1)*S
1430 X=R*COS(A+XC):Y=R*SIN(A)+YC:GOSUB 1210:NEXT A:RETURN
1440 REM  SET LIMITS
1450 INPUT "HORIZONTAL LIMITS: ",X1,X9
1460 INPUT "VERTICAL LIMITS: ",Y1,Y9:RETURN
1470 REM  LOAD SCREEN FROM FILE
1480 INPUT "FILE NAME: ",F$:OPEN "R",1,F$:FIELD #1,128 AS D$
1490 FOR H%=0 TO 149:OUT P1,H%:GET #1
1500 FOR L%=0 TO 127:OUT P0,L%:OUT P2,ASC(MID$(D$,L%+1,1))
1510 NEXT L%:GET #1:FOR L%=128 TO 255:OUT P0,L%
1520 OUT P2,ASC(MID$(D$,L%-127,1)):NEXT L%:NEXT H%
1530 CLOSE #1:RETURN
1540 REM  SAVE SCREEN IN FILE
1550 INPUT "FILE NAME: ",F$:OPEN "R",1,F$:FIELD #1,128 AS D$
1560 FOR H%=0 TO 149:OUT P1,H%:FOR L%=1 TO 128:OUT P0,L%-1
1570 MID$ (D$,L%,1)=CHR$(INP(P2)):NEXT L%:PUT #1
1580 FOR L%=1 TO 128:OUT P0,L%+127:MID$(D$,L%,1)=CHR$(INP(P2))
1590 NEXT L%:PUT #1:NEXT H%:CLOSE:RETURN
1600 REM  DRAW SINE CURVE
1610 INPUT "ARGUMENT(BEGIN,END,STEP): ",X2,X8,DX
1620 FOR X=X2 TO X8 STEP DX:Y=SIN(X/57.2958):GOSUB 1210:NEXT X:RETURN
1630 END
```

If you have gone this far and actually built the board, you probably had some ideas of what you wanted to use it for. Some of the things I wanted to do were to plot 3-dimensional figures, plot orbital data from my astronomy programs, generate Fourier waveforms and use it to display alphanumeric data in any one of a number of character fonts. Since the board is programmable on the dot level, you can make up characters in any shape you like (APL, Chinese, Arabic, Hebrew, Greek, etc.) You can do proportional spacing. You can plot in rectangular or polar coordinates. Its uses are limited only by your imagination.

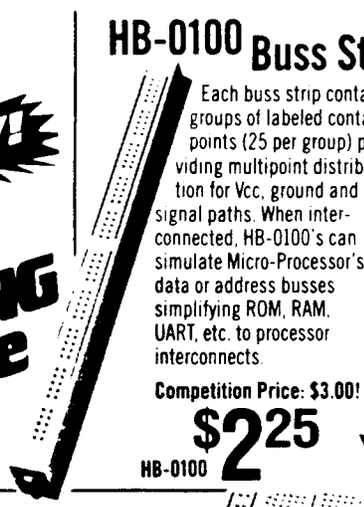
Although this concludes the series of construction articles on this board, there may be other articles on specific applications in the future if interest warrants it. We'd like to hear your views on it. If you build the board, we'd also like to hear your difficulties, if any, and your reactions to the completed product. ■

HANDY

NEW!

The complete BREADBOARDING SOURCE at the lowest prices ANYWHERE!

HB-0100 Buss Strip

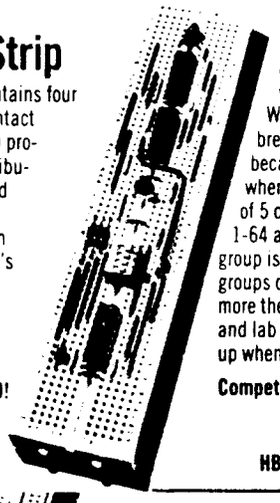


Each buss strip contains four groups of labeled contact points (25 per group) providing multipoint distribution for Vcc, ground and signal paths. When interconnected, HB-0100's can simulate Micro-Processor's data or address busses simplifying ROM, RAM, UART, etc. to processor interconnects.

Competition Price: $3.00!

HB-0100 **$2.25**

HB-1000 Socket

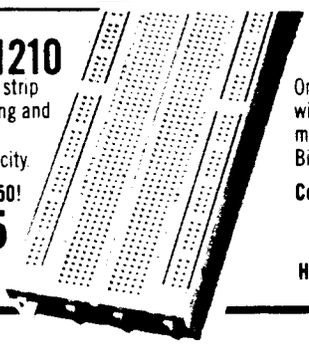


With HANDY sockets, breadboarding is easy because you always know where you're at. Each group of 5 contacts is labeled from 1-64 and each position within a group is labeled from a-j. With 64 groups of 5 contacts each you have more then enough room for infield and lab mock-ups that always come up when you least expect them.

Competition Price: $12.50!

HB-1000 **$9.95**

HB-1210

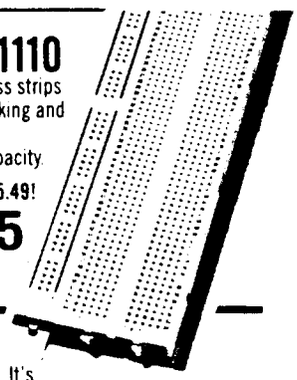


One socket and one buss strip with self-adhesive backing and metal ground plate.
Big nine 14-pin DIP capacity.

Competition Price: $18.50!

HB-1210 **$13.95**

HB-1110

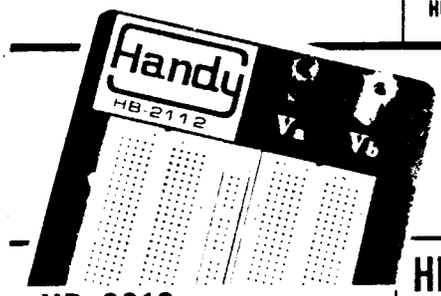


One socket and two buss strips with self-adhesive backing and metal ground plate.
Big nine 14-pin DIP capacity.

Competition Price: $15.49!

HB-1110 **$11.95**

HB-2112



A compact breadboard ideal for beginners, students, professionals and hobbyists. It's quick and easy to use and immediately becomes the most useful tool on your workbench. Includes: • 2 sockets • 1 buss strip • 2 binding posts and is mounted on an 8.7"x5.4" aluminum ground plate. 18 14-Pin DIP capacity.

HB-2112 **$24.95**

ORDER NOW!

HB-2313

Holds enough IC's to meet any quick-fix situation. Includes: • 2 sockets • 3 buss strips • 3 binding posts and is mounted on an 8.7"x5.9" aluminum ground plate. 18 14-Pin DIP capacity.

Competition Price: $51.50!

HB-2313 **$31.00**

HB-3514

Ideal for light industrial as well as secondary level educational and hobbyist experimentation. Includes: • 3 sockets • 5 buss strips • 4 binding posts and is mounted on a 9.4"x7.7" aluminum ground plate. 27 14-Pin DIP capacity.

Competition Price: $66.00!

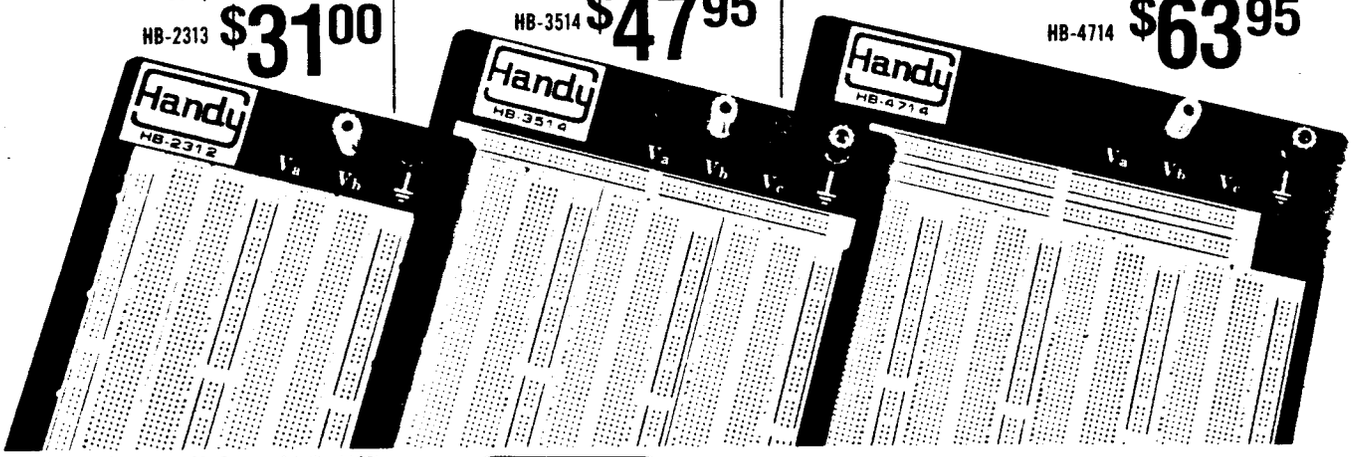
HB-3514 **$47.95**

HB-4714

Largest HANDY breadboard available today. Complete PCB or even system mock-ups can easily fit on this breadboard. This high capacity, heavy duty board is a must for industrial users where time is money ...Great for advanced educational and hobbyist applications. Includes: • 4 sockets • 7 buss strips • 4 binding posts and is mounted on a 10.3"x9.5" aluminum ground plate. Giant 36 14-Pin DIP capacity.

Competition Price: $89.00!

HB-4714 **$63.95**



Mail Orders: Please add $3 (Canada & Int'l add $5) to cover cost of shipping, handling
Charge Cards: (Min. $15) Please include Acct. No., Exp. Date, Bank No. (M/C only) and your signature
Checks: Drawn in U.S. Dollars on U.S. banks only
Conn Residents: Add 7½% Sales Tax. Sorry! No C.O.D. orders

VISA MasterCard AMERICAN EXPRESS

HANDY

a division of RSP Electronics Corp.

To order...call 1-800-34-HANDY
...charge with VISA, MasterCard or American Express.
All items off-the-shelf for Immediate Shipment!

7 Business Park Drive • P.O. Box 699 • Branford, CT 06405 • (203) 488-6603 • TWX: (910) 997-0684
Easy Link Mail Box: 62537580 • CompuServe: 71346, 1070
U.S. and Canadian Distributor inquiries welcomed.

SYSTEM INTEGRATION

Part Three: CP/M 3.0

by Bill Kibler

In the first part of this series, we discussed factors to keep in mind when choosing components for system integration. In part two we looked at how SDSystems Versafloppy II was set up to run CP/M 2.2. I hope that most readers will have at least a 2.2 system running at this stage. If you have not yet used a 2.2 system, a complete reading of this article and other books will be necessary before considering CP/M 3.0. To help the novice and weekend hacker, let's first review what a 2.2 system does.

CP/M 2.2

CP/M is the interface between the hardware of your individual system and the generic types of software you will run. This program, known as an operating system, provides standard interfaces in two directions, one to transient programs and one to fixed hardware. CP/M advanced through many variations and stabilized for some time at version 2.2. This version is intended to run on an 8080 or Z80 CPU with 64K of memory, using fixed peripherals for I/O. An assembler, a debugger, an editor, and several other utilities are provided with the purchase of CP/M. This entire operating system makes it possible to change, modify, and generate new operating systems, in addition to running purchased programs with few problems.

The ins and outs of 2.2 are well known, and for those looking for more knowledge, your local computer store has many books on the subject. 2.2's structure is extremely straightforward. The relationship between the three modules that make up the program are fairly easy to explain. The CCP is the command processor and handles communications with the operator while in the system mode (not running any generic programs). The BDOS (basic disk operating system) is the next module, and does most of the work. It works with running programs by the use of *function calls*. The function calls are separate routines that together form the BDOS and allow canned programs to operate on different hardware. The BDOS takes a function call and talks to the next module, the BIOS (basic in/out system) as many times as necessary to make the hardware perform the requested function.

2.2 has been around for so long now that other products are available that can enhance or replace all or portions of CP/M. With the books and software base available, this operating system has little equal and allows me to recommend it without question as the choice of hackers. However, like most manufacturers, DRI (Digital Research Inc.) felt that an improved version of CP/M that incorporated most of the enhancements would allow them to take back control of their product. An improved version would also present an opportunity to work out the incompatibilities between 2.2 and DRI's multiuser operating system, MP/M. The cost of memory has decreased in quantum leaps, reducing the expense of having more than 64K, and a system was needed that could make use of this cheap memory. So CP/M 3.0 was born.

CP/M 3.0

CP/M 3.0 or "PLUS," as it is sometimes called, is little different from 2.2. It is still the operating interface between the hardware and software, but with some new wrinkles. Memory can now be in multiple banks, outputs from programs can be changed or redirected, additions to the operating system can be created and tacked on, there are help programs and command line editing, and lastly, the system is supposed to be faster. What has been lost is a simple BIOS program and a lot of disk space. Although the gains may seem to outweigh the losses, closer investigation is needed.

To utilize CP/M 3.0 more fully you will need more than one bank of memory. In the non-banked mode most of the enhancements are lost,as well as part of the transient program area (TPA). Current programs require up to 56K of temporary program RAM to run, and the non-banked TPA of 50 or 51K may not work. Once the BIOS has been generated for banked operation the TPA expands to over 60K, which can be more than most 2.2 versions. The extra banks contain the larger BDOS, a copy of the CCP program (3.0 CCP is a transient program and not resident like 2.2), and data buffers. Higher operating speeds are possible for programs that use high disk access by locating the most highly and currently used sectors in the disk buffers (other banks). These buffers are assigned and managed by the BDOS and are not controlled in any way by the operator.

Inputs and outputs are normally assigned through fixed BIOS definitions or routines. In 3.0, tables of I/O types and requirements are maintained, which allow the system configuration to be changed without rewritting the BIOS. This operator freedom of choice is made at the expense of memory (banked RAM) and speed (the whole table is scanned for each output of a character). 2.2 has an I/O byte that has now been expanded to several bytes to take into consideration the different possible ways of redirecting the data flow. A new program type, the RSX module, has been added. This program technique is helpful for those who want to add some special function that the BDOS doesn't currently support. The new module is put below the BDOS and the inputs to the BDOS are redirected through it. In 2.2 this was not possible except for enhancements to the BIOS routines as is done in RAM DRIVE modifications.

The banked version allows the command line to be edited

instead of completely rewritten as is necessary with 2.2. This is made possible by larger command line buffers and the addition of more control functions to the CCP. HELP programs, although listed as a feature, are really not much more than the CPMUG or SIG/M version of HELP.COM. This disk program has help files that explain parts of the operating system's syntax usage. The help files are usually one line statements which tell what form the command must take to do the named function. These HELPs are in no way to be considered "tutorials," but are simply quick reference "cards." As for speed, the system can take longer to boot (it must load more information in different banks) but once booted, should require less disk access (the slowest part of the operation) and result in faster operations. The controlling factor, however, is still the speed of disk reading and writing,—if the information is not in a disk buffer area, it must be loaded into a buffer and then into the TPA. Depending on the type and nature of the programs, this buffering can be faster than 2.2. Also, for warm boots, the CCP is reloaded from a bank and not from the disk as is done in 2.2.

The Implementation

With the brief introduction out of the way, it is now time to "bite the bullet" and get to the "how-to" part. A word of caution is needed for those who found the previous discussions somewhat confusing: don't try to implement 3.0—buy it already done or get 2.2 first. Writing a banked BIOS and deciphering the DRI manuals is not for the novice programmer. In November of 1983, DRI changed its support policy to match that of most other big time software houses, mainly, no FREE support. They now charge a $250 yearly subscription fee to get in-depth information on their programs. Anything not covered in their manuals (and there is a lot not covered) must be explained through your own efforts or by a member of the subscription service.

The first confusing topic is the many subroutines (see the listing for what is on DRI's disk). CP/M 3.0 is composed of many macro files (these are short assembly listings) that will be linked together to form the complete program. To make things worse for those doing the SDSystems implementation, they have their own version of these same programs for their hardware. SD's hardware is the SBC 300, and unfortunately, their programs are so hardware dependent that they are useless to anyone who does not own the defined hardware. The absence of the ASM files of the FLPDRVR and MOVE routines add to the problem. What this leaves for the programmer is the original DRI files and the disk controller routines from the SD VFII manual for generating a running system. On closer examination, I have found that the DRI files, with modifications, are a good source for a running system. Time has not permitted me to fully explore the use of DRI's files directly, but those who feel lost should consider starting there. I did use the BIOSKRNL and the 2.2 BIOS routines as the disk file, and was able to make it work in a non-banked mode. The manual indicates that a 3.0 BIOS can be made from a 2.2 BIOS, but it also says that two modules are supposed to be the root modules and should not be changed. A 2.2 BIOS will work for a non-banked system with some major changes, and the root BIOS is the starting point for the banked version. The SCB module adds more trouble due to the equates and vagueness. This macro has an equate of 0FE00 hex, which would make you think that this is the exact address it will end up at. Wrong. Hidden in one line of the GENCPM discussion is a statement which points out that GENCPM will relocate "FE" locations into the BDOS and link the other references to them. If you check after GENCPMing, you will find this list of values located directly before the beginning of the BIOS (the last bytes of the BDOS).

While covering linking of modules, you should check the listing; I have shown the terminal dialog for the necessary generations. CPMLDR is the program that loads the system, as it is a separate file listed in the directory. The BIOS portion needed for CPMLDR can be the same BIOS as used in the full system, just take care that BOOT is a RETurn if all the initialization has been done. The order of events is not well explained in the manual, but the loader will call the boot entry of the LDRBIOS first, then start loading and displaying data. While debugging my BIOS, I used the print message routine to help tell me at what point it would go into never-never land. The use of statements like "now reloading CCP" would tell me that I was in the warm boot routine and not the cold boot. Make it a point to read the discussion of "debugging CP/M" in the "systems guide," and thoroughly review the programmer's guide. Many of the necessary facts are spread out over many headings and books, making it hard to keep track of them.

In last month's article I listed a BIOS routine and mentioned a similar one in SIG/M 26. Both of these 2.2 BIOSs will use the addresses shown for storing disk data (these are the original addresses chosen by SD in DDBIOS listing). These addresses are in the page 0 area, 040 hex to 07F hex, and must be the same in all three programs (monitor, BOOT, BIOS). Two problems of 3.0 demand that they be relocated; first, 050 hex is the location of several file control blocks. Banked operation will require that certain stacks, buffers and disk values be located in the fixed portion of memory. These locations can be above the PROM when it is located at F00h or just below it if an F800h PROM is chosen. GENCPM asks for the last free space of memory, so any location could be used and GENCPM will move the system to it.

The fixed memory locations will cause some problems because the write and read routines would normally get their data from the SCB files (which are public) and not from the memory locations that SD uses. For PROMS that have been written before understanding this problem, new routines will be needed to read the SCB data and then fill the monitor's buffer location for that data. A lot of these special problems arise from trying to keep the PROM in memory and fit CP/M around those parameters. Using a parallel port to turn the PROM on and off could make life simpler. Here's how; simply modify the boot loader routine so that the BIOS is complete and used for all operations (don't call any PROM entries—include the disk portion in

the BIOS). Now make the BOOT routine output to your port and turn off the PROM. The RETurn instruction jumps back to the CP/M 3.0 loader which loads the system file into upper memory. A master reset will always put you into the monitor if reset forces the parallel port back to the normal state. These steps will also solve the problem of clashing memory usage by removing the PROM and allowing buffers to be in the BIOS. However, any of SD's older programs that called the PROM BIOS entry points directly (format) will not work.

It can be rather hard to determine what procedure to follow first in setting up the system. If using DRI's programs, an attempt to bring up the banked unit first may not be too foolish. Non-banked operation with the PROM will be easier through the simpler routines (calls to PROM) but will be more confusing due to memory conflicts. A non-banked non-PROM system will require more programming, and will also demand that the PROM phantom operation be set up. For system integrators this is normal; considerable time must be spent deciding on the trade-offs of system configurations. Try and understand your limits in both time and skill, and study what must be done. After examining the problems at length, experiment with several of the solutions to discover further problems that you may have missed. The SCB relationship is one that is hard to understand, and is just the kind of problem that haunts beginning programmers. I found the DRI information on the SCB to be rather short and confusing. It is not clear whether the source of data is the SCB or the called routines in the BIOS. I believe it to be the latter with the SCB used as a reference data source.

Relations and Assemblers

The relationship between modules is confusing at first, and will require several practice assemblies. First, assemble a new BIOS unit. Using RMAC (A> RMAC BIOS.ASM), find any errors and correct them. Next, assemble the modules (if there are any) and link them. Remember that the order of linking is controlled by the order of the entry line. In other words, if BOOT is listed before BIOS, it becomes the main program instead of BIOS (as it should be). Generally speaking, only the major program need be first. The others are called from the main and need not be in any order after that, except that the SCB is usually last (and must be a separate file for GENCPM to link properly). Check the listing for more information on this topic. Public and external items need to be marked as such and declared in the beginning of the file. To find out if a declaration was missed, assemble the file and then link it; undeclared items will be shown as errors in the linking and will send you looking for them. It is best to use many short routines in programming, as it is easier to link routines than to handle excessively large files. At first, however, and if the shorter BIOS that calls the PROM is used, the simplest method may be to include the needed routines with a word processor. To boot from a 2.2 system, a CPMLDR.COM file is used. This is called like any COM file and will then load the CP/M system files into memory. Later, the LDR files are put on system tracks and called through the monitor boot command. The loader will need to be linked with an [L100] option to make a COM file for use at 100h. The BIOS is linked with the BOOT and SCB (CHARIO, FLPDRVR and others if needed) using the [B] switch to form an SPR file. This file will then be linked again by GENCPM to form the system files. The symbols or links that are established between the BDOS and the BIOS are not shown (normally a printed statement is displayed after linking).

The procedure will be to write or change the BIOS, assemble it, link it, GENCPM, run CPMLDR, and then test it. Once a working CPMLDR is obtained, it can be used ever after, even though the BIOS may have many major changes. The system files contain a header entry that instructs the loader where to put the system. At this point it is up to you to keep at it until the system works.

Review

These three articles are in no way a complete discussion of system integration or of what it takes to make advanced programmers out of novice hackers. I started out trying to cover it all, only to find that "all" would take many books. To even summarize what I have learned about the system and CP/M 3.0 is difficult. Documentation is still the most important aspect of any system. DRI has improved their documentation somewhat. Since I had not paid the $250 for professional help, I was unable to uncover all of the relationships within CP/M 3.0. I leave some of that for the reader, both as a challenge, and as a possibility for future articles for *The Computer Journal*.

I still find the Versafloppy II/696 to be a good product while operating under CP/M 2.2. However, the manual and documentation for implementing 3.0 are some of the worst that I have seen. It may work fine, but if the manual is any indication, I have my doubts (write and let us know if I am wrong). CP/M 3.0 is a good program if already implemented, and a real bear if not. I like 2.2 and have been able to master all of its strange relationships. 3.0 is new and undocumented other than by DRI. I hope this will change in time, but with all the pressure on CP/M86 and MSDOS, it looks to me like 3.0 will die a slow death. Those who now use 2.2 will probably not change for many reasons. Like myself, most users have developed a large library of utilities. Many of these utilities use the disk parameter data to display information—these programs will need changing for the new data tables. Disk usage is usually low with 2.2 but becomes higher in 3.0, although you do gain the ability to make changes and additions to the system more quickly and easily.

To conclude this discussion I would like to remind users that it is necessary to define your needs and desires before buying anything. As one can tell by the previous paragraph, I am not overly enthusiastic about 3.0. The enhancements and structure have become that of a programmer's operating system. Unix, although being touted as the operating system of the future, is now getting a lot of flack because of its complex nature. One must remember that these systems are written by programmers, for programmers, and not for the general public. An operating

system should allow you to use other programs freely and not require a lifetime of study to run. CP/M 3.0, 86, and MSDOS are being influenced by Unix's structure, so many of the newer versions may not be that desirable. One thing that I have tried to do in this article is "tell it like I see it." I too often see articles that gloss over the bad points only to get inexperienced users into trouble. To those readers who have found many of the industry's hidden and sometimes dark secrets, please write articles and send them in. I have dealt with BIOSs for some time now, and am still amazed at how many secrets they contain that I have yet to find...

Listing for System Integration

```
;       This listing is to show what is on the DRI's two
;       floppy disk as recieved from DRI.
;       resbdos3.spr  2k :bnkbdos3.spr  14k :ccp.com    4k
;       bdos3.spr    10k :device.com     8k :put.com    7k
;       gencom.com   15k :lib.com        7k :dir.com   15k
;       rename.com    3k :rmac.com      14k :save.com   2k
;       qirlbl.rex    2k :cpmlor.rel     1k :link.com  16k
;       copysys.com   2k :device.com     8k :set.com   10k
;       erase.com     4k :initdir.com   32k :date.com    3k
;       gencpm.com   26k :patch.com      3k :mset.com  11k
;       dump.com      1k :get.com        7k :mac.com   12k
;       pip.com       9k
;       disk1 files.....
;       bioskrnl.asm 16k :callvers.asm   1k :boot.asm   3k
;       chario.asm    5k :copysys.asm    1k :cpm3.lib   4k
;       scrovers.asm  1k :drvtbl.asm     1k :help.com   7k
;       fdc797sc.asm 11k :hexcom.com     2k :dump.asm   4k
;       modemaud.lib  1k :random.com    11k :help.hlp 61k
;       setdef.com    4k :hist.uti       2k :move.asm   1k
;       ports.lit     2k :scb.asm        3k :sid.com    8k
;       submit.com    6k :trace.uti      2k :xref.com  14k
;       show.com      9k :type.com       3k :z80.lib    6k
;       ***********************************************
;       if all of the above disk files are not present on
;       your CP/M3 disks--you have not recieved a full
;       version and ask for the rest of the files.!!!!
;            The following is the dialog that is necessary
;       to generate a runing non-banked cpm system. First
;       the BIOS listed below should be created or modified
;       from your existing file. By looking at last months
;       listing you should be able to see the changes and
;       make corresponding one to your own version..
;       A>rmac bios.asm
;       A>rmac scl.asm
;       A>link bios3[p]=bios,scl
;       A>gencpm
;         .....(lots of texts answer as follows
;         (B,),cr,cr,cr,cr,cr,cr,EF,),cr to end
;         (display will show map, recore for possible
;          debugging....all that's needed now is cpmldr)
;       A>link cpmldr.rel01,=cpmldr,bios,bdos
;       A>gsd cpmldr.com
;         #0EC0
;       0EC0  C3 C9  (put a return in the BOOT entry)
;       0EC2  49 .
;         ^C
;       A>save 15 cpmldr.com  (for bios size of below page)
;       A>cpmldr
;            This should now load the files and bring up
;       CP/M3.1 if not start looking for typos or other
;       common type errors. I had to double the allocation
;       vectors to make mine work.  LOTS OF LUCK!!!!!!
;       ***********************************************
;            ADAPTED VERSION FOR CPM3 I/O
; CP/M BASIC INPUT/OUTPUT OPERATING SYSTEM (BIOS)
;  This version boots in single density, and calls
; A&E bimole (tm
; variables and stacks are in FECF space to be fixed
; in core switched modes, space is set aside for other
; possible programs, ie:RAMDISK, TIMP
;
F0C0 =          SDPROM  EQU  0f0c0h      ;LOCATION OF SEPROM(IF USED)
003E =          MSIZE   EQU  62          ;MEMORY SIZE IN KBYTES.
;
;  PUT EQUATES FOR I/O OR OTHER NON PROM SPECIFIC ITEMS
;     use 0fc00h as offset for original bd prom
;          buffer locations
FC42 =          UNIT:   EQU  0fc42h      ;UNIT BYTE FOR DISK SELECT
;
002E =          CONDTA:  EQU   02Eh      ;SERIAL DATA PORT
002F =          CONCTL:  EQU   02Fh      ;SERIAL PORT CONTROL
;
                extrn   @mxtpa
;
                CSEG
;
FBFF =          BTSTK    EQU  0FBFFh
;
                ORG  bios                ;START OF BIOS.
;JUMP TABLE
0000 C31F01     JMP  BOOT       ;FROM COLD START LOADER.
0003 C37C02     ?WBOOT: JMP  WBOOT      ;FROM WARM BOOT.
0006 C3FC01     JMP  CONST      ;CHECK CONSOLE KB STATUS.
0009 C39C01     ?conin: JMP  CONIN      ;READ CONSOLE CHARACTER.
000C C39401     JMP  CONOT      ;WRITE CONSOLE CHARACTER.
000F C3C902     JMP  LIST       ;WRITE LISTING CHAR.
0012 C3AC01     JMP  AUXOUT     ;WRITE PUNCH CHAR.
0015 C3A301     JMP  AUXIN      ;READ READER CHAR.
0018 C36A01     JMP  HOME       ;MOVE DISK TO TRACK ZERO.
001B C35401     JMP  TSELDSK    ;SELECT DISK DRIVE.
001E C31EFC     SETTRK: JMP  sdprom+1eh  ;SET TO TRACK IN REG. A.
0021 C321FC     JMP  sdprom+21h ;SET SECTOR NUMBER.
0024 C324F0     JMP  sdprom+24h ;SET DISK STARTING ADR.
0027 C327F0     JMP  SDPROM+27H ;READ SELECTED SECTOR.
002A C32AF0     JMP  SDPROM+2AH ;WRITE SELECTED SECTOR.
002D C30102     JMP  PRSTAT     ;LIST STATUS CHECK
0030 C30F02     JMP  SECTRAN    ;SECTOR TRANSLATE ROUTINE
0033 C39801     JMP  CONOST     ;CONSOLE OUT STATUS
0036 C3A601     JMP  AUXIST     ;AUX STATUS IN
0039 C3A901     JMP  AUXOST     ;AUX OUT STATUS
003C C30301     JMP  DEVTBL     ;DEVICE TABLE
003F C30601     JMP  DEVINI     ;DEVICE INIT
0042 C3D500     JMP  DRVTBL     ;DRIVE TABLE
0045 C3F900     JMP  MULTIO     ;MULTI SECT COUNT
0048 C30901     JMP  FLUSH      ;FLUSH DISK BUFFERS
004B C3FE00     JMP  MOVE       ;MOVE DATA
004E C31B01     JMP  TIME       ;TIME RETURN
0051 C31101     ?bnksl: JMP  SELMEM     ;SELECT MEMORY
0054 C30C01     JMP  SELBNK     ;SEL MEM BANK
0057 C31B01     JMP  XMOVE      ;INTER BANK MOVE
005A C3AC01     JMP  NOTYET
005D C3AC01     JMP  NOTYET
0060 C3AC01     JMP  NOTYET
;
;       DISK TABLES FOR 8 DRIVES
;
;     dseg
;
0063 =          DPBASE: EQU   $          ;BASE OF DISK PARAMETER BLOCKS
0063 24F0           DW   sdprom+24h
0065 27F0           DW   sdprom+27h
0067 D400           DW   NORET
0069 D400           DW   NORET
006B 0000           dw   0,0
006F A9000000  DPE0:    DW    XLT0,0000H       ;TRANSLATE TABLE
0073 00000000           DW    0000H,0000H      ;SCRATCH AREA
0077 00000000           DW    0000H,0000H
007B C300               DW    DPB0             ;PARM BLOCK
007F 49021B02           DW    CSV0,ALV0
0083 FFFFFEFF           DW    DIRBCB,DTABCB
0087 FFFF               DW    HASH
0089 00                 DB    0                ;HBANK
008A 24F0               DW    sdprom+24h
008C 27F0               DW    sdprom+27h
008E D400               DW    NORET
0090 D400               DW    NORET
0092 0100               db    1,0
0094 A9000000  DPE1:    DW    XLT0,0000H       ;TRANSLATE TABLE
0098 00000000           DW    0000H,0000H      ;SCRATCH AREA
009C 00000000           DW    0000H,0000H
00A0 C300               DW    DPB0             ;PARM BLOCK
00A4 B7027901           DW    CSV1,ALV1        ;CHECK, ALLOC VECTORS
00A8 FFFFFEFF           DW    DIRBCB,DTABCB
00AC FFFF               DW    HASH
00AE 00                 DB    0                ;HBANK
;
;     cseg
;
00AF =          XLT0:   EQU   $                ;TRANSLATE TABLE
00AF 0107010319         DB    1,7,13,19,25,5,11,17,23
00B4 0306091502         DB    3,9,15,21,2,8,14,20,26
00B9 040C1218           DB    4,12,18,24,4,10,16,22
0000 =          DPB0:   EQU   $                ;SD DISK PARM BLOCK
0000 1A00               Dw    26               ;SEC PER TRACK
0004 03                 DB    3                ;BLOCK SHIFT
0005 07                 Db    7                ;BLOCK MASK
0006 00                 DB    0                ;EXTNT MASK
0007 F200               Dw    242              ;DISK SIZE-1
0009 3F00               Dw    63               ;DIRECTORY MAX
000B C0                 DB    192              ;ALLOC0
000C 00                 DB    0                ;ALLOC1
000D 1000               Dw    16               ;CHECK SIZE
000F 0200               Dw    2                ;OFFSET
0011 00                 Db    0                ;PSH
0012 00                 Db    0                ;PHM
;
0013 C9         NORET:  RET
;
FFFE =          DIRBCB: EQU   0FFFEh
FFFF =          DTABCB: EQU   0FFFFh
FFFF =          HASH:   EQU   0FFFFh
0000 =          HBANK:  EQU   0000h
;
0014 210000     DRVTBL: LXI   H,xDTBL
0017 C9                 RET
;
0018 6000F000   xdtbl:  dw    dpe0,dpe1
001C 0000000000         dw    0,0,0,0,0,0
0021 0000000000         dw    0,0,0,0,0,0,0,0
;
0026 32FD00     MULTIO: sta   #CNT
0029 C9                 RET
002A 00         #cnt:   db    0h
;
002B Fb         MOVE:   XCHG
002C EDb0               DB    0EDh,0B0h        ;ldir
002E Eb                 XCHG
002F C9                 RET
;
0030 C3AC01     DEVTBL: JMP   NOTYET
;
0033 C3AC01     DEVINI: JMP   NOTYET
;
0036 C3AC01     FLUSH:  JMP   NOTYET
;
0039 321001     SELbnk: sta   #dbnk
003C C9                 ret
003D 00         #dbnk:  db    0h
;
003E D3FF       SELmem: OUT 0FFH
0040 321701             STA  #CBNK
0043 C9                 RET
0044 00         #CBNK:  DB   0H
;
0045 C3AC01     XMOVE:  JMP   NOTYET
;
0048 C3AC01     TIME:   JMP   NOTYET
;
004B 31FFFB     BOOT:   LXI  SP,BTSTK   ;SET STACK POINTER.
                ;     when using the bios for cpmldr replace
                ;     above line with an "RET"
;
004E AF                 XRA  A
004F 320400             sta  4
0052 320300             STA  3           ;IOBYTE
0055 3242FC             STA  unit
0058 CD3F01     SETCPM: CALL SETUP       ;SET UP JUMPS.
005B 21C201             LXI   H,SMSG
005E CDAEC2             CALL  PMSG
0061 CDD702             CALL   ?INIT
0064 CDDEC2             CALL  ?LDCCP
0067 DB2F               IN    CONDTA     ;CLEAR CONSOLE STATUS.
0069 C30001             JMP  0100H       ;JUMP TO CCP.
;
; SET UP JUMPS TO CP/M
;
SETUP:
                ;       MVI    A,1     ; for banked operation
                ;       CALL      selbnk
006C 3EC3               MVI  A,0C3H      ;PUT JMP TO WBOOT
006E 320000             STA  0           ;ADR AT ZERO.
0071 320500             STA   05H
0074 210300             LXI  H,?WBOOT
```

continued

```
0144 220100             SHLD 1
0147 2A0000             LHLD   @mxtpa     ;PUT JUMP TO BDOS
014A 220600             SHLD 6            ;AT ADR 5,6,7.
014D C9                 RET               ;RETURN FROM SETUP.

                ;
                ; temporary disk select routines
                ;
                TSELDSK:
0150 210000               LXI  H,0
0153 79                 MOV  A,C
0154 FE02               CPI  2
0156 D0                 RNC
0157 E6FD               ANI  0FDH
0159 3242FC             STA UNIT
015C 219000             LXI H,DPE1
015F FE01               CPI 1
0161 C8                 RZ
0162 214D00             LXI H,DPE0
0165 C9                 RET
                ;
                HOME1:
0166 010000               LXI  B,0
0169 C31F00             jmp  settrk
                ;
                ;
                ; WARM-BOOT:  READ ALL OF CPM BACK IN
                ; EXCEPT BIOS, THEN JUMP TO CCP.
                ;
016C 31FFFH     WBOOT:  LXI  SP,btstk   ;SET STACK POINTER.
016F 3A42FC             LDA    UNIT
0172 321A02             STA    TEMP
0175 AF                 XRA    A
0176 3242FC             sta    unit
0179 CD3F01             CALL   SETUP
017C CDDE02             CALL   ?RLCCP
017F 3A1A02             lda    temp
0182 3242FC             sta    unit
0185 C30001             JMP    0100H     ;GO BACK TO CCP
                ;
                ; CHECK CONSOLE I/O STATUS.
                ;
0188 CD06F0     CONST:  CALL   SDPROM+06H       ;READ CONSOLE STATUS.
018B C9                 RET                     ;RETURN FROM CONST.
                ;
                ; READ A CHARACTER FROM CONSOLE.
                ;
018C CD09F0     CONIN:
018F C9                   CALL  SDPROM+09H
                        RET
                ;
                ; WRITE A CHARACTER TO THE CONSOLE DEVICE.
                ;
0190 CD0CF0     CONOT:  CALL   SDPROM+0CH
0193 C9                 RET
                ;
0194 DB0F       CONOST: IN     CONCTL    ;not in original prom
0196 E602               ANI    02H
0198 C8                 RZ
0199 C6FF               ADI    0FFH
019B C9                 RET
                ;
019C C3ACC1     AUXOUT:   JMP  NOTYET
                ;
019F C3ACC1     AUXIN:    JMP  NOTYET
                ;
01A2 C3ACC1     AUXIST:   JMP  NOTYET
                ;
01A5 C3ACC1     AUXOST:   JMP  NOTYET
                ;
01A8 AF         NOTYET: XRA  A
01A9 C9                 RET
                ;
                ; PRINT THE MESSAGE AT H&L UNTIL A ZERO.
                ;
                ?PMSG:
01AA E5         PMSG:     push h
01AB C5                   push b
01AC 7E         PMSG1:    MOV  A,M        ;GET A CHARACTER.
01AD B7                   ORA  A          ;IF IT'S ZERO,
01AE CABF01               JZ   epmsg      ;RETURN.
01B1 4F                   MOV  C,A        ;OTHERWISE,
01B2 E5                   push h
01B3 CD9001               CALL CONOT      ;PRINT IT.
01B6 E1                   pop  h
01B7 23                   INX  H          ;INCREMENT H&L,
01B8 C3AD01               JMP  PMSG1      ;AND GET ANOTHER.
01BB C1         epmsg:    POP B
01BC D1                   POP d
01BD C9                   ret
                ;
                ; CBIOS MESSAGES
                ;
01BE 0D0A435021FSMSG:   DB   0Dh,0AH,'CP/M 3.0 PLUS   SD SYSTEMS-KIBLER
01E7 0D0A3821253        Db   0Dh,0Ah,'8*SD A&B '
01F2 0D0A3430           DB   0Dh,0Ah,MSIZE/10+'0',MSIZE MOD 10 + '0'
01F6 4B20563332F        DB   'k V3.0 of 3/01/84 ',0
                ;
                ; WRITE A CHARACTER ON LIST DEVICE.
                ;
0209 CD0FF0     LIST:   CALL   SDPROM+0FH
020C C9                 RET
                ;
020D AF         PRSTAT: XRA A
020F C9                 RET     ;RETURN ALWAYS NOT READY
                ;
                ;
                ;SECTOR TRANSLATION ROUTINE FOLLOWS
020F 69         SECTRAN:  MOV  L,C
0210 60                   MOV  H,B
0211 7A                   MOV  A,D
0212 B3                   ORA  E
0213 C8                   RZ
0214 Eb                   XCHG
0215 09                 DAD  B
0216 6F                 MOV  L,M
0217 2600               MVI  H,0
0219 C9                 RET
                ;
021A            TEMP:   DS   1
                ;
                ; DISK BUFFERS double normal values
                ;*****************************************************
021B            ALV0:   DS       62
0259            CSV0:   DS       32
0279            ALV1:   DS       62
02B7            CSV1:   DS       32
                ;*****************************************************
                ;       title   'Boot loader module for CP/M 3.0'
                ;    from DRI's file cut down to necessary stuff only
0005 =          bdos    equ 5
```



```
                ;       dseg    ; init done from banked memory
                ?init:
02D7 212503             lxi h,signon$msg
02DA CDAE01             call ?pmsg              ; print signon message
02DD C9                 ret
                ;
                ;   cseg
                ;
                    ;   This version of the boot loader loads
                    ;   the CCP from a file
                    ;   called CCP.COM on the system drive (A:).
                ?ldccp:
                ?RLCCP: ; First time, load the A:CCP.COM file into TPA
02DE AF3226C03          xra a ! sta ccp$fcb+15          ; zero extent
02E2 2100002270         lxi h,0 ! shld fcb$nr           ; start at beginning
02E8 115D03CD11         lxi d,ccp$fcb ! call open       ; open ccp file
02EE 3CCA0503           inr a ! jz noCCP                ; error if no file...
02F2 110001CD16         lxi d,0100h ! call setdmax      ; start of TPA
02F8 118000CD18         lxi d,128 ! call setmulti       ; allow up to 16k byte
02FE 115D03CD20         lxi d,ccp$fcb ! call readx      ; load the thing
                                                        ; now,
0304 C9                 ret
                noCCP:                  ; here if we couldn't find the file
0305 213C03CDAE         lxi h,ccp$msg ! call ?pmsg      ; report this...
030B CD0900             call ?conin                     ; get a response
030E C3DE02             jmp ?ldccp                      ; and try again

                        ; CP/M BDOS Function Interfaces

                open:
0311 0E0FC30500         mvi c,15 ! jmp bdos     ; open file control block
                setdmax:
0316 0E1AC30500         mvi c,26 ! jmp bdos     ; set transfer address
                setmulti:
031B 0E2CC30500         mvi c,44 ! jmp bdos     ; set record count
                readx:
0320 0E14C30500         mvi c,20 ! jmp bdos             ; read records

0325 0D0A0D0A6Csignon$msg       db      13,10,13,10,'loading 3.0 BIOS',13,10,0
033C 0D0A42494Fccp$msg          db      13,10,'BIOS Err on A: NO CCP.COM file',0

035D 0143435020ccp$fcb          db      1,'CCP     ','COM',0,0,0,0
036D                            ds      16
037D 000000     fcb$nr          db      0,0,0

0380                    end
```

Industry News

Computer Pioneer Days Conference

SYBEX Computer Pioneer Days, a new two-day conference to be held in San Francisco, June 15-16, 1984, will, for the first time, honor the "living legends" of the microcomputer industry, on the 10th anniversary of the microcomputer. (The Altair computer was introduced in 1974.)

The pioneer speakers represent most areas of the personal computer industry, from hardware, software, and venture capital to microchip design and development. They will discuss the growth of the microcomputer industry from their own experiences—how they started their companies or developed their products; who, if anyone, was with them from the beginning; how their first projects were financed; what their early successes, failures, and motivations were. They will offer their own perspective on the melding of entrepreneurship and "high technology," for themselves and for other people who may have wondered "what it takes" to strike out on an independent path. Awards for outstanding microcomputer industry achievements will be presented at a special reception on Friday, June 15, 1984, at the Hyatt Regency, from 7:00 to 9:00 p.m. For more information write: SYBEX Computer Pioneer Days, 2344 Sixth Street, Berkeley, CA 94710; or call (415) 848-8233. ■

3½" Microfloppy New Standard?

A paper from Hewlett-Packard entitled "Why the 3½" Microfloppy Will Be the Next Industry Standard" is available to those interested. The document outlines the need for standardization in the sub-5¼" disk drive market, and explains why the 3½" is ideally positioned to be the industry standard. Currently, Hewlett-packard leads the trend and has incorporated the 3½" in its entire line of personal computers. The paper presents a comparison of Hewlett-Packard's 3½" disk drives with the current 5¼" products, and provides technical specifications at the end of the document. It can be obtained be writing: Kathy Kimball, Hewlett-Packard, Greely Division, 700 71st Ave, Greely, CO 80634. ■

LINEAR OPTIMIZATION WITH MICROS

A Product Review

by The Computer Journal Staff

The Computer Journal has been reviewing a very interesting software package called LO, which is short for *Linear Optimization*. The statistical technique on which this program is based has traditionally been done by hand; it involves the simultaneous solution of a group of linear equations, hence the name. It was called linear programming before the term "programming" became associated with computers.

Although the methods have been adapted to computers before, this is the first version your editors have seen that is specifically designed to run on a modest size computer. The version we reviewed runs under CP/M and will work fine in as little as 48K of memory. Considering the memory requirements of much current software, this is indeed a "plus" factor for the package. It is also available for MS-DOS, IBM-DOS, and possibly for other formats.

The purpose of LO, briefly expressed, is to optimize the use of scarce resources. It can be used either to maximize some desirable quality (such as profit) or minimize some other factor (such as cost). It is a decision-making tool which allows the user to weigh several variables against one another for optimum results.

In business and industry we often find that several variables work against one another. The amount of labor available to produce a product, and its cost, are important factors. Another might be the shortage of some necessary material, or the amount of some product which can be successfully marketed. To illustrate this principle, we made up an example of a small computer company which builds CPU boards, memory boards, and video boards. The number of chips each requires differs, as does the number of hours labor to produce each, and the wage that has to be paid the technicians for doing the different kinds of work. Finally, we decided that our imaginary company has only 500 chips on hand (which, for our example, are all alike and will work in any position on any board). Our manufacturer has only $1000 in the bank to meet the payroll, and wants the three available technicians to finish the job this week.

LO problems are described in terms of "bounds" which are the desired product, "goals" which are the optimization of the bounds, and "constraints", or limits, which determine the final outcome. In this example we need to produce some number of each of the possible products; we have a limited number of chips with which to do so, a limited number of working hours, and a limit on the money we can spend for labor.

The "goal" is to produce the "mix" of boards that will most effectively use our resources and yield the maximum profit.

This problem is simple enough to follow but too complicated to be worked by hand in any reasonable length of time. Figure 1 is the solution that LO produced in less than 60 seconds. Lines numbered 1 through 6 are the source file. We'll explain it line by line.

```
                    LINEAR OPTIMIZATION

Hacker Products Unlimited, production schedule
Linear Optimization 1.0 (c) 1982 The Acme Computer Company

     1  option title="Hacker Products Unlimited, production schedule",list;
     2  bounded @cpu,@memory,@video;
     3  limit chips by 20@@cpu + 45@@memory +12@@video <=500;
     4  limit hours by 8@@cpu + 5 5@@memory +3.8@@video <=120;
     5  limit cost by (5@@)8@cpu + ((685.5)8@@memory + (783.8)8@@video <=1000;
     6  max 90@@cpu + 125@@memory + 75@@video;

Number of iterations: 2
Goal function value:    2440.

 VARIABLE      OPTIMAL   ERROR      MARGINAL          COST
 NAME          VALUE     ESTIMATE   RETURN           REGION

 @cpu            0.000     0.00%     -64.76   -INFINITY  ..     154.8
 @@memory        4.381     0.00%      125.0      108.6   ..     281.2
 @@video        25.24      0.00%      75.00      47.80   ..      86.36

 LIMIT         SURPLUS   ERROR      MARGINAL          LIMIT
 NAME                    ESTIMATE   VALUE            REGION

 chips           0.000     0.00%      0.5952    378.9   ..     981.8
 hours           0.000     0.00%     17.86       61.11  ..     144.1
 $cost         184.1       0.00%      0.000     815.9   ..  +INFINITY
```

Figure 1

Line 1 allows us to specify the title and instruct the program to list the source along with the results; this line is optional. Line 2 sets as bounds the number (#) of CPU boards, memory boards, and video boards. The solution of the problem will be expressed in terms of the number of each board we should build.

Line 3 says that we need 20 chips for each CPU (20*#CPU) 45 for a memory and 12 for a video board, and that the total cannot exceed 500 (< = 500).

Line 4 similarly limits the hours that can be used—so many for each kind of board, for a total of, or less than, 120.

Line 5 is the tricky one. It expresses the labor cost for each kind of board in terms of hours times pay rate (i.e.; $5 per hour times 8 hours) and sets $1000 as the maximum that we can spend.

Line 6 states our goal, which is to make the maximum number of CPU boards at $90 each, and/or memories at $125, and /or video boards at $75, and to produce these in a proportion that will yield the maximum sales volume.

As the printout shows, it would be uneconomical to produce any CPU boards under the given conditions, but we can produce 4.381 memories and 25.24 video boards for a total of $2440 (goal function).

The other columns have various interpretations. For example, they show that this solution would use all of our chips and all of the work hours but leave a small balance of our money ($184.10). Other columns give us clues as to the value of increasing the labor, or number of chips, or whatever we have to work with. After gaining experience with the program, these columns might enable us to modify our inputs and find alternate solutions.

For example, it might seem strange that although memory boards are the most profitable item, we are told to

make fewer of them. Why? The marginal value of chips is a clue. Let's run the problem again pretending that we found 250 more chips down in the basement. The result is Figure 2. Notice that we were able to produce more memory boards with no increase (actually a decrease) in our labor costs! Evidently the number of chips we started with had a severe effect on our production.

```
                    LINEAR OPTIMIZATION

Hacker Products Unlimited, production schedule
Linear Optimization 1.0 (c) 1982 The Acme Computer Company

     1  option title="Hacker Products Unlimited, production schedule",list;
     2  bounded @cpu,@memory,@video;
     3  limit chips by 20*@cpu + 45*@memory +12*@video <=750;
     4  limit hours by 8*@cpu + 5.5*@memory +3.8*@video <=120;
     5  limit cost by (588)*@cpu +(685.5)*@memory + (783.8)*@video <=1000;
     6  max 90*@cpu + 125*@memory + 75*@video;

Number of iterations: 2
Goal function value:    2589.

 VARIABLE      OPTIMAL   ERROR      MARGINAL          COST
 NAME          VALUE     ESTIMATE   RETURN            REGION

 @cpu            0.000     0.00%     -64.76   -INFINITY  ..      154.8
 @memory        13.43      0.00%     125.0      108.6    ..      281.2
 @video         12.14      0.00%      75.00      47.80   ..       86.36

 LIMIT         SURPLUS   ERROR      MARGINAL          LIMIT
 NAME                    ESTIMATE   VALUE             REGION

 chips           0.000     0.00%      0.5952   378.9    ..      981.8
 hours           0.000     0.00%     17.86      91.67   ..      150.7
 $cost         233.9       0.00%      0.000    766.1    ..   +INFINITY
```

Figure 2

Getting greedy, we now raise the price of video boards by $10 and reduce the price of CPU boards to make up for it. The result is Figure 3; LO tells us to make the same mix of boards as in Figure 2, but at the new price we will make a little more profit.

The documentation that comes with LO explains the formulation of problems quite well, and provides example problems and solutions from several kinds of activity—from farming (livestock feeding) to finance (stock market investment).

Our impressions of the program follow: first of all, it works as advertised, is modestly priced, and the documentation is very good. It is fast and runs in a small memory. It can be very useful to anyone who has to make the kind of decisions it is designed for. It permits the use of sophisticated mathematical techniques by persons who would not otherwise be able to solve them.

On the negative side, LO requires you to be familiar with your CP/M utilities. Computer users who do not work directly with their operating system might not be ready for this. You write the source input with ED (or any similar editor) and the solution goes back onto the disk in a new file with an LIS extension. The only way to view the output is by way of the TYPE or PIP features of CP/M.

```
                    LINEAR OPTIMIZATION

Hacker Products Unlimited, production schedule
Linear Optimization 1.0 (c) 1982 The Acme Computer Company

     1  option title="Hacker Products Unlimited, production schedule",list
     2  bounded @cpu,@memory,@video;
     3  limit chips by 20*@cpu + 45*@memory +12*@video <=750;
     4  limit hours by 8*@cpu + 5.5*@memory +3.8*@video <=120;
     5  limit cost by (588)*@cpu +(685.5)*@memory + (783.8)*@video <=1000;
     6  max 80*@cpu + 125*@memory + 85*@video;

Number of iterations: 2
Goal function value:    2711.

 VARIABLE      OPTIMAL   ERROR      MARGINAL          COST
 NAME          VALUE     ESTIMATE   RETURN            REGION

 @cpu            0.000     0.00%     -98.57   -INFINITY  ..      178.6
 @memory        13.43      0.00%     125.0      123.0    ..      316.7
 @video         12.14      0.00%      85.00      43.60   ..       86.36

 LIMIT         SURPLUS   ERROR      MARGINAL          LIMIT
 NAME                    ESTIMATE   VALUE             REGION

 chips           0.000     0.00%      0.07143  378.9    ..      981.8
 hours           0.000     0.00%     22.14      91.67   ..      150.7
 $cost         233.9       0.00%      0.000    766.1    ..   +INFINITY
```

Figure 3

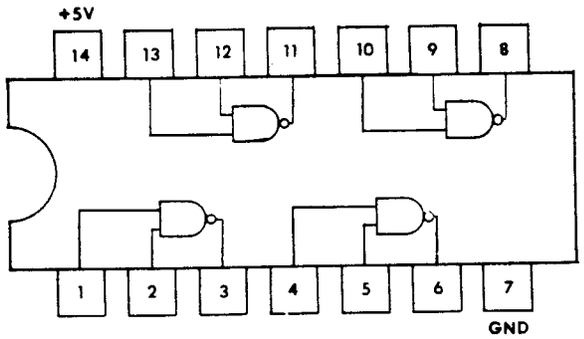
We feel that a more "automatic" printout would be useful. A real bonus would be the ability to enter the variables interactively rather than having to compose the formulae by hand and write the source with the editor. Another nice feature (not necessary, but perhaps helpful) would be some "action" on the screen while the program runs. These are cosmetic suggestions—we are otherwise very enthusiastic about this software product. It is available from Acme Computer Co., Box 51193, Seattle, WA, for about $150.

The level of manufacturer support is a very important consideration when selecting software, and we are favorably impressed with Acme's support of their products. One month after we received the program, they sent a questionaire with a stamped addressed envelope to obtain information for future revisions. Later, we received an announcement of a revision which can be obtained by either returning the original disk or sending $5 to cover the cost of material and handling.

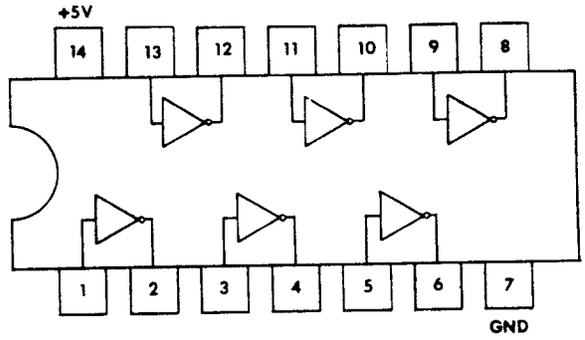
We sent Acme our comments on the program, and they responded by return mail to advise that most of these improvements are already in progress for the next revision. It is very unusual to find this level of support in the microcomputer industry, and we do not hesitate to recommend Acme as a supplier. ■

LSTTL Reference Chart

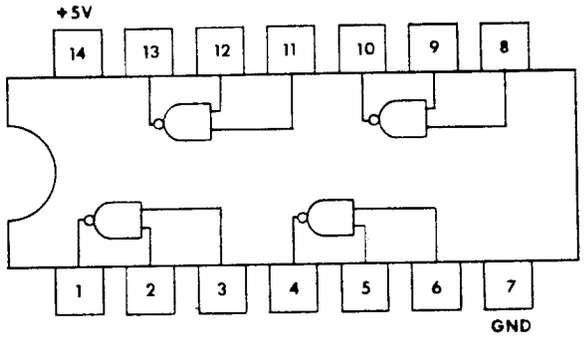
74LS00



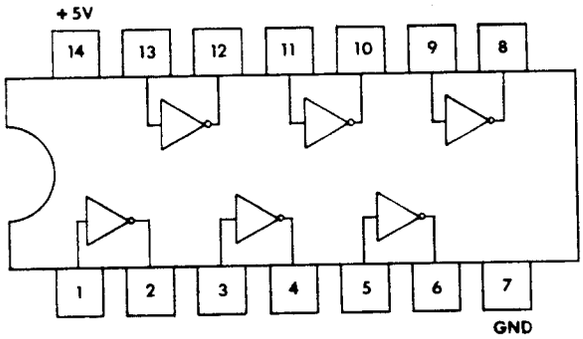
74LS04



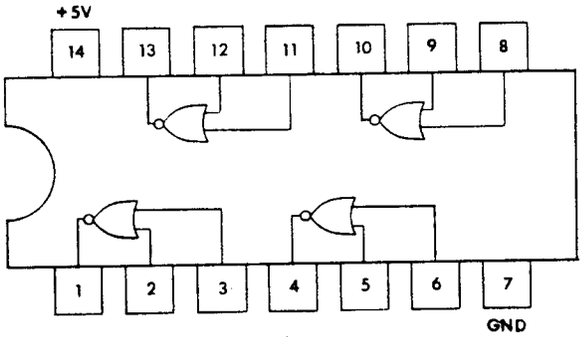
74LS01



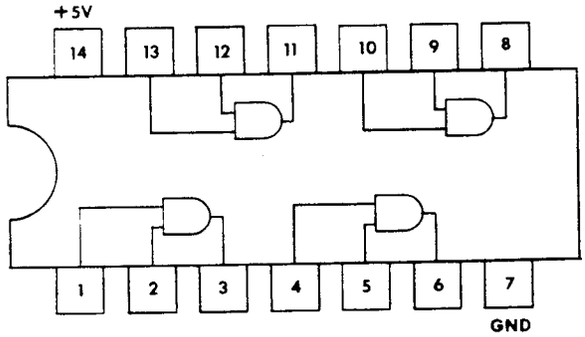
74LS05



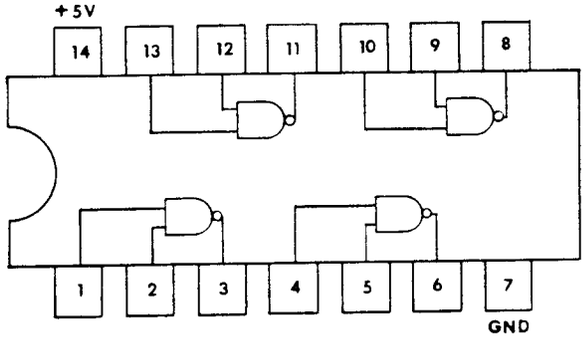
74LS02



74LS08

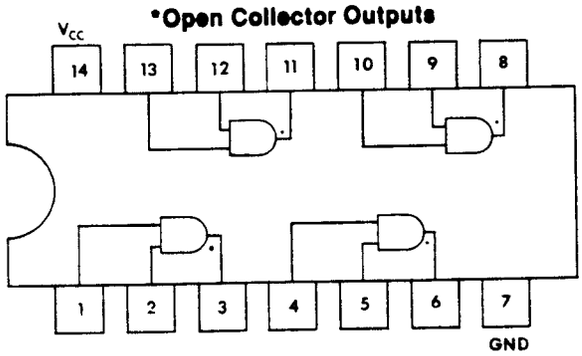


74LS03

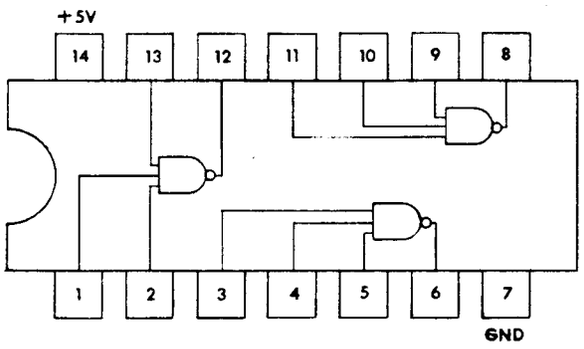


74LS09

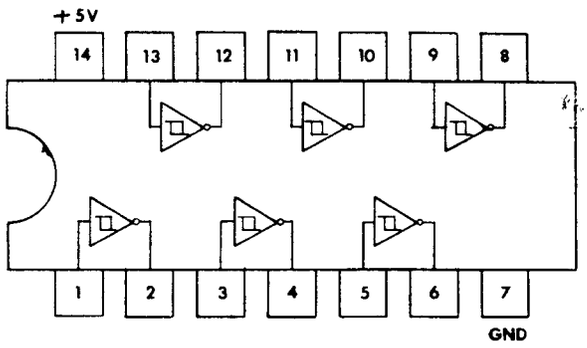
***Open Collector Outputs**



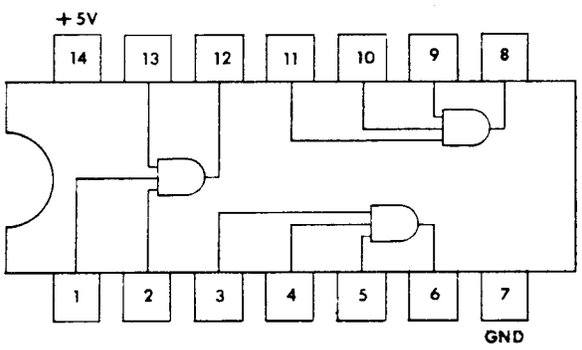
74LS10



74LS14

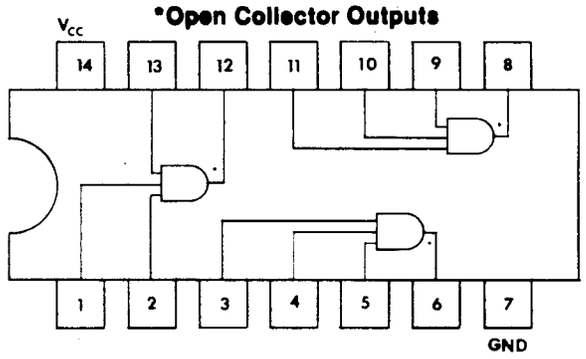


74LS11



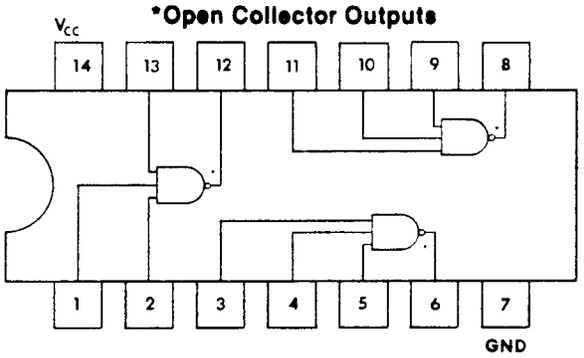
74LS15

*Open Collector Outputs

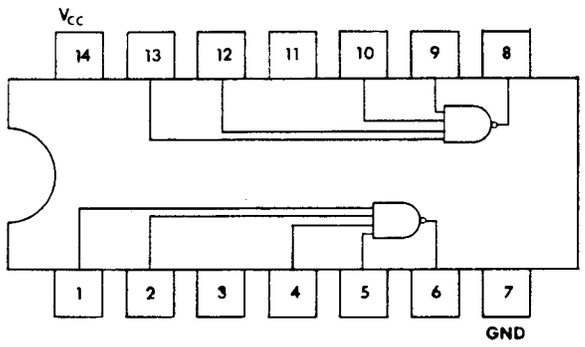


74LS12

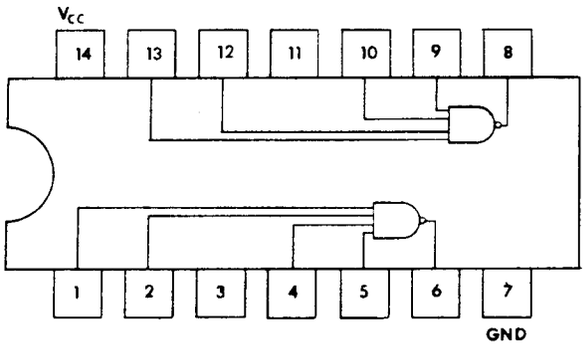
*Open Collector Outputs



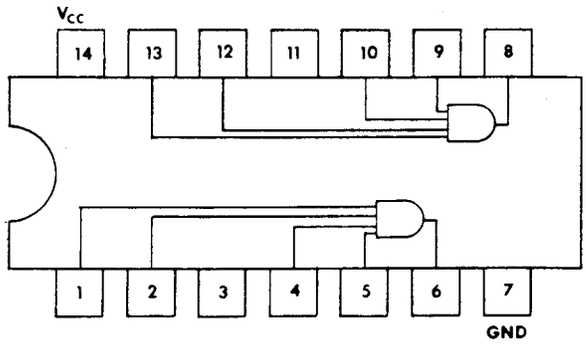
74LS20



74LS13



74LS21



Books of Interest

STD Bus Interfacing

by Christopher A. Titus, Jonathan A. Titus, & David G. Larson.
Another book in the BLACKSBURG Continuing Education Series®
Published by Howard W. Sams, Inc.
286 pages, 5½ × 8½ softbound, $13.95

What do you do when you need a microcomputer for a specific application and the medium cost appliance computers do not have the features you need, the larger business computers are too expensive and still don't have the I/O features you need, and an S-100 bus system is overkill? A STD Bus system may be a good choice because there is a wide variety of cards available so that you can tailor a system to have the special features you need without paying for a lot of unnecessary frills. It is also relatively easy and inexpensive to modify the system by adding or replacing cards.

This book explains the STD Bus, but even more importantly, it contains a very thorough explanation of what a microcomputer bus is, what signals are needed, and how these signals are used.

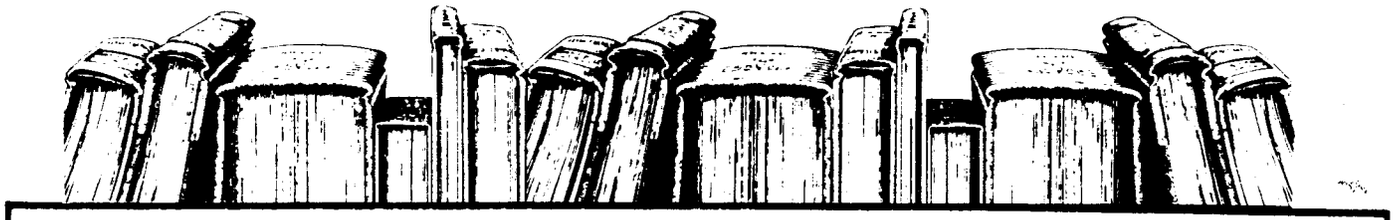
The information on accumulator I/O versus memory-mapped I/O, addressing and address decoding, I/O interfacing, and interrupts is especially helpful. The book is written in an easy to follow manner. A good example of the care taken in the writing is the sample programs, which are written in assembly language for the 8085, Z-80, 6502, and 6800 CPUs to make the book useful to people working with different CPUs.

The contents are as follows:

- **Chapter 1 What Is The STD Bus?.** Other Control Signals, Physical Standards, Why Use the STD Bus?, STD Bus Processors, Memory, Read-Only Memories, Read/Write Memory, Control Signal Generation, Memory Maps, I/O Devices, Memory-Mapped I/O, Accumulator I/O, CPU Compatibility, Data Transfer Timing, Chip Incompatibility, and Nonstandard Signals.
- **Chapter 2 I/O Device Addressing.** Address and Control Signals, Device Addressing, Using Gates for Address Decoding, Using Decoders, Larger Decoders, Using Comparators, Memory-Mapped I/O, and Using PROMS.
- **Chapter 3 Output Port Interfacing.** Output Timing, Latches in Output Ports, A Traffic Light Controller, LED Displays, Digital-to-Analog Converters, Data Displays, Other DAC Considerations, I/O Chips, and Memory-Mapped Output Ports.
- **Chapter 4 Input Ports.** Designing Input Ports, An ASCII Keyboard Interface, Flags, Another Keyboard Interface, An Analog-to-Digital Converter Interface, A Simple Logic Tester, and Memory-Mapped Input Ports.
- **Chapter 5 Interrupts and Direct-Memory Access.** Basic Interrupt Operation, STD Bus Interrupts, The 8085, The 8088, The Z-80 and NSC800, Serial Priority, The 6800, 6809, and 6502, 6809 Improvements, An Interrupt Review, Interrupt Software, Interrupts and the Stack, Interrupt Timing, Direct-Memory Access, Requesting the Bus, Direct-Memory Access Controllers, and DMA Software.
- **Chapter 6 General-Interest Interface Cards.** The Mostek DIOB/BIOP, The Enlode 214 Display System, The Analog Devices RT1-1260, The Atec 710 Thumbwheel Switch Interface, The Matrix 7911 Stepper-Motor Controller, and the Pro-Log 7304 Dual UART Card.
- **Appendix A The STD Bus Standard.** Organizational and Functional Specifications (With Pin Definitions), Electrical Specifications, and Mechanical Specifications.
- **Appendix b Voltage Input Configurations.** Input Multiplexer Guidelines, and Analog Input Multiplexer.
- **Appendix C Index of STD Bus Manufacturers.**

The detailed information on how a bus is organized is worth the price of the book even if you never intend to use a STD bus, because this knowledge can be applied to other computer buses. ■

This book is available from Group Technology, Ltd., P.O. Box 87, Check, VA 24072, for $13.95 plus $1.00 shipping.



The Bookshelf

CP/M Primer

Helps microcomputer veterans and novices alike find the answers about CP/M in a complete, one-stop sourcebook that's a Sams best-seller! Gives you complete CP/M terminology, hardware and software concepts, startup details, and more for this popular 8080/8085/Z-80 operating system. Helps you begin using and working with CP/M immediately, and includes a list of compatible software, too. By Stephen Murtha and Mitchell Waite. 96 pages, 8½x11, comb. ©1980.....$16.95

Soul of CP/M: Using and Modifying CP/M's Internal Features

Teaches you how to modify BIOS, use CP/M system calls in your own programs, and more! Excellent for those who have read *CP/M Primer* or who otherwise understand CP/M's outer-layer utilities. By Mitchell Waite. Approximately 160pages, 8x9½, comb. ©1983.....$18.95

The S-100 and Other Micro Buses (2nd Edition)

Examines microcomputer bus syestems in general and 21 of the most popular systems in particular, including the S-100. Helps you expand your computer system through a better understanding of what each bus includes and how you can interface one bus with another. By Elmer C. Poe and James C. Goodwin, II. 208 pages, 5½x8½, soft. ©1981$9.95

Interfacing & Scientific Data Communications Experiments

This book introduces you to the principles involved in transferring data using the asynchronous serial data-transfer technique. It focuses on using the universal asynchronous receiver/transmitter (UART) chip in order to help your understanding of communication chips. Explores operation of teletype-writer interfaces and serial transmission circuits. With experiments and circuit details. By Peter R. Rony. 160 pages, 5½x8½, soft. ©1979.....$7.95

Active-Filter Cookbook

A practical discussion of the many active-filter types and uses, written by one of Sams' most popular authors. Teaches you how to construct filters of all types, including high-pass, low-pass, and bandpass having Bessel, Chebyshev, or Butterworth response. Easy to understand—no advanced math or obscure theory. Can also be used as a reference book for analysis and synthesis techniques for active-filter specialists. By Don Lancaster. 240 pages, 5½x8½, soft. ©1975.....$14.95

Regulated Power Supplies (3rd Edition)

Newest, most comprehensive discussion you'll find of regulated power supplies, including their internal architecture and operation. Thoroughly explains how to use regulation in your designs and projects when the need arises, and discusses practical circuitry and components. A valuable book for any technician or engineer involved in servicing or design. By Irving M. Gottlieb. 424 pages, 5½x8½, soft. ©1981.....$19.95

TTL Cookbook

Popular Sams author Dan Lancaster gives you a complete look at TTL logic circuits, the most inexpensive, most widely applicable form of electronic logic. In no-nonsense language, he spells out just what TTL is, how it works, and how you can use it. Many practical TTL applications are examined, including digital counters, electronic stopwatches, digital voltmeters, and digital tachometers. By Don Lancaster. 336 pages, 6½x8½, soft. ©1974.....$12.95

IC Op-Amp Cookbook

An informal, easy-to-read guide covering basic op-amp theory in detail, with 200 practical, illustrated circuit applications to reflect the most recent technology. JFET and MOSFET units are shown in both single and multiple formats. Includes manufacturers' data sheets, and lists addresses of the companies whose products are featured. By Walter G Jung. 480 pages, 5½x8½, soft. ©1980.....$15.95

IC Converter Cookbook

Discusses and explains data conversion fundamentals, hardware, and peripherals. A valuable guide to help you understand and use d/a and a/d converter applications. Includes manufacturers' data sheets. By Walter G. Jung. 576 pages, 5½x8½, soft. ©1978....$14.95

IC Timer Cookbook (Second Edition)

Learn more ways to use the IC timer in this easy to use second edition that includes many new IC devices with ready to use applications in practical, working circuits. All circuits and component relationships are fully defined and discussed for clarity. By Walter C. Jung. 384 pages, 5½x8½, soft. ©1983.....$17.95

The Programmer's CP/M Handbook

An exhaustive coverage of CP/M-80®, its internal structure and major components is presented. Written for the programmer, this volume includes subroutine examples for each of the CP/M system calls and information on how to customize CP/M—complete with detailed source codes for all examples. A dozen utility programs are shown with heavily annotated C-language source codes. An invaluable and comprehensive tool for the serious programmer. By Andy Johnson-Laird. 750 pages, 7½x9¼, softbound.....$21.95

Interfacing to S-100 (IEEE 696) Microcomputers

This book is a must if you want to design a custom interface between an S-100 microcomputer and almost any type of peripheral device. Mechanical and electrical design is covered, along with logical and electrical relationships, bus interconnections and more. By Sol Libes and Mark Garetz, 322 pages, 6½x9¼, softbound.....$16.95

Microprocessors for Measurement and Control

You'll learn to design mechanical and process equipment using microprocessor-based "real time" computer systems. This book presents plans for prototype systems which allow even those unfamiliar with machine or assembly language to initiate projects. By D.M. Auslander and P. Sagues, 310 pages, 7 3/8x9 1/4, softbound.....$15.99

Osborne CP/M® User Guide (Second Edition)

A new revised edition which includes expanded sections on CP/M® 86 and CP/M® 80, as well as CP/M® 's relationship to assembly language programming, MP/M® and CP/NET® operating environments. By Thom Hogan, 292 pages, 6½x9¼, softbound.....$15.95.

Discover FORTH

Whether you are a beginner seeking information on this multi-faceted programming language or a serious programmer already using FORTH, this book is a reference that should not be overlooked. Long considered a computer language of building blocks, FORTH has been optimized for speed and requires little computer support. By Thom Hogan, 146 pages, 6¼x9¼, softbound.....$16.95

68000 Assembly Language Programming

Each of the 68000's instructions is individually presented and fully explained in this assembly language tutorial. For experienced programmers, this book is also a complete reference to the 68000 instruction set and programming techniques. By Lance A Leventhal, 614 pages, 6½x9¼, softbound.....$18.95

Z8000® Assembly Language Programming

This book is filled with real-world programming examples, sample problems, and troubleshooting hints that will guide the reader to mastery of this powerful new 16-bit "super chip". The entire Z8000® instruction set is described in detail. By Lance A Leventhal, Adam Osborne, and Chuck Collins. 928 pages, 6½x9¼, softbound.....$19.99

The 8086 Book

Anyone using, designing, or simply interested in an 8086-based system will be delighted by this book's scope and authority. As the 16-bit microprocessor gains wider inclusion in small computers, this book becomes invaluable as a reference tool which covers the

timing, architecture and design of the 8086, as well as optimal programming techniques, interfacing, special features, and more. By Russell Rector and George Alexy, 624 pages, 6½x9¼, softbound.....$16.99.

Z80® Assembly Language Programming

Programming examples illustrate software development concepts and actual assembly language usage. More than 80 sample programming problems with solutions and a complete Z80® instruction set reference table. By Lance A. Leventhal, 640 pages, 6½x9¼, softbound.....$18.95.

8080A/8085 Assembly Language Programming

More quality programming examples and instruction sets than can be found in any other book on the subject. Information on assemblers, program loops, code conversion and more. A must for 8080A/80805 programmers. By Lance A Leventhal, 448 pages, 6½x9¼, softbound.....$18.95

Microprocessor-Based Robotics

Introduces you to robotics—a dynamic new field of science that uses your computing and electronic talents as well as your mechanical and electrical knowledge. First, you'll learn the mechanics of robot hands, arms, and legs; then, tactile sensing, motion and attitude sensing, and vision systems. After that, you learn controlling with microprocessors and BASIC programs, and finally, you learn to control the entire robot system with voice commands! Fascinating and *not* machine specific. By Mark J. Robillard, 224 pages, 8½x11, softbound.....$16.95

TV Typewritter Cookbook

Shows you how to quickly and easily project words and pictures from a common, microprocessor-based system onto an ordinary TV set. You'll be introduced to TVT communications by best-selling author Don Lancaster, who discusses basic TVT system design, memory types, interface circuitry, hard-copy output, and color graphics. By Don Lancaster, 256 pages, 5½x8½, softbound.....$12.95

Microcomputer Math

A step-by-step introduction to binary, octal, and hexidecimal numbers, and arithmetic operations on all types of microcomputers. Excellent for serious BASIC beginners as well as assembly-language programmers. Treats addition and subtraction of binary, multiple-precision and floating-point operations, fractions and scaling, flag bits, and more. Many practical examples and self-tests. By William Barden, 160 pages, 5½x8½, softbound$11.95

Understanding Digital Logic Circuits

A working handbook for service technicians and others who need to know more about digital electronics in radio, television, audio, or related areas of electronic troubleshooting and repair. You're given an overview of the anatomy of digital-logic diagrams and introduced to the many commercial IC packages on the market. By Robert G. Middleton, 392 pages, 5½x8½, softbound.....$18.95.

CMOS Cookbook

One of the best-selling technical books on the market, this cookbook gives you a solid understanding of CMOS technology and its application to real-world circuitry. Explains how CMOS differs from other MOS designs, how it's powered, and what its advantages are over other constructions. The final chapter shows you how to put all preceding information to work constructing several large-scale, working instruments. Includes a mini-catalog of more than 100 devices, with pinouts and application notes. By Don Lancaster, 416 pages, 5½x8½, softbound.....$13.95

Real Time Programming: Neglected Topics

This book presents an original approach to the terms, skills, and standard hardware devices needed to connect a computer to numerous peripheral devices. It distills technical knowledge used by hobbyists and computer scientists alike to useable, comprehensible methods. It explains such computer and electronics concepts as simple and hierarchical interrupts, ports, PIAs, timers, converters, the sampling theorem, digital filters, closed loop control systems, multiplexing, buses, communication, and distributed computer systems. By Caxton C. Foster, 190 pages, 6¼x9¼, softbound.....$9.95

Interfacing Microcomputers to the Real World

Here is a complete guide for using a microcomputer to computerize the home, office, or laboratory. It shows how to design and build the interfaces necessary to connect a microcomputer to real-world devices. With this book, microcomputers can be programmed to provide fast, accurate monitoring and control of virtually all electronic functions—from controlling houselights, thermostats, sensors, and switches, to operating motors, keyboards, and displays. This book is based on both the hardware and software principles of the Z80 microprocessor (found in several minicomputers, Tandy Corporation's famous TRS-80, and others). By Murray Sargent III and Richard Shoemaker, 288 pages, 6¼x9¼, softbound.....$15.55

Mastering CP/M

Now you can use CP/M to do more than just copy files. For CP/M users or systems programmers—this book takes up where our CP/M handbook leaves off. It will give you an in-depth understanding of the CP/M modules such as, CCP (Console Command Processor), BIOS (Basic Input/Output System), and BDOS (Basic Disk Operating System). Find out how to: incorporate additional peripherals with your system, use console I/O, use the file control block and much more. This book includes a specal feature—a library of useful macros. A comprehensive set of appendices is included as a practical reference tool. Take advantage of the versatility of your operating system! By Alan R. Miller, 398 pages, 6"x9", softbound.....$16.95

Designing With the 8088 Microprocessor

If you're searching for an introduction to software and hardware design using the 8088, you will find this book to be indispensable. After describing the 8088, the author leads you step-by-step through the complete system design of a powerful, yet inexpensive, single-board controller. This single-board controller design illustrates a typical application of the 8088, using 64K-bit dynamic RAMs, the 8203 VLSI dyncamic RAM controller, and state-of-the-art peripherals such as the 8254 programmable interval timer and the 8274 multi-protocol serial controller. Controller software design is also described in detail—from the basic assembly language test programs to high-level language interrupt control procedures. In addition, the 80188 software and hardware enhancements are briefly described. By John Zarrella, 304 pages, 6"x9", softbound.....$19.95

FORTH Tools, Volume One

FORTH Tools is a comprehensive introduction to the new international FORTH-83 Standard and all its extensions. It gives careful treatment to the CREATE-DOES construct, which is used to extend the language through new classes of intelligent data structures. FORTH Tools gives the reader an in-depth view of input and output, from reading the input stream to writing a simple mailing list program. Each topic is presented with practical examples and numerous illustrations. Problems (and solutions) are provided at the end of each chapter. FORTH Tools is the required textbook for the UCLA and IC Berkeley extension courses on FORTH. By Anita Anderson and Martin Tracy, 218 pages, 5¼x8¼, softbound.....$20.00

The Computer Journal

PO Box 1697 Kalispell, MT 59903

Order Date:_____

Print Name_____

Address_____

City_____ State_____ Zip_____

☐Check ☐Mastercard ☐Visa

Card No._____ Expires_____

Signature for Charge_____

Qty	Title	Price	Total

Shipping charges are: $1.00 for the first book, and $.50 for all subsequent books. Please allow 4 weeks for delivery.

Book Total

Shipping

TOTAL

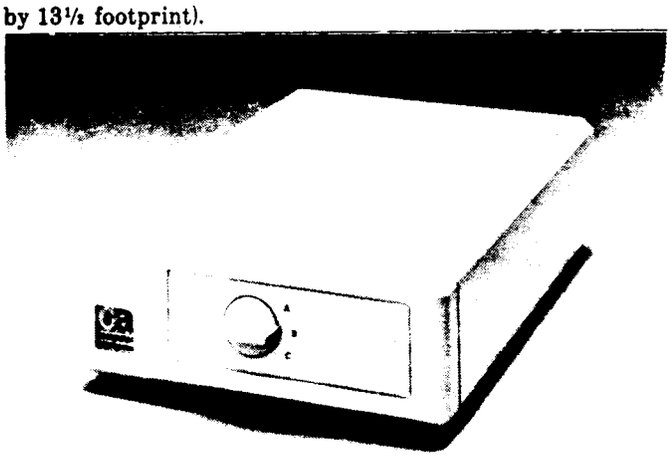
New Products

Port Expanders from Computer Accessories Corporation

Computer Accessories Corporation (San Diego) now offers a way to avoid the cost of added slotware when additional devices need to be connected to a computer I/O (input/output) port. The answer is a series of "port traffic cops"—port expanders with A/B/C switch routing— that can switch one port to any of three devices, or one device to any of three ports. Six Data Director models are available: three in stand-alone cabinets, available for $199 each (suggested US resale), and three designed to fit inside Computer Accessories' Model P12 Power Director® (line-conditioning power control accessory) cabinet, available for $189 each.

Three models in each series each meet a specific connector need for a given port or system. For standard RS-232 serial ports, Data Director offers DB25 female (models Q13/Q23) or male (models Q14/Q24) connectors. For standard parallel (printer) ports, Data Director (models Q15/Q25 offers 36-pin Centronics-stlye connectors.

Data Directors boast printed curcuit board construction (which is more reliable than point-to-point wiring with individual wires) and use sealed rotary switches (to avoid data errors from the effects of contamination on switch contact integrity). Each model is fully shielded, and a full line of supportins cables is available from Computer Accessories. A P12 Power Director complete with a Data Director is available for $388, or alone for $199. It's designed to fit atop an IBM® PC system unit (or others with its 19½ by 13½ footprint).



For additional information, contact Computer Accessories Corporation, 7696 Formula Place, San Diego, CA 92121; (619) 695-3773. ■

Free Catalog From Group Technology

The Spring 1984 catalog contains descriptions of all of the new and current books and products available from Group Technology, Ltd. It is directed particularly to those who wish to gain or teach hands-on experience in interfacing external devices to microcomputers. The experiment-based books that comprise the world-renowned Blacksburg Series provide practical guidance in acquiring these skills. Topics covered include the major 8-bit and 16-bit microprocessors, analog and digital electronics, electronic data processing, robotics, circuit design, machine design, electronic music and speech synthesis, fiber optics, microcomputers in astronomy, FORTH language, and more. Hardware described includes interfacing and circuit design boards for the Apple II, Timex/Sinclair, Commodore 64 and Vic-20, TRS-80 Models I, III, 4, and Color Computer. Software listed includes utility programs for the TRS-80; scientific software for the Apple II + /e for a variety of curvefitting and parameter estimation problems; and a LISP interpreter for the IBM PC.

Both novice and experienced microcomputer enthusiasts will find a treasury of practical resource material in the catalog. It is available without charge from Group Technology, Ltd., PO Box 87, Check, VA 24072, tel. 703-651-3153. ■

Computer Nostalgia:

How Personal Computers Have Changed

When micros first appeared, in the middle 1970s, memory was precious. Microsoft made a 4K BASIC (that's how much memory it occupied) and there was even a smaller version, known as "Tiny BASIC," which would run successfully in computers having less than 4K of total memory. Many now-familiar features were sadly missing; versions with only integer numbers and no arrays at all were not unusual.

Nowadays, BASIC may occupy 16K or so of ROM, or be loaded from disk into 32K or more of your available memory. It usually comes free when you buy your computer.

Altair (the first mass-marketed micro) in 1975 advertised 1K memory boards for $139, 4K boards for $338, and a single disk drive for $1980. The 4K BASIC was $350, the 8K version was $500. You could buy the source listing of BASIC (in case you wanted to modify it?) for $3000.

The Altair 8800, the original "S-100" computer, was sold in 1975 for $439 in kit form or $621 assembled. It came (in either version) with 4 slots and two boards. One board was the CPU; the other was a ¼K (that's right, one fourth of a K, or 256 bytes) board that could be expanded all the way up to a full 1K by adding six additional memory chips! Kits were available to add more S-100 slots; if you added enough you could cram in enough memory to run one of those early versions of BASIC. ■

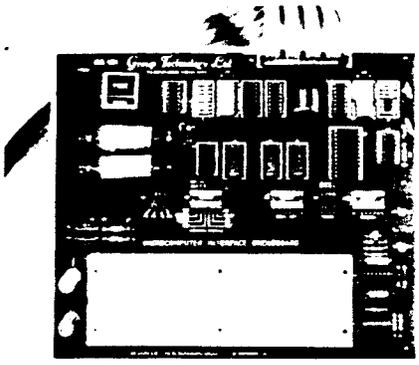
LEARN MICROCOMPUTER INTERFACING
VISUALIZE SCIENCE PRINCIPLES

Using GROUP TECHNOLOGY BREADBOARDS with your APPLE® ...COMMODORE 64® ...TRS-80® ...TIMEX-SINCLAIR® ...VIC-20®

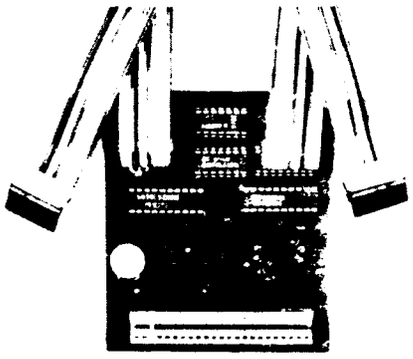
Versatile breadboards and clearly written texts with detailed experiments provide basic instruction in interfacing microcomputers to external devices for control and information exchange. They can be used to provide vivid illustrations of science principles or to design interface circuits for specific applications. Fully buffered address, data, and control buses assure safe access to decoded addresses. Signals brought out to the breadboards let you see how microcomputer signals flow and how they can be used under BASIC program control to accomplish many useful tasks.

Texts for these breadboards have been written by experienced scientists and instructors well-versed in conveying ideas clearly and simply. They proceed step-by-step from initial concepts to advanced constructions and are equally useful for classroom or individual instruction. No previous knowledge of electronics is assumed, but the ability to program in BASIC is important.

The breadboards are available as kits or assembled. Experiment component packages include most of the parts needed to do the experiments in the books. Connecting cables and other accessory and design aids available make for additional convenience in applying the boards for classroom and circuit design objectives. Breadboard prices range from $34.95 to $350.00



The INNOVATOR® BG-Boards designed by the producers of the highly acclaimed Blacksburg Series of books have gained wide acceptance for teaching microcomputer interfacing as well as for industrial and personal applications. Detailed, step-by-step instructions guide the user from the construction of device address decoders and input/output ports to the generation of voltage and current signals for controlling servo motors and driving high-current, high-voltage loads. BG-Boards are available for the Apple II, II+, IIe; Commodore 64 and VIC-20; TRS-80 Model 1 with Level II BASIC and at least 4K read/write memory, Models III and 4. The books, *Apple Interfacing* (No. 21862) and *TRS-80 Interfacing Books 1 and 2* (21633, 21739) are available separately.



The FD-ZX1 I/O board provides access to the Timex-Sinclair microcomputer for use in automated measurement, data acquisition, and instrument control applications. A number of science experiments have been developed to aid teachers in illustrating scientific principles. The operating manual contains instructions for constructing input and output ports. A complete text of the experiments will be available later in 1984. The FD-ZX1 can be used with Models 1000, 1500, 2068, ZX81, and Spectrum.

The Color Computer Expansion Connector Breadboard (not shown) for the **TRS-80 Color Computer** makes it possible to connect external devices to the expansion connector signals of the computer. Combined with a solderless breadboard and the book *TRS-80 Color Computer Interfacing, With Experiments* (No. 21893), it forms our Model CoCo-100 Interface Breadboard providing basic interfacing instructions for this versatile computer. Experiments in the book show how to construct and use a peripheral interface adapter interface, how to input and output data, and how digital-to-analog and analog-to-digital conversion is performed.

Our new Spring Catalog describes the interface breadboards, dozens of books on microcomputer interfacing, programming, and related topics including the famous Blacksburg Continuing Education Series, a resource handbook for microcomputers in education, and a comprehensive guide to educational software; utility software for the TRS-80, scientific software for the Apple II, and other topics. We give special discounts to educational institutions and instructors. Write for the catalog today.

PUTTING HANDS AND MINDS TOGETHER

Group Technology, Ltd.
P.O. Box 87N
Check, VA 24072
703-651-3153

The Computerist's Calendar

June 07C0

SUN	MON	TUES	WED	THURS	FRI	SAT
					1	2
3	4	5	6	7	8	9
A	B	C	D	E	F	10
11	12	13	14	15	16	17
18	19	1A	1B	1C	1D	1E