ISSN # 0746-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Vol. 2 No. 9 Issue Number 13 $2.50 US

Editor's Page

Turmoil In The Computer Publishing Market

There has been a lot of conjecture about a drastic shakeout in the microcomputer hardware and software fields, and apparently the publishing field is not immune from this danger.

Until the early 1980's the majority of the people buying micros were technically oriented because they had to configure and interface the computers which were then available. At that time, a few magazines were started by technically involved individuals to provide the information needed by these pioneers. As micros became accepted as a legitimate tool by the business community, people rushed to purchase appliance type micros for the office, and the number of micros in use increased very rapidly. The microcomputer industry saw the fantastic quantities of micros being sold as an almost unlimited opportunity to sell *anything* related to micros to this inexperienced audience. While the hardware and software people rushed poorly conceived and untested products to market, the publishers forgot the basic fundamentals of market analysis and envisioned expanding their magazine circulation to a million subscribers, and publishing books with sales in the millions.

The point that the publishers failed to recognize was that the people in this vast new market were buying appliance type computers with the intention of plugging them in and using them for business applications. **These people are not interested in how a computer works, they are just buying a tool to use.** As a comparison, consider the television. Almost everybody has a TV set, but most people do not buy books or magazines on how the TV works, or how to build their own antenna, or how to repair the set, or even how to adjust the set. A few technically oriented people are interested in these topics, but most people are satisfied with a TV program listing which they usually get in their newspaper. They expect to just plug it in and have it work without having to go back to school in order to learn how to use it! The majority of the people buying micros today feel the same way about their computer. They expect to plug it in and use it without having to refer to a book—and this trend will become even more pronounced as the industry develops the user friendly systems which these people insist on.

The sales of technical publications did not increase at the same rate as the increase of microcomputer sales, so many of the publishers expanded into general, non-technical subjects in order to penetrate this vast new market. Unfortunately, this expansion required additional capital to finance the expansion, and many of the previously individually owned or closely held businesses went to the venture capital market for the cash needed to expand. The people who gained control of the businesses were not technically oriented, but they could read the bottom line of the quarterly report and could calculate their potential financial gain from taking the business public and selling stock. However, rapid financial gain requires rapid growth, and they went for the big market by making their publications so general that they would appeal to everyone.

When the management of the magazines changed from the technically oriented originators to bottom-line-oriented businessmen who were unfamiliar with the market, they also made another very significant change. In order to maximize the short term profits, they changed the nature of the publications from a subscriber-driven magazine with a maximum of technical information for the reader, to an advertiser-driven magazine with the content designed to maximize the the amount of paid advertising.

The original market—the technically oriented people—got tired of reading reviews and rewritten advertising fluff, so they stopped buying. The huge general audience never was a market for a large quantity of books or magazines, so they never started buying. The temporary increase in circulation resulted in significant increases in the advertising page rate cost. The hardware and software industry is suffering from poor business practices, including overexpansion and mediocre "me-too products" with poor customer support. Many hardware and software suppliers are laying off their help or filing for bankruptcy, which means greatly reduced advertising expenditures and advertising bills which will never be paid. The result of all this is that a number of magazines have ceased publication during the last few months, and there are indications that several book publishers are in a difficult position. The next

continued on page 8

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.

Controlling the Apple Disk][Stepper Motor

by Jan Eugenides

One of the more interesting aspects of computing is the controlling of mechanical devices. These devices can be anything from burglar alarms to giant robot arms. In the Apple Disk II there is a small robot arm with one axis of movement, which is controlled from the Apple by a set of soft switches. This article will show you how to directly access and control the Disk II arm from assembly language.

The Stepper Motor

Inside the Apple disk drive, in fact inside of nearly all disk drives, is a device called a stepper motor. Unlike the usual electric motors we are used to, the stepper motor does not continuously spin, but instead moves in small steps according to the control voltages it receives. Hence, the name stepper motor. By properly controlling such a motor, it is possible to accurately position a mechanical arm connected to the motor. A lever system can be used to translate the twisting movement of the motor (horizontal movement of the arm in the case of a disk drive) to a straight line movement of some kind. By the proper use of several motors along different axes, very complex ranges of movement can be obtained and controlled by computer. Controlling the arm of the Apple Disk II is a good place to begin to experiment along these lines.

The Soft Switches

Most of the hard stuff is taken care of for you by the Apple circuitry, and direct access to the disk drive can be easily obtained from assembly language. This is accomplished through a series of soft switches inside the Apple itself. These switches are set up as memory locations which, when accessed in any way, toggle according to a preset design. See Figure 1 for a table of these addresses.

The addresses in the table are the base addresses for a theoretical drive in slot 0. The actual addresses are slot dependent, according to the following formula:

Actual Address = Base Address + (Slot × 16).

The usual way of handling this from assembly is to load the x-register with the slot times 16, and use it as an offset from the base address. For example, to turn on a drive in slot six,

```
LDX #$60
LDA $C089,X
```

Phases

Once the drive is on, the disk will begin spinning. Now the drive is "live" and the stepper motor can be directly controlled. In order to move the arm, it is necessary to turn on and off the four phases of the stepper motor. This is done by sequentially accessing the proper soft switches. When the phases are accessed in ascending order, the arm moves inward. When they are accessed in descending order, the arm moves outward. Each phase actually moves the arm one-half track. Thus it is possible to move the arm between the normal 35 tracks on the Apple Disk II. However, it is not possible to write between tracks, because the read/write head is too wide, and you would wind up destroying data on both sides. Some copy protection schemes use the half-track positioning to make their disks difficult to copy. As long as data is written at least one track apart, the half-track positions can be used.

The Arm Mover Program

Listing 1 is the source code for a program which demonstrates a method for controlling the disk arm from assembly language. It allows the user to input a track number in hex, and the program will move the arm to that track. With a little modification, you could add the ability to read any given track into memory, perhaps as raw data, and thereby examine the formatting of the disk. As it stands, the program is purely educational (but fun!).

The source is well-commented, and you will have no trouble following the flow. The main control routine is in lines 1560-1970. The rest of the program is user-interface stuff.

How to Use the Program

To use the program, once you have typed in the machine code from Listing 2, or assembled it from Listing 1, simply BRUN it. The drive will come on, and you will hear the familiar sound of the disk arm being recalibrated to track 0. The screen will clear and you will be asked which track you wish to move the arm to. Input a two-digit hex number

Base Address	Effect
$C080	Phase 0 off
$C081	Phase 0 on
$C082	Phase 1 off
$C083	Phase 1 on
$C084	Phase 2 off
$C085	Phase 2 on
$C086	Phase 3 off
$C087	Phase 3 on
$C088	Drive off
$C089	Drive on
$C08A	Select drive 1
$C08B	Select drive 2

Figure #1

between $00 and $23 (that's right, the Disk II will access 36 tracks, although DOS 3.3 does not use track $23), and the arm will move to that track. You may wish to remove the cover of the drive in order to more clearly see the movement. Notice that if you input a track such as $FF, the drive tries to move out that far, but of course it can't, so it continually bumps up against the stop. Don't worry, this won't hurt it a bit. After trying for track $FF, try track $00 again. The program thinks the drive was on track $FF, so it moves all the way in and then bumps repeatedly against the other stop. This, in fact, is the normal way of recalibrating the arm to track zero. In the first few lines of the program, you will notice we used this method, putting a large number ($30) in the current drive variable, and then pulling the arm back across $30 tracks, to be sure it starts at track 0. By the way, hitting the RETURN key instead of inputting a track number will exit the program and turn off the drive.

Possible Uses

Direct access of the disk drive is necessary for many programs, such as user-written operating systems, bit copiers, system-independent programs, etc. I am not personally into copy protecting my disks, however a fine knowledge of direct disk access would certainly be useful there.

We have not gone into the actual reading or writing of the disk in this article. That will have to wait for a future issue. If you find this as interesting as I do, you might consider disassembling DOS from $B9A0 to $BA28. Those are the addresses of the DOS 3.3 arm move routines, which use a slightly different method to calculate the phases. Any questions you have may be directed to me, care of *The Computer Journal*—I'll try to help if I can.

LISTING #1

```
1000 *S.TEST.ARM.MOVER2
1010 *
1020 GETLN   .EQ $FD6A
1030 CH    .EQ $24
1040 CV    .EQ $25
1050 CTR   .EQ $06
1060 INBUFF  .EQ $200
1070 IPTR    .EQ $08
1080 SLEN    .EQ $FF
1090 BASIC   .EQ $3D0
1100 COUT    .EQ $FDED
1110 PROMPT  .EQ $33
1120 HOME    .EQ $FC58
1130 TABV    .EQ $FB5B
1140 MON.DELAY  .EQ $FCA8
1150 PRBYTE  .EQ $FDDA
1160 *--Disk I/O
Selects----------------------------------------
1170 STEPMOTOR0  .EQ $C080  Step motor position
1180 STEPMOTOR1  .EQ $C081
1190 STEPMOTOR2  .EQ $C082
1200 STEPMOTOR4  .EQ $C084
1210 STEPMOTOR6  .EQ $C086
1220 DRIVEOFF    .EQ $C088  Turn drive off after 6 revs
1230 DRIVEON  .EQ $C089  Turn drive on
1240 DRIVE1.SELECT  .EQ $C08A  Select drive 1
1250 DRIVE2.SELECT  .EQ $C08B
1260
*---------------------------------------------------------
1270  .TF ARM.MOVER2
1280
*---------------------------------------------------------
1290 RESET.ARM LDA #$30    Put $30 in current track
1300  STA CURRENT.TRACK
1310  LDA #0   Set destination track to zero
1320  STA DESTINATION.TRACK
1330  LDX SLOT    Put slot * 16 in x-reg
1340  LDA DRIVE1.SELECT,X   Select drive 1
1350  LDA DRIVEON,X   Turn on drive
1360  JSR MOVE.ARM    Recalibrate arm to track zero
1370  LDX SLOT    Assure x-reg is correct
1380  LDA STEPMOTOR0,X   Turn off all phases
1390  LDA STEPMOTOR2,X
1400  LDA STEPMOTOR4,X
1410  LDA STEPMOTOR6,X
1420  JMP START    Skip data
1430
*---------------------------------------------------------
1440 CURRENT.TRACK   .HS 00
1450 DESTINATION.TRACK  .HS 00
1460 YOFFSET   .HS 00
1470 NUMBER.OF.TRACKS   .HS 00
1480 SLOT      .HS 60
1490
*---------------------------------------------------------
1500 * User interface
1510
*---------------------------------------------------------
1520 START  JSR SCREEN  Clear screen and print message
1530  JSR MOVE.ARM   Move arm to selected track
1540  JMP START   Do it again
1550 *--Arm move
routine----------------------------------------
1560 MOVE.ARM LDA #0
1570  STA YOFFSET  Initialize offset
1580  LDA CURRENT.TRACK   Get current track
1590  SEC
1600  SBC DESTINATION.TRACK   Subtract destination track
1610  BEQ .4       If equal, then we're done
1620  BCS .1       Continue if result positive
1630  EOR #$FF     otherwise, make it positive
1640  ADC #1
1650 .1  STA NUMBER.OF.TRACKS Save number of tracks to move
1660  ROL YOFFSET    Set offset for in or out movement
1670  LSR CURRENT.TRACK    Is track even or odd?
1680  ROL YOFFSET    Set offset for even or odd
1690  ASL YOFFSET    Adjust value for table offset
1700  LDY YOFFSET    Put offset in y-reg
1710 .2  LDA PHASE.TABLE,Y   Get phase to turn on
1720  JSR DO.PHASE       Do it
1730  LDA PHASE.TABLE+1,Y    Two phases required for each
track,
1740  JSR DO.PHASE       Do the next one
1750  TYA
1760  EOR #$02    Change offset for next phase
1770  TAY
1780  DEC NUMBER.OF.TRACKS  Decrement number of tracks to
move
1790  LDA NUMBER.OF.TRACKS
1800  BNE .2    Not done, do another
1810  LDA DESTINATION.TRACK Update current track
1820  STA CURRENT.TRACK
1830 .4  RTS        All done
1840 *---Turn on a phase, delay 20 ms, then turn it off
again---
1850 DO.PHASE  ORA SLOT  Add slot to phase
1860  TAX    Put it in x-reg
1870  LDA STEPMOTOR1,X  Turn on a phase
1880  JSR WAIT    Delay is necessary while arm moves
1890  LDA STEPMOTOR0,X  Turn off a phase
1900  RTS
1910 *--20-ms delay
routine----------------------------------------
1920 WAIT   LDA #$56
1930  JSR MON.DELAY  Use montior's delay routine
1940  RTS
1950 *--Phase
table------------------------------------------
1960 PHASE.TABLE  .HS 02.04.06.00  Move arm inward
1970     .HS 06.04.02.00  Move arm outward
1980 *-----------------------------------
1990 *GET INPUT
2000 *-----------------------------------
2010 SCREEN JSR HOME
2020  LDA #5
2030  JSR TABV    Vtab 5
2040  JSR PRINT   Print message
2050  .AS -"TRACK TO MOVE ARM TO"
2060  .HS 00
2070  JSR INPUT   Get response
2080  LDA SLEN    Length of string
2090  CMP #2   If longer than 2, it's wrong
2100  BNE ERROR   Print error message
2110  JSR CONVERT.TO.HEX   Convert to hex data
2120 *
2130  RTS
2140 *
2150 *--CONVERT TO HEX AND STORE------
2160 CONVERT.TO.HEX
2170  LDX #1
2180  LDA IPBUF,X    Get the first ascii byte
2190  JSR ASCII.TO.HEX  Convert it to hex
2200  STA DESTINATION.TRACK   Store it
2210  DEX       Get the other ascii byte
2220  LDA IPBUF,X
2230  JSR ASCII.TO.HEX  Convert to hex
2240  ASL       Shift it to proper nibble
2250  ASL
```

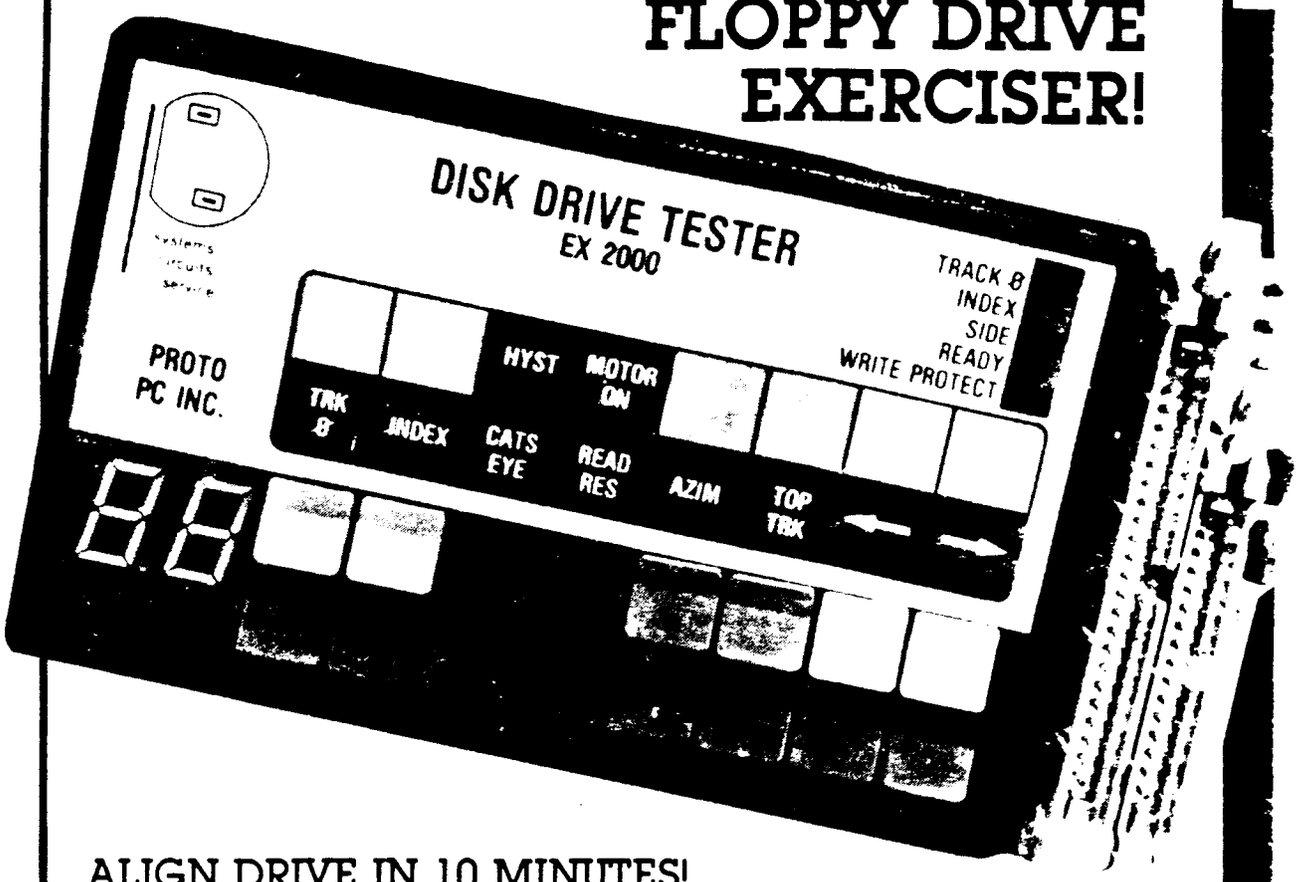
```
2260  ASL
2270  ASL
2280  ORA DESTINATION.TRACK   Combine with hi nibble
2290  STA DESTINATION.TRACK   Final result
2300  RTS
2310 *--------------------------------
2320 ASCII.TO.HEX  SEC
2330  SBC #$B0
2340  CMP #10
2350  BCC A2HEX1   Must be 0-9, so we're done
2360  SBC #7     Must be A-F, so adjust value
2370 A2HEX1 RTS
2380 *--ERROR-----------------------------
2390 ERROR  JSR ERR   Print error message
2400  JSR DELAY Wait a bit
2410  JMP STAR  Start over
2420 ERR JSR PRINT
2430  .HS 8D
2440  .AS -"TRACKS MUST BE 2 HEX DIGITS"
2450  .HS 8D8D
2460  .AS -"ENTER LEADING ZEROES IF NECESSARY"
2470  .HS 8D00
2480  RTS
2490 *--DELAY-----------------------------
2500 DELAY  LDX #$10
2510 .1  LDA #$FF
2520  JSR MON.DELAY
2530  DEX
2540  BNE .1
2550  RTS
2560 *--------------------------------
2570 *INPUT
2580 *--------------------------------
2590 INPUT  LDA #$BF
2600  STA PROMPT  Put question mark in prompt
2610  LDA #IPBUF  Store in buffer
2620  STA IPTR
2630  LDA /IPBUF
2640  STA IPTR+1
2650  LDX #0
2660  JSR GETLN  Get a line of input
2670  STX SLEN   Save length
2680  TXA     X holds length of line
2690  BEQ EXIT   If <CR>, just exit
2700  TAY
2710  LDA #$0
2720  STA INBUFF,Y  Put zero at end of line
2730 .2  LDA INBUFF,Y
2740  STA (IPTR),Y  Move to buffer for safekeeping
2750  DEY
2760  CPY #$FF
2770  BNE .2
2780  RTS
2790 EXIT   LDX SLOT
2800  LDA DRIVEOFF,X Turn off drive
2810  JMP BASIC   Return to basic
2820 *--------------------------------
2830 *PRINT ROUTINE
2840 *
2850 *--------------------------------
2860 PRINT  PLA     Get address
2870  STA CTR
2880  PLA
2890  STA CTR+1
2900  LDY #$01
2910 .1  LDA (CTR),Y   Get data to print
2920  BEQ .3     Zero? Then end
2930  JSR COUT   Print character
2940  INY
2950  BNE .1
2960 .3  CLC     Return to next instruction after data
2970  TYA
2980  ADC CTR
2990  STA CTR
3000  LDA CTR+1
3010  ADC #$00
3020  PHA
3030  LDA CTR
3040  PHA
3050  RTS
3060 IPBUF   .BS 3
3070  .LIST OFF
```

LISTING #2

```
0800: A9 30 8D 28 08 A9 00 8D
0808: 29 08 AE 2C A0 BD 8A C0
0810: BD 89 C0 20 36 08 AE 2C
0818: 08 BD 80 C0 BD 82 C0 BD
0820: 84 C0 BD 86 C0 4C 2D 08
0828: 00 00 00 00 60 20 97 08
0830: 20 36 08 4C 2D 08 A9 00
0838: 8D 2A 08 AD 28 08 38 ED
0840: 29 08 F0 36 B0 04 49 FF
0848: 69 01 8D 2B 08 2E 2A 08
0850: 4E 28 08 2E 2A 08 0E 2A
0858: 08 AC 2A 08 B9 8F 08 20
0860: 7B 08 B9 90 08 20 7B 08
0868: 98 49 02 A8 CE 2B 08 AD
0870: 2B 08 D0 E8 AD 29 08 8D
0878: 28 08 60 0D 2C 08 AA BD
0880: 81 C0 20 89 08 BD 80 C0
0888: 60 A9 56 20 A8 FC 60 02
0890: 04 06 00 06 04 02 00 20
0898: 58 FC A9 05 20 5B FB 20
08A0: 74 09 D4 D2 C1 C3 CB A0
08A8: D4 CF A0 CD CF D6 C5 A0
08B0: C1 D2 CD A0 D4 CF 00 20
08B8: 44 09 A5 FF C9 02 D0 2B
08C0: 20 C4 08 60 A2 01 BD 95
08C8: 09 20 E1 08 8D 29 08 CA
08D0: BD 95 09 20 E1 08 0A 0A
08D8: 0A 0A 0D 29 08 8D 29 08
08E0: 60 38 E9 B0 C9 0A 90 02
08E8: E9 07 60 20 F4 08 20 39
08F0: 09 4C 2D 08 20 74 09 8D
08F8: D4 D2 C1 C3 CB D3 A0 CD
0900: D5 D3 D4 A0 C2 C5 A0 B2
0908: A0 C8 C5 D8 A0 C4 C9 C7
0910: C9 D4 D3 8D 8D C5 CE D4
0918: C5 D2 A0 CC C5 C1 C4 C9
0920: CE C7 A0 DA C5 D2 CF C5
0928: D3 A0 C9 C6 A0 CE C5 C3
0930: C5 D3 D3 C1 D2 D9 8D 00
0938: 60 A2 10 A9 FF 20 A8 FC
0940: CA D0 F8 60 A9 BF 85 33
0948: A9 95 85 08 A9 09 85 09
0950: A2 00 20 6A FD 86 FF 8A
0958: F0 11 A8 A9 00 99 00 02
0960: B9 00 02 91 08 88 C0 FF
0968: D0 F6 60 AE 2C 08 BD 88
0970: C0 4C D0 03 68 85 06 68
0978: 85 07 A0 01 B1 06 F0 06
0980: 20 ED FD C8 D0 F6 18 98
0988: 65 06 85 06 A5 07 69 00
0990: 48 A5 06 48 60 00 00 00
0998: 00 00 00 00 00 D4
```

■

FLOPPY DRIVE EXERCISER!



ALIGN DRIVE IN 10 MINUTES!

Use with scope and alignment disk (SS $49, DS $75)

- SINGLE KEYSTROKE FOR ALL ALIGNMENT TRACKS
- JOG KEYS-MOVE TO ANY TRACK
- INCLUDES "OSBORNE" TYPE POWER HOOKUP
- RUNS ANY STANDARD 34 PIN (5") OR 50 PIN (8") DRIVE
- SHOWS SPEED AND SPEED AVERAGE!
- HYSTERESIS CHECK BUILT IN
- SELECT 5" 48, 96, 100 TPI, OR 8" 48, TPI
- POWER "Y" CABLE=$10 DRIVE DATA CABLE=$20

USED BY: IBM, ARMY, NAVY, RCA, ETC...

EX 2000 $299

FREE Air Freight on Prepaid Orders. COD: Add $5 Plus Shipping

PROTO PC inc. CALL NOW! 612-644-4660

2439 Franklin, St. Paul, MN 55114

Interfacing Tips and Troubles

A Column by Neil Bungard

Interfacing the Sinclair Computers Part One

Bryan Lepkowski of Mechanicville, New York wrote in and wanted to know what changes would be required to adapt the VIC-20 EPROM programmer (*The Computer Journal* Vol.II, No.4) to operate with the Sinclair TS2068 color computer. As I attempted to answer his question, I realized two things. One, the question cannot be answered in a single statement and two, a large number of the Sinclair machines (which include the TS1000, TS1500, Spectrum, and the TS2068) have been sold to people like yourselves who would like to use them in an interface application. These two realizations have prompted me to address the subject of Sinclair computer interfacing in the next two installments of "Interfacing Tips and Troubles." This month we will look at the hardware and list the idiosyncrasies involved in interfacing the Sinclair computers. Next month we will explore the software necessary to complete the interface.

Every computer manufacturer's design creates "uniquenesses" in the way their particular machine operates. On the "user" level the uniquenesses remain transparent because of industry standards like the RS-232-C serial standard, Centronics parallel standard, IBM disk format, CP/M disk operating system, etc. But on the "hacker/designer" level the uniquenesses become painfully obvious.

The Sinclair designs are abundant with uniquenesses, and until they are all identified, interfacing them can be a real headache. Fortunately, I have paid my Sinclair interfacing initiation fees (about 100 hours of hair pulling investigation), and I have successfully connected the Sinclair machines to several circuits. In this two-part article I will explain the idiosyncrasies involved in interfacing the Sinclair machines and demonstrate general interfacing practices using the Sinclair computers. As far as interfacing the Sinclair family is concerned, we will assume that there are no hardware differences between the TS1000, TS1500, SPECTRUM, and TS2068. For the software, however, we will split the machines into two categories. The SPECTRUM and the TS2068 are color computers and constitute the first category. The TS1000 and the TS1500 are black and white models which constitute the second category. The actual software differences between these two categories of Sinclair computers will be discussed later.

Listing The Sinclair Computers' Uniquenesses

To begin with, data can be moved into or out of a computer via two methods; memory mapped input/output (MMI/O) and accumulator input/output (AI/O). The type of input/output supported by a computer depends primarily upon the type of central processing unit (CPU) that it uses. In general, the 6800 & 6500 CPU families support only MMI/O and the 8080, Z80, and 8086 CPU families support both MMI/O and AI/O. The Sinclair family of computers uses the Z80 CPU, so technically these machines should support both MMI/O and AI/O. Prepare to start your list of uniquenesses—the Sinclair machines do not support MMI/O. Normally the Z80 CPU does, but because of the way that the address bus is decoded by the Sinclair computers, MMI/O is not possible. This leaves AI/O as the only means of transferring data, which leads us to the second uniqueness... The Sinclair hardware will support AI/O, but no commands have been included in the Sinclair BASIC instruction set to accomplish an AI/O. This means that the input/output routines must be written in Z80 CPU machine language, placed in RAM memory, and called as subroutines by the BASIC language programs. Fortunately Sinclair did provide this capability with the BASIC "USR" command, which I will discuss later.

Before we look at the software details involved in interfacing the Sinclair computers, let me add two hardware details to your list of system uniquenesses. First, if data is input using even device codes (e.g., IN 00H, IN 02H, IN 04H, etc.) the two highest order bits of the data bus (D6 & D7) will automatically be masked to zeroes. Note that the "H" after the above numbers refers to hexadecimal format. Using a "D" after a number means decimal format. This means that if you use even device codes, only six bits of the eight bit data bus are valid. Secondly, if data is output using odd device codes (e.g., OUT 01H, OUT 03H, OUT 05H, etc.) the computer will crash! The solution to these hardware problems is to always use odd input device codes and even output device codes. This still leaves 128 input and 128 output device codes to work with. The important thing is to keep these details in mind so that you can design around these limitations as you create interfaces for the Sinclair machines.

Armed with the above details, we are ready to look at the hardware and write the programs required to accomplish an interface using the Sinclair machines. Figure 1 shows a general hardware configuration which will allow data transfers between the Sinclair and the interface circuit. The two "OR" gates, the "AND" gate, and the 74LS138 form an

address decoder which is capable of generating eight device codes (from 00H to 07H). Using the above decoder configuration, four even device codes are generated (00H, 02H, 04H, 06H) which can be used for outputting data, and four odd device codes are generated (01H, 03H, 05H, 07H) which can be used for inputting data. As shown in Figure 1, the even decoder outputs are connected to latches which transfer data out of the computer, and the odd lines are connected to tristate devices which transfer information into the computer. As you can see, the hardware required to accomplish an interface is relatively straightforward. Unfortunately, there are software oddities which make the programming somewhat more involved.

Writing Programs For the Interface

The first task of writing software for the interface is to reserve "safe" space in the Sinclair's RAM memory to store the machine language routines. This is required because the Sinclair moves data around in its RAM memory as it executes a BASIC program, and if your machine language routines are not protected they could be overwritten. The task of reserving space for the machine language routines will be different, depending upon which computer you are using. For the TS1000 and the TS1500, reserving space can be accomplished in the first line of the BASIC program with the use of a REMARK statement. The REMARK statement

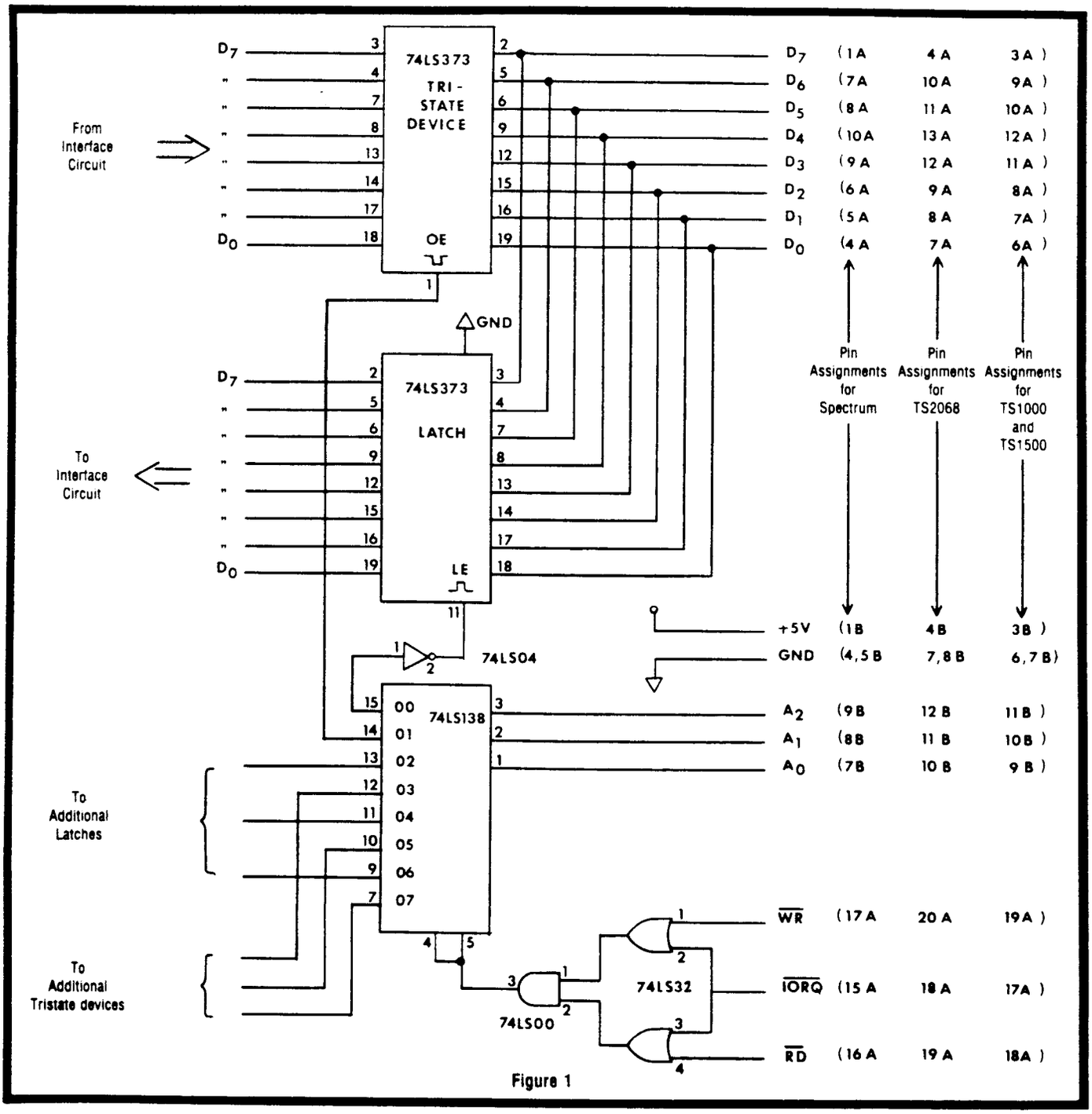


Figure 1

takes the following form:

1 REM 1234567890.

In the above statement, the space reserved for machine instructions is filled with the characters 1 through 0 following the REM statement. The TS1000 and the TS1500 begins storing BASIC instructions at memory location 16509 (D). Two bytes are used for the line number, two bytes for the line length, and one byte is used for the REM command code. This places the character "1" at memory location 16514 (D). The reserved space ends with the character "0", which resides in memory location 16523. If you need more space for machine instructions, you simply place more characters after the REM statement. If you want to know how much space you have reserved, count the characters after the REM statement. If you want to know the ending address of the reserved space, just add the number of characters after the REM statement to 16513. The reason for reserving space via the REM statement when using the TS1000 and TS1500 is because it is the only way to save machine language routines on a cassette tape. For the color computers (SPECTRUM and TS2068) however, memory space can be reserved above the BASIC command area and still be saved on cassette tape. Reserving space on the color computers is accomplished with the CLEAR command. The CLEAR command takes the following form:

1 CLEAR 32129.

The above command reserves memory from location 31130 (D) to the top of RAM memory. As mentioned before, this space is protected when BASIC executes, and it can be saved on cassette tape.Once you have reserved sufficient space for the machine language routines, you can then place the desired machine language instructions into the reserved space. This is accomplished by sequentially POKE-ing each machine instruction into the reserved area. Of course, if your machine language routines become large, this POKE-ing process can be tedious and error prone. In next month's "Interfacing Tips and Troubles," we will present a short BASIC routine which automatically POKES the machine language instructions for you. In addition to the automatic POKE-ing routine, we will explore the BASIC and machine language routines necessary to complete an interface using the Sinclair computers. ■

Editor's Page, continued

few months should be very interesting.

When I receive my copy of *InfoWorld*, the first thing I do is look for the latest news about who is laying off their help or going out of business. The list grows every week and I was startled to see that *Softalk* was one of the latest causalities. Some of the other recent drop outs are: *Micro Computing* (formerly Kilobaud), *PCjr*, *Peanut*, Compute's *PC* and *PCjr*, *St. Mac*, *Whole Earth Software Review*, *Color Computer*, *Computer Accessories & Peripherals*, *Atari Connection*, *St. Game*, and *Personal Software*. Another large magazine ran a total of 1,472 pages for the period July thru September of 1984 compared with 1,808 pages for the same period last year. That is a decrease of 336 pages, or about 19% for the three month period.

Not everyone danced to the same drummer, however, and there are publishers who appear to be in good condition. Some which I am familiar with are: *Nibble Magazine*, which contains very good assembly and BASIC programming for the Apple® , *Micro Cornucopia* with hardware and software information for the Big Board® , Kapro® , and Xerox 820® , and *Apple Assembly Line* with assembly language for the Apple (they also market the S-C assembler and word processor). I am sure that there are others, but these are ones with which I am familiar, and it is interesting to note that they are all controlled by the originator.

What does all this mean for *The Computer Journal?* It means that we intend to stick to our convictions of serving our readers and restricting our growth to that which we can finance without losing control to the venture capitalists. We would like to expand the magazine to about 64 pages without too many advertisments and spend about $60,000 for additional promotion, but we cannot afford to do that right now and we'll just have to grow as fast as we can by bootstrapping ourselves up from where we are. We intend to provide the information that the large circulation cannot afford to cover, and we really do pay attention to what our readers have to say. Take the time and effort to drop us a line with your thoughts. ■

Nibble Magazine, 45 Winthrop Street, Concord, MA 01742
Micro Cornucopia, PO Box 223, Bend, OR 97709
Infoworld, 1060 Marsh road, Suite C-200, Menlo Park, CA 94025
Apple Assembly Line, PO Box 5537, Richardson, TX 75080.
Apple is a registered trademark of Apple Computer, Inc.
Kapro is a registered trademark of the Kaypro Corporation.
Xerox 820 is a registered trademark of the Xerox Corporation.
Big Board is a registered trademark of Digital Research, Inc.

"Gone Fishing!"— That's What Our Authors Said

This issue of TCJ is later than usual because we did not anticipate the delay in receiving manuscripts during the summer vacation period. We have a number of very interesting articles in progress, including some from new authors, but they are not quite ready for this issue.

We are looking forward to expansion and growth during our second year of publication and are in the process of relocating and revising our printing department in order to reduce the lead time and improve the quality.

We appreciate your letters and suggestions. Take the time to let us know what you want and tell us about people you feel are qualified authors. ■

Books of Interest

Microprocessors for Measurement and Control

by David M. Auslander and Paul Sagues
Published by Osborne/McGraw-Hill
630 Bancroft Way
Berkeley, CA 94710
310 pages, 7" x 9", softcover, $16.95

The design of mechanical and process equipment using microprocessors requires skills which are not treated in depth by most computer books. **Microprocessors for Measurement and Control**, which includes actual case studies, explores problems of increasing complexity and presents the complete plans and specifications for prototype systems. The authors take you step by step through the design examples. This is very helpful, as too many other books treat the subject on a theoretical level without concrete examples of real world applications.

The case studies are not tied to any particular language or computer, but stress the importance of developing machine independent problem solutions. The solutions to the examples are provided in either assembly language, FORTRAN, BASIC, Pascal, or C, and cover the 8080/8085/Z-80 series of 8 bit processors and the PDP-11/LSI-11 series of 16 bit processors.

•**Chapter 1 "Microprocessors as Components in Engineering Systems"** introduces the use of microprocessors in measurement and control and previews the case studies. This chapter includes information on speed/duty cycle characteristics of DC motors, pulse width modulation, A/D and D/A conversion, and response time.

•**Chapter 2 "Information and Power"** introduces the concepts of analog and digital signals and includes information on analog signals, coding of analog information, digital signals, timing of digital signals, typical output thresholds, synchronization using a clock, digital communications protocol, tristate logic, and noise and noise sources.

•**Chapter 3 "DC Motor Control and Testing"** covers the speed control of DC motors, and includes information on modulated digital signals, pulse width modulation, pulse frequency modulation, problems of pulse detection, timing programs, velocity determination program, and the motor controller.

•**Chapter 4 "Position Control with a Stepping Motor"** discusses the speed-power-cost trade offs in sensing and controlling the position of an object. The application of a stepper motor is illustrated with information on feedback versus no-feedback, dedicated microprocessor controllers, storing the program, the parallel interface, CPU assembly language instructions, accessing the control port, binary instruction codes for output port test, loading and running the program with a monitor, pulse train generation, timed waits, running the stepping motor, direction control, pulse timing using an external clock, and the stepper motor controller circuit.

•**Chapter 5 "Temperature Control"** continues the exploration of programmed I/O and discusses the use of a terminal and programming techniques to permit the operator to interact with the control program without interfering with the control process itself. Sections include interrupts, clock interrupt program, terminal interrupt program using indirect addressing, terminal output interrupt test program, the control algorithm, the blending control program, terminal input interrupt program, terminal output interrupt program, and the clock interrupt program.

•**Chapter 7 "Automatic Weighing"** covers the design of a scale using strain gauges in a DC bridge with an amplifier, and includes information on weighing algorithms, filtered output, weighing program logic, graphic display of weight distribution, communicating with a supervisory computer, program structure, ring buffers, character handling, serial line interrupt routines, message processing, modified clock interrupt routine, and strain gauge bridge and amplifier circuit.

•**Chapter 8 "A Polar Plotter"** emphasizes the hardware and system design aspects of microprocessor systems. Topics include the design trade offs associated with the drawing speed, resolution and line quality of a radial arm digital plotter, software control of the stepper motor, timing and ramping, micro stepping, polar plotting algorithm, real time software time constraints, floating point arithmetic operations, scaling, high level language considerations, and construction of the final prototype.

•**Chapter 9 "An Automated Cutting Machine"** covers the complete design cycle for a stand-alone, single-board system for use in production machinery. Covered in this design of a computer controlled cut-off machine for producing discrete lengths of a product from a continuously produced material are sequential logic, relays, programmed logic, the design approach, hardware selection, switch debouncing, operation simulation, a single board computer, and microprocessor development systems.

•**Appendix A** contains DC motor control programs in BASIC, Pascal, and C, plus 8080 assembly language listings for stepping motor, temperature control, and blending control programs.

•**Appendix B** contains DC motor control, automated weighing, and polar plotter programs in FORTRAN, automated weighing programs in C, plus PDP-11 listings for stepping motor, temperature control, and blending control programs. ■

RPM vs ZCPR:
A Comparison of Two CP/M Enhancements
by Bill Kibler

In the series of articles on System Integration (*The Computer Journal*, Vol. II, Nos. 2-4) I expressed the point of view that a user might be better off buying an enhanced disk operating system than getting the newer and more complex CP/M 3.0. Well, that line bothered me for some time and I finally decided to validate my position with an article that covered two ways to go. The first program is a CP/M replacement called RP/M. The other choice is the public domain program ZCPR. I will try and cover the good and bad points of these two programs so that you can decide for yourself if they will give you the desired enhancements.

Introduction

CP/M is a disk operating system composed of three parts. The CCP and BDOS are supplied by Digital Research, and the BIOS is written either by yourself or by the maker of your system. The CCP is the command processor and takes your request for action and either runs the requested program or provides one of the internal functions such as listing the directory. Most of the requested commands are turned into multiple calls to the BDOS (Basic Disk Operating System) to produce the necessary results. Some of those results will require many hardware inputs and outputs, which are done through the BIOS (Basic Input and Output System). These basic operations are what gives CP/M its standardized format, allowing many programs to work on many systems.

A more precise understanding of the interactions is needed before you can see why some changes might be wanted. First, the entire program is written in 8080 assembly language. For systems running on Z80 CPUs, this is not the most effective use of the hardware. The Z80 has many more registers and some special functions that can move large amounts of data much faster than 8080 code. Besides speed improvements, CP/M has several little quirks that can be most annoying. My favorite complaint is the TYPE command of the CCP. This function will start scrolling off text at a rather rapid speed without any way of going back a page or controlling just which page you are looking at. I suppose many people dislike the PIP command destination = source file description, and would like to change that too.

What Does What

What is needed now is an understanding of exactly what does what. The CCP has six built in commands which are TYPE, DIR, ERA, SAV, REN, and USR. The CCP takes your command line of data (again in the CCP) and converts it to the number and type of BDOS function calls to achieve the desired results. If your request is not one of the six resident commands then the program is loaded and your requested extra commands are passed along to the new program. Some error handling is done in the CCP, but most of it relates to the internal commands.

The BDOS has about 40 function calls that represent all the possible ways information is passed within the system. These function calls cover everything from a request for warm boot to writing data to a random file. The most commonly used one is for reading and writing to the console device (console in and out). The BDOS entry point is called with the C register loaded with the appropriate number for the function requested. The BDOS then reads the C register and jumps to the corresponding routine. To keep compatibility, these routines should remain the same—the use of non-standard function calls are to be avoided at all cost.

Some systems, especially those enhanced CP/Ms like Heathkits, have added their extras on the BIOS. These extras can be done by trapping commands in either console routines or disk routines. A common example of an enhancement is a RAM-DRIVE routine. The one I use loads above the BIOS and replaces the disk call entry points at the front of the BIOS with the new entries of the RAM-DRIVE. The new entry points will check for reference to the new drive and jump into the proper place. For non RAM-DRIVE calls it jumps to the original routine's address that was saved before the BIOS entry points were changed. This type of enhancement is easy to install and modify. An extra feature of such programs is the ability to use several versions, each with a feature turned on or off. For recovering data in RAM-DRIVE after a master reset, I run a version that does not clear the directory area.

ZCPR and RP/M

Versions one and two of the public domain program ZCPR are Z80 versions of the CCP. This program is a replacement for the normal CCP and also uses a handful of extra programs to obtain the enhancements. Besides speed (from the Z80), many of the all-time annoyances have been changed. The program and its files are spread over many 8" disks. If you buy these disks from your club or user group, it will cost you about $50 to $70. Although the cost is a factor, it is nothing compared to the thickness of the documentation. One point I didn't make in my article about documentation is "don't overdo it!" I feel that the authors of ZCPR have written *so* much about the program that the documentation becomes both confusing and redundant. This also applies to the programs supplied with ZCPR. There are

many accessory programs for use with ZCPR, each with its own documentation. Considerable disk space will be needed for the accessories but the Z80 version of ZCPR is the same size as the CCP.

For those looking for limited enhancements without major operation changes, a complete CP/M replacement is RP/M. Several minor but desirable changes to the RCP (the CCP part) give you paging in the type command. The biggest advantage of the program is the complete listing. Because of the listing, it is possible to change minor features by using DDT. I didn't like the fact that this program was exactly like CP/M in every way, including the > symbol. I decided I needed a way to tell the difference, so I got out the book, found where the routine outputs the >, and presto, now RP/M uses a } symbol instead. This was helpful later when I found that DSKFIX (a disk fixer utility) wouldn't work, as I immediately knew to load my CP/M version and it worked.

Installation

Of the two programs, RP/M is by far the easiest to install. The manual's first page provides all the needed instruction for a normal installation. I asked several friends to look at the book and try the installation, and all had little trouble. Of the several systems I tried installing it on, only the SDSystems caused any problem. If you have been following my articles over the last year, you will know something about this system. The SDSystem has a ROM at F000h and a rather short BIOS, as all calls are into the PROM. RP/M has an install program that finds the BDOS scratch table and subtracts 200h for the beginning of BDOS. Now this method is supposed to be the safest way to get the address of your CP/M (page 0 data can be messed up by DDT type programs).

My 62K CP/M gives an erroneous system value and yet works properly when moved to 64K. I have as yet to find out why this is, but the author of RP/M has provided an alternate method of loading a system. This alternate method is both for loading a system at unusual or hard to find addresses, and for generating oddly placed systems. My only complaint here is that it took me a considerable amount of time to find the instructions for locating the system at an unusual address. The instruction is on the disk under "MEMSIZ.DOC," not "RP/MGEN.COM" as the book states. This problem was quite hard to identify at first, as the system was loaded in the right place, but nothing functioned. After considerable experimenting and some help from the author which sent me to all the wrong places, I discovered that it was just assembling the program 200h low. So don't forget to check where it is relocating itself to before looking for more difficult problems. This involves calculating new addresses for the first two jumps in the RCP from the listing in the manual.

To install the ZCPR program will simply require doing a system modification under DDT and reinstalling it on the boot tracks of the disk. For full use of all accessory operations however, changes in the BIOS will also be needed (total changes may reduce TPA by 1K). If you buy the ZCPR from one of the systems houses, or from a magazine like **Microcornucopia**† (a magazine for single board systems like KAYPROs) all the assemblies will be done. If this is your first attempt at system modification, the programs will be a bit of a challenge. The manual even states this up front, which helps account for the extra amount of documentation. Some tutorial information is given for those with little experience in doing system programming. Many of the accessory programs will also need assembling before use. This program is in Z80 code, MACROs, and many different forms (an 8080 version is available too) so you may face several other problems in getting the thing together (you will need DRI's MAC assembler or Microsoft's M80/L80). In reality, it goes together fairly easily and can be a good practice or learning session on using all the programs needed.

Support

For support, RP/M is still the best, as you can call them up and get help. ZCPR is a public domain program and as such, lots of people have worked on it—it can be pretty hard getting help from them. Most computer clubs, however, will have several people running ZCPR (few will know about RP/M) and they will provide the best source for help. RP/M, on the other hand, was intended for OEMs as an alternative to CP/M. This commercial slant to RP/M gives it some other benefits, mainly a professionally printed manual. RP/M's manual is good not only in content but also in the way it explains some of the information. The manual does not cover everything, but gives enough information for you to see how it is all done. The listing for assembly language programmers provides a wealth of information about how things are done. If you really get creative, you can disassemble RP/M and have your own versions. It took me about three days to disassemble it and find my errors (use DISASM.BAS on SIG/M 23). Now, I would not recommend this for just anyone, but I had thought that I was getting the ASM file on disk, since the advertisement says, "printed listing provided." Should you really want to change things around you can get the full assembly of RP/M, which is not possible with CP/M. You could also disassemble CP/M and with the help of RP/M, try and figure out what does what. I felt it was just easier to buy RP/M.

ZCPR is provided in assembly on the disk and will allow you to change it as much as you want. The difference is the BDOS. ZCPR is just a CCP enhancement and is not going to effect the BDOS calls at all. So it comes down to whether you want changes in the BDOS or the CCP. This brings us to what your real goals are, and what shortcomings you wish to correct.

Problem Solving

What both of these programs demand is that you understand your needs and your system. I prefer RP/M, as it fits exactly within the same space as CP/M. You may want ZCPR for all the extra system enhancements. Some may find both programs a waste of time and money. I personally feel that your level of programming experience will determine your needs. Systems programmers spend several hours each day working within the system. This may include

†*Microcornucopia*, PO Box 223, Bend, OR 97709.

checking the directory, using threaded directories (like in ZCPR), assembling and reassembling (using fancy SUBMIT programs), and possibly quick loading systems. All of these system operations are for special needs. Now, you might be doing lots of data handling and need the threaded file directories to speed your operation, but for a lot of people, plain old CP/M will do. I guess my big complaint would be the need for more disk space for ZCPR's extra programs. If I had a hard disk, I think ZCPR would be an absolute necessity, as keeping track of the number of programs on a disk could drive one crazy. Both programs have the USER 0 common to all areas, with ZCPR able to set the path for searching the drives and user areas for a program.

Conclusion

You might still have a lot of questions about these programs, but the fact is, there just isn't that much good or bad to say about them. RP/M worked as advertised, and the two problems are minor. The failure of the auto install programs was handled by the ability to assign it an address to load at. One problem that came up was the non-operation of DSKFIX and lack of an ASM file on the disk. I consider the ASM file to be the main reason for buying the program, and without it I really question its usefulness. If paging in TYPE and USER 0 function are your only reason for getting the program, don't—there are patches to CP/M available to change that. There are also several public domain programs that can give you some of those enhancements without changing anything in your system.

In ZCPR I have trouble seeing enough advantages over the utilities I already have on disk. As I just said, making the patches to CP/M may be more than enough for most users. I generally do most of my work on single sided single density 8' disks (I do lots of work over several systems) and find that disk space is a problem—I would have no use for a threaded directory listing. I guess what it comes down to again is knowing what you need and seeing if these programs meet that need. These are not the only enhancement programs available, so check the advertisements closely. I still feel that you will be better off checking with your local computer club and finding someone using what you think you might want, and trying it out first. Nothing replaces experience with these systems, or with any system for that matter.

Soapboxing

In concluding this talk on enhancements, I feel some comments on my philosophy of computing are needed. I am a stickler for documentation, not only for the needed information, but for information you may not *know* you need. This point is also important when buying software, because your needs will change as your skills in using it change. These two programs are good examples of changes, which, once they are installed, may breed more reasons for their use. I personally prefer a system in which I can have full control over how it does what. The idea that someone else has decided how I will do my computing defeats the idea of microcomputing. Personal computing doesn't only mean that it fits on your desk, but that it meets your personal style of doing business, keeping data files, and in my case, writing articles for publication. The more I write and use my Z100, the more I like it. At the same time, however, I keep finding how little I use a lot of what I paid for. All these extras are worthless if they are never used, and I am afraid that buying computers is the newest form of status.

It seems that speed trials and fancy packaging are more important now than good design. Although this may be a fad that will pass once the majority of users become educated, I have my doubts. The craze over the IBM PC, a rather poorly designed product, shows how little the knowledgeable people can affect a product's sales appeal. Without continuing my soapbox stand, let me conclude by saying how important it is, and will continue to be, not to be swayed by advertising. Start by developing real shopping habits that include finding and cultivating friendships where knowledge becomes a part of your activity goals. Not everybody can afford to buy every piece of software or book written on the subject, but through eyeball to eyeball contact we can spread our own experiences around and save considerably on the frustration involved. *The Computer Journal* is also a medium for expressing those newly learned answers to problems. We know people are reading these articles and have different viewpoints on the topics covered. I have received a few comments on my articles, but what I would rather see is articles that grow on what I have started. If you have installed ZCPR2 and found it to be just what you wanted, let us know WHY, HOW, and WHAT happened. Only through this exchange can we improve both *The Computer Journal* and your computing abilities.

DISASM.BAS

I know several people will want the information on using the disassembler (especially since I enhanced it), so I have supplied the changes to it. These changes give you control over the size of the disassembly, as the program is rather slow and creates an extra large file. The normal program turned the 6K RP/M into a 98K text file with a line of text produced for each byte. After shortening it (address, hex value, mnemonics per line), it will do a 38K text file. Several versions are possible, and with the program being in BASIC it will be possible for just about everybody to change and run it. I tried other disassemblers on SIG/M disks and found this to be the best. My next project with this program is to give it a set of 8088/86 files and try using it under MSDOS. After a disassembly, I used Wordstar's subsitution program to change addresses to names, but watch out—sometimes the equates use the same address for two different functions.

```
45 PRINT "This program modified for shortened output sequence
7/84 "
545 FX=0:'SHORTER FLAG
735 PRINT " S - Eliminate long listing ";:IF FQ=1 THEN PRINT
TAB(35);" Shortened" ELSE PRINT
845 IF S="s" OR S="S" THEN IF FQ=0 THEN FQ=1 ELSE FQ=0
2950 IF AI>0  THEN D=20 ELSE D=0
2960 IF FC=1 AND ( FQ=0 OR AI=0) THEN PRINT N;:IF AI>0 THEN PRINT
TAB(10);ZCOM;
2970 IF FP=1 AND(FQ=0 OR AI=0) THEN LPRINT N;:IF AI>0 THEN LPRINT
TAB(10);ZCOM;
```

continued

```
2980 IF FW=1 AND (FQ=0 OR AI=0)THEN PRINT #2,N;ZT;:IF AI>0 THEN
PRINT#2,ZCOM;ZT;ZT;ZT;
2990 IF C(1)>0 AND C(2)>0 THEN Y0=Y(1)+","+Y(2) ELSE Y0=Y(1)
3000 IF FC=1 AND (FQ=0 OR  AI=0) THEN PRINT TAB(10+D);Y(0);:IF
C(1)>0 THEN PRINT TAB(16+D);Y0;
3010 IF FP=1 AND (FQ=0 OR AI=0) THEN LPRINT TAB(10+D);Y(0);:IF
C(1)>0 THEN LPRINT TAB(16+D);Y0;
3020 IF FW=1 AND (FQ=0 OR AI=0) THEN PRINT #2,Y(0);ZT;:IF C(1)=0
THEN PRINT #2,ZT;ZT; ELSE PRINT #2,Y0;ZT;:IF LEN(Y0) 8
THEN PRINT #2,ZT;
3030 IF FW=1 AND AI=0 THEN PRINT #2,ZT;ZT;ZT;
3040 IF FC=1 AND FQ=0  THEN PRINT TAB(50);ZCOM;ZBYTE;"    ";O;"
";P
3041 IF FC=1 AND FQ=1 AND AI=0  THEN PRINT
3050 IF FC=1 AND FQ=0 THEN LPRINT TAB(50);ZCOM;ZBYTE;"    ";O;"
"P
3051 .F FP=1 AND FQ=1 AND AI=0  THEN LPRINT
3060 IF FW=1 AND FQ=0  THEN PRINT #2,ZCOM;ZBYTE;ZT;O;ZT;P
3061 IF FW=1 AND FQ=1 AND AI=0 THEN PRINT #2,ZT
3070 REM IF FC=1 AND MID$(N,5,1)="F" THEN PRINT
3080 IF FP=1 AND MID$(N,5,1)="F" THEN LPRINT
```

Sources:

RP/M from MICROMETHODS, PO BOX G, 118 S.W. 1st, Warrenton, Oregon 97146; Cost $75 plus $5 mailing. By Jack D. Dennon

ZCPR SIG Disks:

#66 FIRST CUSTOMIZED CCP REPLACEMENT ZCPR-14.ASM ON DISK
#77 ZCPR-16 FOR ZCPR 1.0 AND 1.6
#86 ZCPR-10 UPDATE ON THIS DISK WITH OTHER CP/M UTILITIES
#88 SYSLIB DOCUMENTATION, LIBRARY UTILITIES USED IN ZCPR
#89 SYSLIB.HLP,MAC FILES AND DOCUMENTS
#90 MORE SYSLIB MAC FILES
#98 1 OF 10 ZCPR2 DISKS, VERSION 2 OF SYSLIB 2.3 LIBRARY
#99 ASM FILES OF 5 SUBPROGRAMS OF ZCPR2 ie:DU2
#100 14 MAC FILES FOR UTILITIES TO GO WITH ZCPR2
#101 13 MORE MAC UTILITIES
#102 31 UTILITY COM FILES AND 4 HELP FILES
#103 11 HELP FILES
#104 ZCPR2 CONCEPTS AND INSTALLATION MANUALS
#105 ZCPR2 USERS GUIDE
#106 ZCPR2 SYSLIB DOCUMENTATION
#107 MORE ZCPR2 SYSLIB DOCUMENTATION
#108 ZCPR2 MODS AND UPGRADES
#116 SOME UNOFFICIAL ZCPR2 UPGRADES
#122 ZCPR2 FOR 8080 MACHINES, NEEDS VOLUMES 98-108 FOR SUPPORT
#124 VOLUME ONE OF UPGRADES TO ZCPR2
#125 REST OF UPGRADE WHICH INCLUDES KAYPRO, MORROW, OSBORNES ■

AC CIRCUIT ANALYSIS ON A MICRO

Applications of BV Engineering's "ACNAP"

by Art Carlson

Analysing bandpass filter designs has always been something I avoided if at all possible, so when I became interested in active filter design I looked for a way to turn the drudge work over to the micro.

The requirements of the program I wanted were reasonable cost, ease of use, flexible (with easy to change parameters for rapid design changes), and the power to handle many types of circuits. I really didn't expect to find a program which would fill all the requirements, especially at a reasonable cost, but I saw BV Engineering's ad for their ACNAP (Electronic Circuit Analysis Program) which sounded interesting and was priced at only $69.95. We requested a copy for evaluation on our CP/M-80 system, and received the disk plus a 27 page manual with instructions on using the program and a step-by-step example.

ACNAP is written in FORTRAN, compiled into machine code for the system on which it is to be used, and is menu driven. The initial screen is shown in Figure 1, and the first step is to choose whether you want to enter a new circuit or read an existing circuit. If you choose to read an existing

```
****************************************
* AC NETWORK ANALYSIS PROGRAM (ACNAP)  *
* ------------------------------------ *
* CP/M-80   v1.4      Serial # 21807   *
*       Written by G.P. Boucher        *
*  Copyright 1983     BV ENGINEERING   *
****************************************

            MAIN MENU:
            ----------
   1) Enter new circuit manually
   2) Read old circuit file

         Enter choice ?
```

Figure 1

circuit, you are asked for the drive number and the filename, and then presented with the command menu shown in Figure 2. If you choose to enter a new circuit, you are asked for the number of nodes, and then the component values and tolerances. After entering the data, the circuit can be analyzed with the output to either the screen or the printer, or the circuit design can be saved to the disk. In addition to listing the magnitude and phase of the output, you can also run a Monte-Carlo analysis to determine the effect of random component variations, compute component sensitivities, or save special disk files for plotting the results or for spectral data analysis. The best way to understand a program is to use it, and the following example will demonstrate using ACNAP on a multiple feedback bandpass filter.

```
MENU:    1) Enter new circuit manually
         2) Read old circuit file
         3) Save circuit in file
         4) List circuit values
         5) Change component value(s)
         6) Run circuit (or PLOTPRO dump)
         7) Run Monte-Carlo on circuit
         8) Compute component sensitivities
         9) Save spectral data in file
        10) Toggle printer on
        11) Compute noise equivalent bandwidth
        12) Quit
```

Figure 2

Bandpass Filter Analysis

This program analyses an existing design, but does not determine the component values for the initial design, so we will analyze the circuit on page 151 of *Design of Op-Amp Circuits* by Berlin (from the Blacksburg series published by Howard W. Sams) in order to demonstrate the use of ACNAP. This is a multiple feedback bandpass filter as shown in Figure 3.

On the first menu we will choose the new circuit option, enter the number of nodes (4), and then enter the component values as requested. Note that all frequencies, component

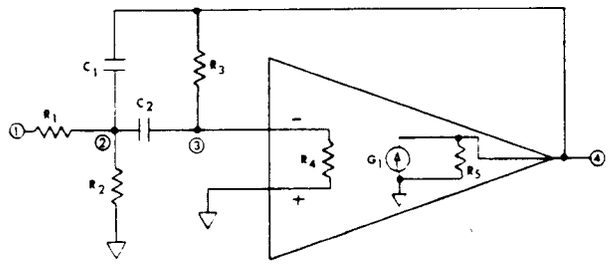


Figure 3: Operational amplifier such as the 741.

values, and component tolerances must be entered as floating point numbers. A floating point number is a number with a decimal point in it or one described in scientific notation such as 1000.0 (the decimal point and trailing zero are necessary) or 1E + 3. Node 0 is always the ground node, and node 1 is always the input node. The output may be any node and is requested by ACNAP during the analysis. The

allowable range is 2 to 20 nodes not including the ground node 0, and a maximum of 60 components with not more than 10 of these being controlled voltage sources.

After entering the number of nodes, ACNAP asks for the capacitor value, tolerance, and node connection points starting with C1. ACNAP will ask for the values of C2, C3, etc. until a capacitor value of zero has been entered. Next it will ask for the values of the inductors, and since we do not have any inductors in this circuit we will enter zero for the first inductor value. Then we enter the values for the resistors. The "+Node" and the "−Node" are interchangeable for inductors, resistors, and capacitors. No polarity is assigned for these components and the nodes only refer to each end of the component.

The last component type to be entered is the voltage controlled current source which must be entered a little differently because ACNAP needs to know the nodes between which the current source is connected and also the nodes which control the current source. The polarity is critical for the current source, as positive current is delivered to the "+" node if the "+" control node is more positive than the "−" control node. After entering all the values, ACNAP will return to the main command menu as shown in Figure 2.

Now ACNAP is ready to analyze the circuit, but first we will choose item 3 to save the circuit values to a disk file which we will name ACTFIL. The file naming procedure varies with the different operating systems, but for the CP/M system which we are using we are allowed eleven characters which must be all upper case and we are not allowed to use a period for file extension. After saving the file, we will choose item 4 to print out a list of the component values so that we can check for errors (see Figure 4). Next, we select item 6 to run the circuit with a minimum frequency of 700.0 to 800.0 (remember the decimal point and the trailing zero) with 25 intervals using a linear sweep and node 4 as the output with the data going to the printer. ACNAP asks for the required parameters and then runs the circuit as shown in Figure 5.

Examination of the data shows that the frequency of the filter using the nominal component values is between 736 and 740 Hz. Another run could be performed between these limits to find the exact design frequency, but in real life the frequency depends on the actual component values and not their nominal value. A more useful step at this time would be to choose option 7 from the menu to run a Monte-Carlo analysis on the circuit during which ACNAP builds a

Component	Value	Tolerance	+N	−N	+V	−V
C 1	.1000E-07	.200	2	4		
C 2	.1000E-07	.200	2	3		
R 1	.6800E+05	.100	1	2		
R 2	.2700E+04	.100	2	0		
R 3	.1800E+06	.100	3	4		
R 4	.2000E+07	.200	3	0		
R 5	.7500E+02	.300	4	0		
G 1	.2500E+05	.300	0	4	0	3

<Enter 0 for main menu> 0

Figure 4

Frequency	Magnitude(dB)	Phase(Deg)
700.00	1.7314	-157.26
710.00	2.0578	-163.26
720.00	2.2891	-169.55
730.00	2.4135	-176.02
740.00	2.4263	177.51
750.00	2.3309	171.18
760.00	2.1383	165.13
770.00	1.8643	159.47
780.00	1.5268	154.27
790.00	1.1433	149.53
800.00	.72929	145.27

Figure 5

number of circuits using random component values within the tolerance ranges of the components. We specify the frequency range, number of intervals, type of sweep, node number to be analyzed, and the number of circuits to be built. We can run the test at a single frequency by entering the same frequency for both the minimum and maximum frequencies. It may take quite a while to run the test if we specify a large number of circuits and many frequency intervals, so ACNAP displays the number of circuits analyzed on the terminal so that we know how the program is progressing. It took about four minutes to analyse 20 circuits for the active filter over 25 frequency intervals during our test. ACNAP displays the minimum, mean, maximum, and 3 sigma limits of both the magnitude and phase for each frequency as shown in Figure 6 (we only show data for three of the frequencies). This report will indicate the variations in the performance of the filter due to the normal component value variations within their tolerance ranges.

You should be aware that manufacturers often screen

```
**************************************************

Frequency=   720.00

Magnitude (dB)...Min=   -4.4481          Max=    3.3610
                 Mean=    .38962     3 Sigma=    5.9607

Phase (Deg.)     Min=   -166.21          Max=    175.67
                 Mean=  -38.318     3-Sigma=    432.91



**************************************************

Frequency=   740.00

Magnitude (dB)...Min=   -3.6033          Max=    3.9109
                 Mean=    .61960     3 Sigma=    5.9055

Phase (Deg.)     Min=   -178.31          Max=    163.04
                 Mean=  -47.621     3-Sigma=    431.80



**************************************************

Frequency=   760.00

Magnitude (dB)...Min=   -3.7462          Max=    4.0809
                 Mean=    .65767     3 Sigma=    6.2780

Phase (Deg.)     Min=   -176.68          Max=    178.99
                 Mean=  -2.9222     3-Sigma=    461.45



**************************************************
```

Figure 6

components and remove the parts which are close to the nominal value to be sold as precision parts, and that when you order parts with a large tolerance the values of the parts you receive may not follow the normal distribution curve because the values near the center of the curve have been removed from the population. If you decide that the data from the Monte-Carlo analysis indicates that you need to reduce the performance variations because of component tolerances, you should select item 8 from the menu to compute the component sensitivities in order to determine which components have the greatest effect on the parameter you want to improve. Like the Monte-Carlo analysis, the sensitivities analysis may be run over a range of frequencies. The analysis tells you how many decibels the output magnitude changes, or how many degrees the output phase changes for a 1% increase in the value of that component. The sensitivity analysis for the active filter is shown in Figure 7, and provides the information needed to

Component sensitivities at 720.0 Hz.

Component	Mag. Sensitivity(dB)	Phase sensitivity(Deg.)
C 1	.14066E-01	-2.3116
C 2	.10050	-2.3116
R 1	-.83939E-01	-.89203E-01
R 2	.56278E-01	-2.2609
R 3	.14294	-2.2714
R 4	0.0000	0.0000
R 5	.28610E-05	0.0000
G 1	-.14305E-05	0.0000

Component sensitivities at 740.0 Hz.

Component	Mag. Sensitivity(dB)	Phase sensitivity(Deg.)
C 1	-.66358E-01	-2.3632
C 2	.20074E-01	-2.3632
R 1	-.87030E-01	-.89523E-01
R 2	-.21592E-01	-2.2502
R 3	.63072E-01	-2.3874
R 4	-.59605E-05	0.0000
R 5	-.16689E-05	0.0000
G 1	0.0000	0.0000

Component sensitivities at 760.0 Hz.

Component	Mag. Sensitivity(dB)	Phase sensitivity(Deg.)
C 1	-.13856	-2.1943
C 2	-.52135E-01	-2.1943
R 1	-.89728E-01	-.81558E-01
R 2	-.87846E-01	-2.0347
R 3	-.12563E-01	-2.2747
R 4	0.0000	0.0000
R 5	-.14305E-05	0.0000
G 1	-.14305E-05	0.0000

Figure 7

determine which component tolerances can most economically be tightened to reduce the variation in performance. By selecting the appropriate items from the menu, we can change the tolerance (or value) of selected components, save the revised circuit under another name, and rerun the circuit in order to evaluate the effects of the changes.

ACNAP can also compute the circuit's noise equivalent bandwidth, and save two special types of files for use with other programs from BV Engineering. When running the circuit from option 6 on the menu, ACNAP asks if you want to save the data to disk for use by PLOTPRO (BVE's scientific graph printing program). If you answer yes to the question "Write data to diskfile for PLOTPRO (Y/N)?," ACNAP will ask for a file name of one to eight characters with no extension allowed, and will write two files to the disk with extensions automatically appended by ACNAP. One file will contain the magnitude data in dB with the extension .MAG for the CP/M system, the other file will contain the phase data in degrees with the extension .PHS for CP/M systems. The PLOTPRO program will not be reviewed at this time, but we have included a high resolution graph of this circuit printed on an Epson MX-80 printer (Figure 8). PLOTPRO can print low resolution graphs with standard 80 column printers or high resolution graphs with printers capable of printing 132 columns in compressed mode. I find it is much easier to review the results of circuit changes using graphs instead of tables of raw data.

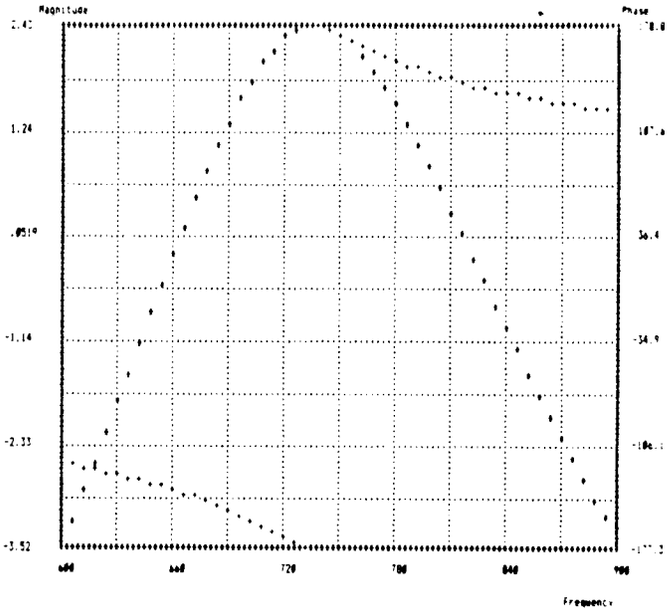


Figure 8

Option 9 on the menu saves the output response to a range of frequencies in a file which consists of 255 pairs of complex numbers which define the response of that circuit to sinusoidal stimulus of 255 harmonics of the basic fundamental frequency. This file can be used by SPP (BVE's Signal Processing Program) to compute the Time-Domain response of the circuit to any input wave form that can be defined as a function of time, or as a series of 512 data points. The result of SPP is the time domain output that the input wave form would be transformed to by the circuit, and is useful for transient analysis or analysis of the response of linear networks to non-linear stimulus.

ACNAP performed well, the menu driven commands made it easy to use, and the documentation was well written. There are only three minor improvements which I would like to see. The first is really the result of my poor memory—I'd like to be able to see the disk directory without exiting to the system, because I forget which file

continued on page 22

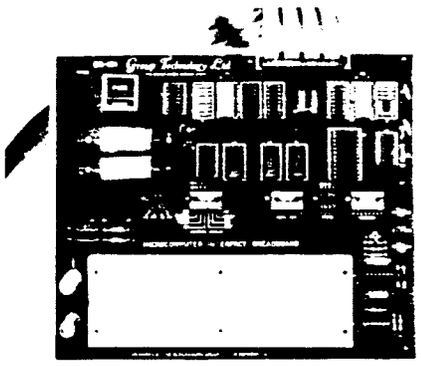
LEARN MICROCOMPUTER INTERFACING
VISUALIZE SCIENCE PRINCIPLES

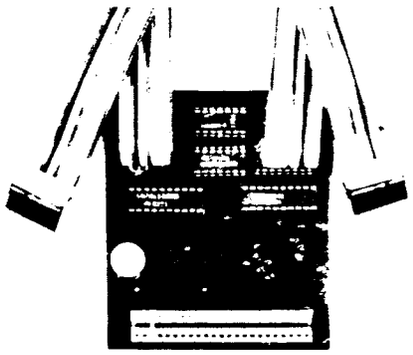
Using GROUP TECHNOLOGY BREADBOARDS with your
APPLE® ...COMMODORE 64® ...TRS-80® ...TIMEX-SINCLAIR® ...VIC-20®

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





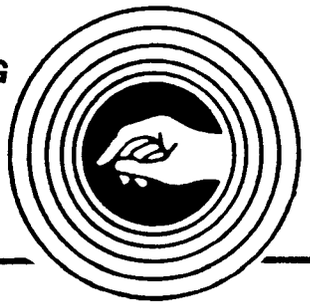
The INNOVATOR® BG-Boards designed by the producers of the highly acclaimed Blacksburg Series of books have gained wide acceptance for teaching microcomputer interfacing as well as for industrial and personal applications. Detailed, step-by-step instructions guide the user from the construction of device 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

BASE

A Series on How To Design and Write Your Own Database

By E.G. Brooner

Part One: Introduction

We readers of *The Computer Journal* like to know how our microcomputers work and how to get the most out of them without buying everything the market is trying to push on us. It is hoped that this series will contribute to this effort, and that readers will contribute their comments and questions about this subject. We are especially interested in your experiences with any special application of data bases that you have encountered.

The terms "database" and "data base" have been much abused and mean many things to many people. In the most fundamental sense a database is *any significant collection of information.* It need have nothing at all to do with computers, but the kind you and I are interested in usually has that connotation. We get closer to a meaningful definition if we add that there is *some precise and particular way, or ways, to access the information.* A comparison could be made to the difference between a library and a warehouse full of books. The warehouse is merely storage, while the library is organized in specific ways. Perhaps the ultimate refinement of a library, or a database, is the provision for extracting different kinds of reports, for different purposes, from the same overall collection of data.

Information utilities, such as "The Source" and "Compuserve," are examples of fairly elaborate databases; they are characterized by vast quantities of information and some useful but fairly primitive ways of accessing it.

A telephone directory is a simple analogy of a database. If the names were not in alphabetical order it would be a virtually useless collection of names, addresses and numbers. As the directory exists, you have to know the name to find the number and the address. This makes it vastly more useful. The yellow pages add another dimension—you can find businesses by category. Now, wouldn't it be nice if you could also find the name or address if you knew the phone number? Or list all of the merchants of a certain category in some particular part of town? A good database could add all of these features and more.

Believe it or not, there are database programs capable of helping a doctor diagnose and treat disease. Software of this sophistication is more properly known as an "expert system" but is basically just a very elaborate database. There is a world of difference between traditional, large system databases and those available for micros. The concept has only been applied to small computers in the past few years and some of the first packages were indeed crude.

Better packages, many with different features and capabilities, are becoming available. Some are more difficult to use than others, and the documentation for some of them is enough to discourage their practical use. In one sense any program that stores data might be considered a database even though it is designed for a single purpose. As we add flexibility, it more closely resembles a database. If we add too many features we find that a completely new "query language" is needed for its use, and we are once again writing programs from scratch to suit our immediate need. Definitions differ, and most of us will never see the inside of the more exotic systems. Very few of us need a database program that does everything, and if we had it, we probably couldn't run it on our micros.

Somewhere there is a balance. Your choice of a database depends a lot on what you expect to use it for. A program that prints mailing labels, for example, can dispense with many features, but it may have some capabilities that are not necessary for other applications. Knowing what you need and how it can be done can be a great help in getting just the right package for your application.

Although the concepts are often blended together, any database can be functionally described as consisting of four major parts.

1. The first task is to define the purpose and organization of the base; this determines everything that follows.
2. The second is the entry and updating of data.
3. The third is data retrieval, and
4. The fourth is the generation of useful reports. Report, as used here, is a very broad term. It could mean something as simple as a set of mailing labels or as complex as the material for a research project.

This series will concentrate on the sort of things that are practical for the readers of *The Computer Journal*, most of whom use relatively modest equipment and are primarily interested in how things work rather than "what you could do if you had a million dollars." It is not intended to be a tutorial on how to use DBII or any other commercial package, although we might reference such things from time to time. In this column we hope to clarify the construction and use of database programs and perhaps comment on some of the commercial versions. The main objective, though, is to present the fundamentals in enough detail so that you can write your own, or at least evaluate those that might be available to you.

In talking to the users of commercial database software for micros, it becomes apparent that no single package suits

everyone's needs. The complaints involve such things as record size, speed, and methods of retrieval, to name just a few. One of the best available packages for micros permits only one key field per record—this is a severe limitation. All of the available packages have some very good qualities but none of them, apparently, will satisfy all needs.

An element of primary importance is the manner in which data is stored and retrieved. This involves file types, file generation and access, sorting and searching, and many other features that are part of the programmer's tool-kit. The fundamentals are the same in any language but, needless to say, we have to pick one in order to provide examples and program listings. I have chosen C-BASIC for a number of reasons; among these are its overall sophistication and the fact that it can be compiled. It is similar enough to other common languages that anyone the least bit familiar with programming should be able to follow and apply what will be exposed.

The foregoing, of course, is simply an introduction to the column. In the next issue we plan to go into some detail in the matter of disk files and their manipulation. It should be evident that there is no practical application of database programming if you do not have at least one disk drive—the more the better, and the larger the better. A database does not become a really effective tool unless it can be applied to very large collections of data. The principles, though, can be learned with relatively little disk space. (The kinds of tape systems available to most hackers do not lend themselves well to the rapid, random, and frequent file access that is the database's major characteristic.)

If you do not use CP/M and C-Basic, or something very similar, you may have to do a bit of translating; the general principles, though, will be kept as simple and universal as possible. Although the program that will be developed and listed is quite long, it will be given in segments that can "run" independently of one another in a relatively small memory.

If you'd like to do a bit of advance studying, try to become familiar with your system's means of writing both sequential and random access files, limitations on record length and file length if any, and especially the means of addressing or accessing individual records within a given data file. Find out, too, if your system requires data to be written to the disk in any particular format, i.e: as text, as variables, or whatever. If you use some of the Microsoft BASICs you might also have to learn that rather cumbersome technique known as "fielding" string variables. If you are completely new to disk files you might also want to learn what limits there are, if any, on the number of files open at any one time and how disk files are created, named, opened and closed.

When you are that far along, we'll begin building files—in theory at first—and relating them to the database that we'll construct as the series progresses. As we go along, we'll try to construct a simple but useful data base program and suggest ways that additional features might be added to it. In the next issue we'll take up some of the necessary details of organizing and manipulating data files. ■

UNDERSTANDING SYSTEM DESIGN

by Bill Kibler

To begin working with computers one must first understand how they are built. This understanding must cover not only the individual components but also the whole system concept. The most common stumbling block for new users is understanding how all the pieces fit together. The interrelationship of components is not as large a problem as the word might indicate. The basic computer has not changed much over the years from the three basic parts: CPU, memory, and I/O.

CPU

The CPU, or Central Processing Unit, is the main part of any system. This unit is currently a single VLSI (Very Large Scale Integrated Circuit). The CPU which first achieved wide use at a reasonable price was the Intel 8080. Several other devices were made before the 8080 but their designs had many limiting features. The Motorola 6502, which is used in the Apple® , has very limiting design constraints but is still being used. Prior to the VLSI, CPUs with separate logic circuits were used; they are still used in some very large systems.

There are many books which cover the history and design of CPUs in greater detail than I will do in this article. What a system integrator is most concerned about is the support and availablity of the types of hardware he may encounter. The most common types of used systems will be 8080 or Z80, some 6502s, and in most new systems, the 8086/8088. Some 6809, 68000, Z8000, and 16000 types may be in the works for those people who deal with the leading edge of techknowledge. Whichever CPU is used, support will be the primary stumbling block. No matter how the system is designed, some form of assembly language programming will be necessary. This programming means that that you will need to understand the internal architeture of the device and its support logic.

Internally, all the CPUs are similar; they will have a number of registers for storing temporary data. These registers will most likely have other functions, such as indirect addressing. These registers are used for doing the work of the system, much as you might write temporary figures on a piece of paper when balancing your checkbook. All CPUs have ALUs, or Arithmetic Logic Units, which do the addition, subtraction, division, sometimes multiplication, and most logic evaluations (NORs, ORs, ANDs, test for zero, positive, negative). These ALU activities are sometimes limited only to the accumulator or "A" register and may be 8 or 16 bits wide. The 8080/Z80 has an 8 bit accumulator whereas the 8086/8088 has a 16 bit accumulator—otherwise the devices are quite similar. Another point of understanding is the way the devices move data to and from memory. The 8080, Z80, and 8088 work in 8 bit data structures, while the 8086 has a 16 bit wide data path. The 68000 series of devices come in several versions for different data paths, as do most of the newer CPUs.

For each device the internal and external methods of handling data must be understood to write assembly code. This information is usually obtained from the manufacturer's data sheets and programmer's guides. Trying to do assembly programming without these books is nearly impossible, and for critical timing they are absolute necessities. Not only will the books explain the code, but they will also tell how long it takes to do a given operation. When dealing with disk units, for example, it is necessary to know how long a certain type of data transfer will take to see if that function will actually work or not.

Memory

The CPU's ability to perform functions is made possible by memory. Memories are devices that will store data, much as post office boxes store letters. These devices consist of two general types; static and dynamic. Static devices maintain their memory without any assistance as long as the power is on. Dynamic devices require refreshing of the data, since they are capcitors and the charge disappears quickly. These devices are the most common forms of RAM (Random Access Memory). Another type is ROM (Read Only Memory) which is the device that stores programs permanently. Temporary permanent memory storage is done in EPROM (Erasable and Programmable Read Only Memory) or EEPROM (Electrically Erasable PROM).

All of these devices provide a place for programs and data to be stored and utilized by the CPU. Memory units are normally decribed as *bytes* or *words* of memory. Bytes are usually understood to be 8 bits wide, whereas words are the data size that the CPU works in (8, 16, 32, or larger). The CPU will read or write data into and out of memory as specified by the program. Some internal programs, such as resets, will automatically start reading data from a fixed location as they start looking for instructions. The memory devices and the CPU have no way of knowing whether the data is information or instruction. If instructed to go to the wrong place, a CPU may load raw data but treat it as instruction, locking up the system.

When dealing with system integration, the size and type of memory must be fully understood. Memory constraints will determine the speed at which the system operates, how large the programs can be, and what types of operating systems are possible. These building blocks, with the

addition of I/O, make an actual computer.

I/O

The I/O is the input and output system. It talks to the outside world, and allows the unit to perform meaningful work. The most common form of I/O is the serial interface. This operation takes the normal parallel data stream and converts it into a serial stream (single data bits following each other). This information is supplied in a standard format that allows computers from different manufacturers to talk to each other. There are many different types and styles of I/O, one for each use the computer can be put to. For example, a common system might have a serial terminal, a parallel printer, and an analog data source for collecting data.

This data is usually obtained through some VLSI device. These devices are, in fact, special dedicated CPUs that will perform a number of functions depending on the program that was stored in them after reset. A typical device is the Floppy Disk Controller or FDC. The CPU devices can be programmed to write data in certain formats, check the data string aginst a prerecorded value for lost or missing data bits (CRC), change the data from parallel to serial, move the heads, and perform other hardware functions such as turning the motor on or off. To do this, the programmer first sets up the device by loading the internal registers through software, and then makes sure the hardware handshaking is coordinated with the main CPU's operations. This handshaking can be critical, and some devices will not work if data is not removed from the device in a certain amount of time (critical timing loops).

Buses

Timing is also a critical consideration when dealing with bus type systems. One of the most common (and inexpensive) buses around is the S-100. This is the bus that the 8080 got started on by Altair and later IMSAI. Although the actual assignment of signals on the bus is not necessarily the best layout, it has now become a standard. This standard is called IEEE 696, and represents the group number assigned by the IEEE organization, the task force which put in cement all the needed specifications of the bus. Meeting the IEEE specs is now necessary for any bus to be called a true S-100 system.

The S-100 bus is composed of 100 foiled contacts located on the end of an approximately 6 inch by 8 inch PC board. There are 50 contacts on each side of the bottom edge of the board, although the card can be mounted in any direction. Air flow and cabling off the top edge usually determine the position of the boards. One sign of a good board will be the location of the heat sink. It should be found at the top left hand corner of the board, providing for maximum heat dissipation with the least heating of other components. When located at the lower left hand corner, which is also closest to the bus source pins, heat will flow upward onto other devices, causing them to overheat, and possibly resulting in system crashes.

Within the IEEE 696 Standard, there are several groups of signals—they are data paths, control paths, power, and system lines. System lines are typically reset, clock, and bus status. Memory paths include the address and data lines. The address can be up to 24 lines. The power standards are 8,16, and -16 volts DC. Regulation is usually done on the board, but a few manufacturers have done it off the board for better cooling and less noise. Control paths represent the control signals of the CPU and are typically I/O, memory, read, write, and hold. Data paths represent the 8 in data lines and the 8 out data lines, although some systems may use both for a true 16 bit data structure.

S-100 is not the only bus, but most of the other styles will have some variation of the type of data and signal paths found on the S-100. Failures which occur when new boards are installed in a system are usually due to incompatible timing signals caused by a differnce in CPUs or by pre IEEE-696 standards. Some trial and error fudging may correct these problems. Unfortunately, expensive digitial analyzers are needed for true timing analysis. These special scopes will provide pictures of many lines so that you can compare the timings against one another. Generally, these problems are in memory access and not I/O access. The I/O timing is usually the least affected from system to system, except for the disk operations, where multiple reads or writes must occur within a given time frame. DMA (Direct Memory Access) disk controller cards will pose many problems for the system builder. These cards must stop the CPU from reading or writing to memory while they control the system and do the reading or writing. If they hold the CPU in a wait state for too long, memory devices that use a CPU refresh signal will lose their memory. The Z80 stops refresh while in a wait state, and although most waits are rather short and the refresh count normally has some room for them, waits for long DMA operations may exceed the maximum time.

The most common problem I have encountered is the PDBIN signal. This 8080 style signal informs the devices when the CPU is ready for the data. This signal is not on the Z80 and must be created from the Z80 equivalents. When doing this, some designers ignore the old needs of 8080 cards with respect to the PSYNC signal. This signal sets the begining of a bus cycle and the falling edge should start PDBIN. Lots of Z80 cards let the PDBIN stay high and not go low during PSYNC, which is when some of the older memory boards will do a refresh. The use of an inverter and an AND device can gate PDBIN to PSYNC and solve the problem.

Upgrading

When checking over the design of a new computer or replacement board, upgrading is one important design consideration. Upgrading refers to the ability to use newer style chips as they become available—this is usually reserved for the memory devices, where falling prices will make it possible to expand memory with little physical changes. These devices have been standardized by the industry, and the upgrading process may involve as little modification as changing one or two jumpers. Well

Circuit Analysis, continued from page 16

names I have used. Secondly, it would be more convenient to be able to use whole numbers (such as 10) in addition to the floating point and scientific notation provided. My third suggestion is also a minor matter of convenience—the program returns you to the system each time an improper file name is used, forcing you to reload the program. It would save time and frustration if the program returned you to the menu instead, with an error message stating that the file name was not found.

These criticisms are very minor, and ACNAP and the companion programs PLOTPRO and SPP are very useful and reasonably priced. ACNAP is currently available for CP/M, MSDOS, PCDOS, and TRSDOS for $69.95 from BV Engineering, Box 3429, Riverside, CA 92519 ■

documented systems with this kind of ability are a delight for system integrators who enjoy being on top of the current trends.

Memory is not the only place to look for changes; one CPU will soon be upgradable. National Semiconductor has made a CMOS CPU, the NSC800, for some time now. This device interfaces to the systems as if it were a 8085. The signal timing is based on the Intel device, which also has an built in clock circuit. The internal architecture however, is that of the Z80, complete with the expanded block instructions. For people like myself who run a Zenith Z100 with an 8085 CPU, this means that their soon-to-be-released plug compatible version (the older unit had a different pinout) will allow Z80 operation without any physical changes.

System Design Reviewed

When looking at a system, either an existing one or one you want to build, all of the above topics must be understood. In the case of a S-100 system, we will have a CPU card, a memory card, an I/O card, and most likely a Disk Controller. These components represent a complete system. All of these items are available on one card, but should a problem occur, trouble shooting can be rather difficult. My personal recommendation is a system with CPU and I/O on one card (ideally with enough RAM and ROM to run a monitor), a separate memory card with options to go to 256K of memory, and a disk controller card for both 8" and 5¼" drives. For faster system integration, a disk card that has a serial port is ideal. The manufacturer can then modify the BIOS for the known serial device, allowing you to bring the system up without having to modify the software first.

This design ideal has many reasons behind it. Substituting cards to determine a bad unit is now possible. Using the monitor to check memory or disk operation can get you started in the right direction quickly. Heat problems are reduced, as you have three sets of regulators on three separate cards (instead of one big one cooking the components mounted next to it). Spacing and repair work is easier because two-layer boards are used instead of multi-layer ones. There may not be any PALs or special devices that are supplied only by the manufacturer, which means that repair can be done by anyone. Lastly, the repair cost can be lowered by using less expensive and easier to obtain parts. I always consider the cost of repair or replacement in my cost of a system, and some of the cheaper products are repairable only by complete replacement at full cost value. The system specifications also meet the availablity of used items, as most products of older design meet the above design parameters.

Review

What has been discussed in this article is the whole system approach. What was not covered is how the software relates to the system. This discussion was meant to review some hardware design considerations which everybody needs to understand before approaching system purchases or modifications. ■

The Bookshelf

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 Programmer's CP/M Handbook

An exhaustive coverage of CP/M 80's 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. 7⅜x9¼, 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. 6¾x9¼, 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.....$16.95

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

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 microcomputers, Tandy Corporation's Radio Shack TRS-80, and others. By Murray Sargent III and Richard Shoemaker. 288 pages. 8½x8¼, softbound.....$17.95

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 special 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

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 UC Berkeley extension courses on FORTH. By Anita Anderson and Martin Tracy. 218 pages. 5½x8¼, softbound.....$20.00

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, 5½x8½, soft. © 1974.....$12.95

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
	Book Total		
	Shipping		
	TOTAL		

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

Searching for Useful Information?

The Computer Journal is for those who interface, build, and apply micros. No other magazine gives you the fact filled, how-to, technical articles that you need to use micros for real world applications. Here is a list of recent articles.

Volume 1, Number 1:
•The RS-232-C Serial Interface, Part One
•Telecomputing with the Apple][: Transferring Binary Files
•Beginner's Column, Part One: Getting Started
•Build an "Epram"

Volume 1, Number 2:
•File Transfer Programs for CP/M
•The RS-232-C Serial Interface, Part Two
•Build a Hardware Print Spooler, Part One: Background and Design
•A Review of Floppy Disk Formats
•Sending Morse Code With an Apple][
•Beginner's Column, Part Two: Basic Concepts and Formulas in Electronics

Volume 1, Number 3:
•Add an 8087 Math Chip to Your Dual Processor Board
•Build an A/D Converter for the Apple][
•ASCII Reference Chart
•Modems for Micros
•The CP/M Operating System
•Build a Hardware Print Spooler, Part Two: Construction

Volume 1, Number 4:
•Optoelectronics, Part One: Detecting, Generating, and Using Light in Electronics
•Multi-user: An Introduction
•Making the CP/M User Function More Useful
•Build a Hardware Print Spooler, Part Three: Enhancements
•Beginner's Column, Part Three: Power Supply Design

Volume 2, Number 1:
•Optoelectronics, Part Two: Practical Applications
•Multi-user: Multi-Processor Systems
•True RMS Measurements
•Gemini-10X: Modifications to Allow both Serial and Parallel Operation

Volume 2, Number 2:
•Build a High Resolution S-100 Graphics Board, Part One: Video Displays
•System Integration, Part One: Selecting System Components
•Optoelectronics, Part Three: Fiber Optics
•Controlling DC Motors
•Multi-User: Local Area Networks
•DC Motor Applications

Volume 2, Number 3:
•Heuristic Search in Hi-Q
•Build a High-Resolution S-100 Graphics Board, Part Two: Theory of Operation
•Multi-user: Etherseries
•System Integration, Part Two: Disk Controllers and CP/M 2.2 System Generation

Volume 2, Number 4:
•Build a VIC-20 EPROM Programmer
•Multi-user: CP/Net
•Build a High-Resolution S-100 Graphics Board, Part Three: Construction
•System Integration, Part Three: CP/M 3.0
•Linear Optimization with Micros
•LSTTL Reference Chart

Volume 2, Number 5:
•Threaded Interpretive Language, Part One: Introduction and Elementary Routines
•Interfacing Tips and Troubles: DC to DC Converters
•Multi-user: C-NET
•Reading PCDOS Diskettes with the Morrow Micro Decision
•LSTTL Reference Chart
•DOS Wars
•Build a Code Photoreader

Volume 2, Number 6:
•The FORTH Language: A Learner's Perspective
•Build an Affordable Graphics Tablet for the Apple][
•Multi-user: Some Generic Components and Techniques
•Make a Simple TTL Logic Tester
•Interfacing Tips and Troubles: Noise Problems
•Write Your Own Threaded Language, Part Two: Input-Output Routines and Dictionary Management
•TTL Reference Chart

Volume 2, Number 7:
•Putting the CP/M IOBYTE To Work
•Write Your Own Threaded Language, Part Three: Secondary words
•Interfacing Tips and Troubles: Noise Problems, Part Two
•Build a 68008 CPU Board for the S-100 Bus
•Writing and Evaluating Documentation
•Electronic Dial Indicator: A Reader Design Project

Volume 2, Number 8:
•Tricks of the Trade: Installing New I/O Drivers in a BIOS
•Interfacing Tips and Troubles: Noise Problems, Part Three
•Beginner's Project: 555 Timer Breadboard
•LSTTL Reference Chart
Multi-user: Cables and Topology
•Write Your Own Threaded Language, Part Four: Conclusion

Back issues: $3.25 in the U.S. and Canada, $5.50 in other countries (air mail postage included.) Send payment with your complete name and address to The Computer Journal, PO Box 1697, Kalispell, MT 59903. Allow 3 to 4 weeks for delivery.