ISSN # 0748-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 16

$2.50 US

Debugging 8087 Code page 2

Using the Apple Game Port page 6

BASE:

Part Four in a Series on

How to Design and Write Your Own Database page 12

Using the S-100 Bus and the 68008 CPU page 14

Interfacing Tips and Troubles:

Build a "Jellybean" Logic-to-RS232 Converter page 20

THE COMPUTER JOURNAL
P.O. Box 1697
Kalispell, Montana 59903
406-257-9119

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

The subscription rate for twelve issues is $24 in the U.S., $30 in Canada, and $48 airmail in other countries. Please make payment in U.S. funds.

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

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

Address all editorial, advertising and subscription inquiries to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59901

Editor's Page

Small Company Comeback?

Back in the seventies — before micros — when people mentioned computers they meant the huge, expensive mainframes. These systems were almost always designed for batch processing where you would punch a card deck for your program, submit it to the data processing department to be run at their convenience, and wait hours or days for a printout of the results. If your program bombed, you had to revise it and reenter the long cycle to try it again. The development of minicomputers with interactive terminals was a great improvement because the programmer could work directly with the computer, but these systems still cost millions of dollars and required a lot of space and tons of air conditioning.

At that time computers were constructed of discrete components using thousands of individual resistors, capacitors, and transistors, and the equipment could not be made small or inexpensive. When the integrated circuit microprocessor was developed people started talking about the possibility of smaller systems, but the manufacturers wouldn't even consider the idea of an individual having a complete computer on their desk. They felt that the future of the computer market was in larger, more expensive systems, with batch type processing or possibly time sharing.

The only way to get your own computer at that time was to build it, so the hardware hackers scrounged around for parts and helped each other get their systems up and running. It wasn't long before small companies offered kits, and it mushroomed into the microcomputer industry of today. It is important to note that the initial development was done by very small, previously unknown businesses. MITS started with an idea and was swamped with orders for kits. Bill Godbout started selling kits out of an old hanger and developed the business into CompuPro. Steve Jobs and Steve Wozniak sold a VW Van and a HP calculator to start building Apples in a garage. The large, established companies didn't enter the market until long after it had been founded and proven by the pioneers. The spectacular technological advances were made by individuals or small start-up businesses; the big boys just added a few enhancements and a lot of promotional marketing when they decided that the field was ripe for picking.

"...it takes a minimum promotional budget of ten million dollars to bring out a major program in the business market."

The entry of big name companies established the microcomputer as legitimate for the business office environment, and opened the possibility of selling extremely large quantities of micros to a technically unsophisticated audience. These companies built factories for high volume production of a conservatively designed system intended to serve a wide variety of users, and spent millions on promotion. The result was that anyone competing with them in this market had to follow the same plan targeted towards large quantites and high promotional expenditures.

At this point it was no longer possible to start from the garage with a few dollars, at least not if you wanted to battle the big boys for a share of their market. One of the software publishers has said that it takes a minimum promotional budget of ten million dollars to bring out a new major program in the business office market. This means that only programs with

(continued on page 4)

DEBUGGING 8087 CODE

by Lance Rose

Calling all number crunchers! The good news is that the price of the 8087, Intel's floating point numeric processor, just fell again and it can now be bought for about $150. (Compare this with its initial price of $400.) In addition, you can now buy an 8MHz version of the chip (the standard version is rated at 5MHz), although the price is quite a bit steeper ($275). The bad news is that in order to use the chip, you need an assembler or compiler that will generate the floating point opcodes for it. So far, the compilers available that will do this have a pretty steep price tag attached to them ($300 and up) and for those of us on a limited budget, it may come down to a choice between buying either the math chip itself or the compiler without the math chip. Since the latter choice doesn't make any sense, we might look at some ways of using the 8087 that don't involve compilers.

I think it's only fair to say here that for math, science or engineering applications, a high level language is much better than assembly language, unless running the program at maximum speed is your primary goal. Debugging applications programs written in assembly language is tedious and frustrating and the program listings tend to be much longer (by about a factor of 5-10 in my experience) than the high level language equivalent. Still, if you don't have unlimited funds this may be your only choice.

My own system uses a Compupro 8085/8088 Dual Processor board which has been modified to incorporate an 8087 floating point processor (see "Add an 8087 Math Chip to Your Dual Processor Board" in Vol. I, No. 3 of **The Computer Journal**). My system runs CP/M-86 which is file compatible with CP/M-80, an important factor if you have a hard disk and want to switch back and forth between systems. CP/M-86 comes with an assembler called ASM86 and a debugger called DDT86. These function much the same as their CP/M-80 counterparts. In addition there is a macro library of 8087 instructions included with the system so that you can write programs which include floating point operations.

As an aside here, let me warn anyone using this library that it does not entirely agree with Intel's instruction set as far as the function of some instructions go. I have found it useful to fix up the library so as to be compatible with Intel's description of the opcodes as described in the Intel "iAPX 86,88 User's Manual" which is available from Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. Anyone who is interested in the changes required (they are minimal) can contact me through **The Computer Journal** for more information.

With the macro library available, it is possible to write assembly language applications programs. However, debugging them is another matter, since the DDT86 debugger has no way of examining or modifying the 8087 registers the way it does for the 8086/88 registers. Also, during program tracing all 8087 instructions show simply as ESC instructions with meaningless operands rather than the actual mnemonics (e.g. FADD ST,ST3 or FSQRT). While this latter problem is not very easy to fix, it isn't too hard to allow the display of the 8087's stack of 8 floating point registers as well as the control and status registers.

Modifications

My original plan for incorporating 8087 code debugging into DDT86 was to show all the 8087 registers along with the 8088 registers whenever the latter were displayed. Aside from the complexity of this approach, it would make it awkward to use the debugger for 8088-only code, thereby necessitating two separate debuggers on the disk. After some thought, I decided on a different method.

DDT86, like ordinary DDT, accepts single-letter commands, with or without arguments, to do things like Display memory, eXamine registers, Set memory and so forth. There is a jump table located within DDT86 that vectors to the proper routine corresponding to the letter of the command entered. Since many of the letters of the alphabet are unused by DDT86, it is a simple matter to appropriate one of them to display the 8087 registers. All that is required is to: (1) patch the jump table at the position of the desired letter code to jump to a new routine, (2) write the display routine and (3) merge it with the original DDT86 to get a new debugger, which I call DDT87.

The additional code necessary to implement the 8087 register display is the program listing shown in Figure 1. It is pretty straightforward and uses the 8087 instuctions FSAVE and FRESTOR to put the entire 8087 machine state into memory where it can then be examined and displayed by the 8088. I used the letter 'Z' as the instruction to do this, simply because it is positioned near the 'X' on the keyboard and I'm used to displaying the 8088 registers with the 'X' command. If you prefer a different letter it's a simple matter to patch the corresponding word in the jump table once you know where to find it. (It begins at 0369H relative to the beginning of the DDT86 file). Just make sure you don't choose a letter that's already in use or you will lose one of DDT86's standard functions.

After you have entered the patch program with a text editor or word processor, simply go through the procedure shown in Figure 2. This figure is simply a copy of the console commands required to assemble the patch, merge it with DDT86 and test its function. Once it is known to be working you can remove DDT86 from your working disk (**not** your archive diskette) and use DDT87 instead. When not examining 8087 registers it will function just as DDT86 would. The only difference is the 'Z' command.

Summary

I have found that the hardest thing in debugging 8087 code is not being able to see the floating point registers. This makes it hard to find floating point stack overflows and the like. This debugger patch rectifies this and makes floating point program debugging much easier. Some additional enhancements that might be nice would be to display the 8087 register values in floating decimal instead of hex, allow alteration of individual 8087 registers and display floating point opcodes with their proper mnemonics. I might decide in the future to add some of these, but the ability to actually examine the 8087 registers seems to solve the majority of the debugging problems. In retrospect, the patch is so simple that I wonder why I didn't add it sooner.

```
;        Patch to allow DDT86 display of 8087 registers
;        12/15/84
;
         CSEG
;
         ORG      3660H              ;End of DDT86
;
         FSAVE    STATE              ;Save 8087 state
         FRSTOR   STATE              ;Now restore it
         MOV      DX,OFFSET HDG1
         MOV      CL,9
         INT      224                ;Display first heading
         CALL     SPACE
         MOV      AX,STATE
         CALL     DISPW              ;Display control word
         CALL     SPACE
         MOV      AX,STATE+2
         CALL     DISPW              ;Display status word
         CALL     SPACE
         MOV      AX,STATE+4
         CALL     DISPW              ;Display tag word
         CALL     SPACE              ;Add an additional space
         MOV      SI,OFFSET STATE+22
         MOV      CX,3
         CALL     DISPFP             ;Display ST0,ST3,ST6
         MOV      DX,OFFSET HDG2
         MOV      CL,9
         INT      224                ;Display second heading
         MOV      SI,OFFSET STATE+32
         MOV      CX,3
         CALL     DISPFP             ;Display ST1,ST4,ST7
         MOV      DX,OFFSET CRLF
         MOV      CL,9
         INT      224                ;Do a CRLF
         MOV      AL,BYTE PTR STATE+9
         MOV      CL,4
         SHR      AL,CL
         CALL     DISPN              ;Display high nybble of IP
         MOV      AX,STATE+6
         CALL     DISPW              ;Display low word of IP
         CALL     SPACE
         MOV      AX,STATE+8
         AND      AH,07H
         OR       AH,0D8H
         CALL     DISPW              ;Display opcode
         CALL     SPACE
         MOV      AL,BYTE PTR STATE+13
         MOV      CL,4
         SHR      AL,CL
         CALL     DISPN              ;Display high nybble of OP
         MOV      AX,STATE+10
         CALL     DISPW              ;Display low word of OP
         MOV      SI,OFFSET STATE+42
         MOV      CX,2               ;Display ST2,ST5
DISPFP:  PUSH     CX
         PUSH     SI
         CALL     SPACE              ;One leading space
         POP      SI
         MOV      CX,5               ;Five words per register
DISPF1:  STD                         ;Set to decrement
```

continued

Editor's Page, *continued*

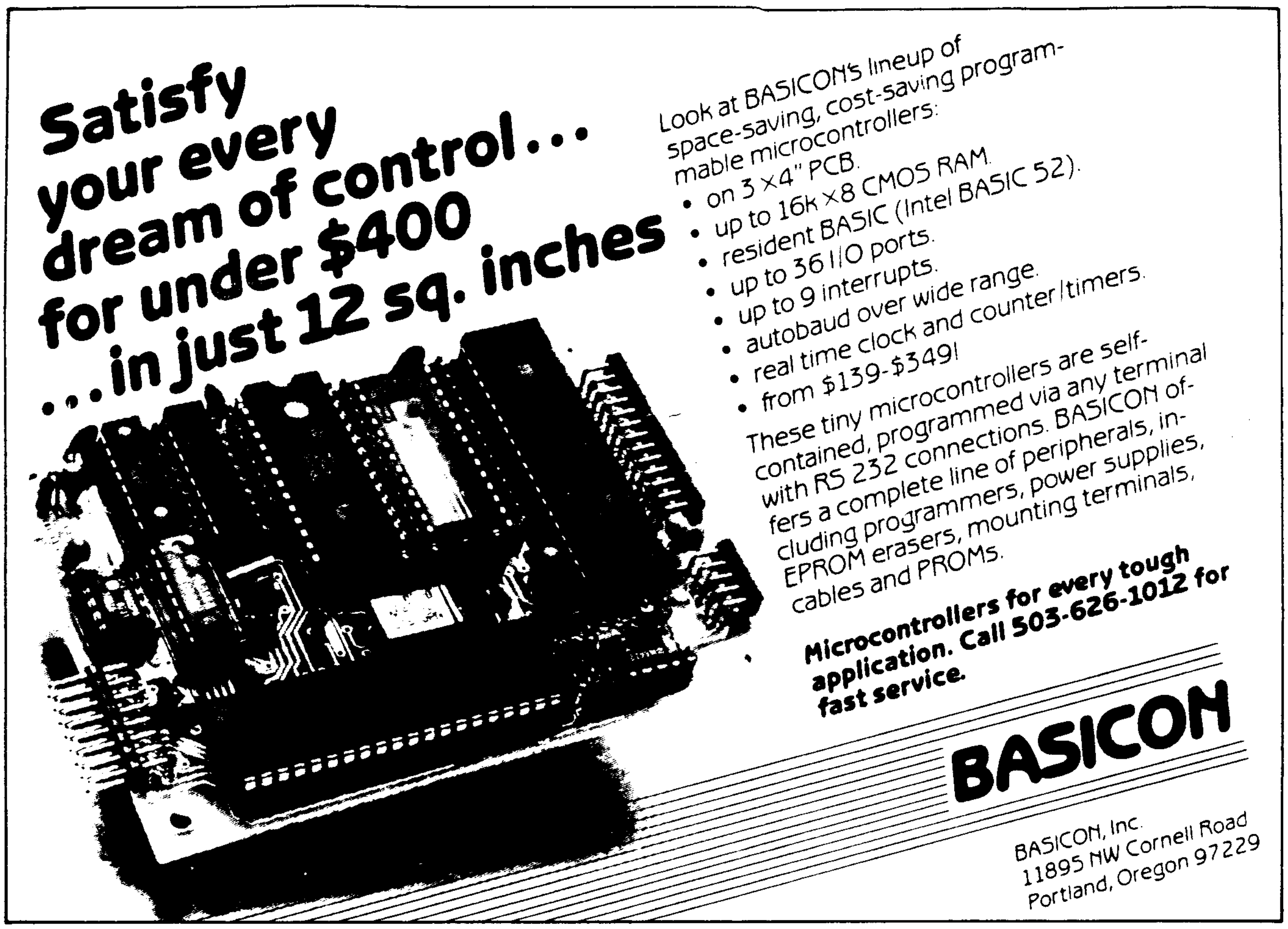
the potential for very large sales can be considered.

Those of us whose interests do not coincide with the business office market have been ignored, especially if we use an older system such as CP/M-80 which is not compatible with the IBM-PC. Yet there are thousands of people with fully-paid-for systems who are satisfied with what they are using, and do not want to buy one of the new systems just to be able to use the newest software. These people cannot be served by the large, high budget companies because of their diverse needs, but their business is attractive for smaller companies who can tightly target a specific market niche. There are indications of a strengthening market for speciality programs which are not limited to just the IBM-PC market. Some examples are the engineering programs from BV Engineering, multitasking MTBASIC from Softaid, Z80ASM assembler from SLR Systems, and SMAL/80 from Chromod. There is also a lot of activity in single board computers for the Z-80, 68000, and other CPUs, and in the area of enhancement boards for the Apple and other systems. Many of the people developing new products are those who have created something to fill their own needs, and are not necessarily employees of a large company. While these may not be garage shop operations, they are definitely not major corporations.

"There are thousands of people with fully-paid-for systems who are satisfied with what they are using..."

I feel that we have seen a peak in the growth of the "computers for everybody" market and that it will split up into several different fragments, among which will be the major business office portion for corporations and a smaller share for individuals and minor businesses. Perhaps it's time to clean out the spare bedroom, or the garage, and work on the products which are needed by the smaller markets. ■

```
          LODSW
          PUSH    CX
          PUSH    SI
          CALL    DISPW
          POP     SI
          POP     CX
          LOOP    DISPF1
          POP     CX
          ADD     SI,40
          LOOP    DISPFP
          RET
DISPW:    PUSH    AX                ;Display word
          MOV     AL,AH
          CALL    DISPB
          POP     AX
DISPB:    PUSH    AX                ;Display byte
          MOV     CL,4
          SHR     AL,CL
          CALL    DISPN
          POP     AX
DISPN:    AND     AL,0FH            ;Display nybble
          CMP     AL,10
          JB      DISPN1
          ADD     AL,7
DISPN1:   ADD     AL,'0'
          MOV     DL,AL
          MOV     CL,2
          INT     224
          RET
SPACE:    MOV     DL,' '            ;Display a space
          MOV     CL,2
          INT     224
          RET
;
ENDCS     EQU     OFFSET $
;
          DSEG
;
          ORG     ENDCS
;
HDG1      DB      0DH,0AH,'  CW    SW    TW              ST0-ST2  '
          DB      '                ST3-ST5                ST6-ST7'
CRLF      DB      0DH,0AH,'$'
HDG2      DB      0DH,0AH,'  IP    OC    OP    $'
;
MAXCOD    EQU     (OFFSET $+0FH) SHR 4    ;New code size
;
STATE     RW      47                ;Area for 8087 state
;
MAXPAR    EQU     (OFFSET $+0FH) SHR 4    ;New total size
;
          INCLUDE 8087.LIB
;
          END


A>ASM86 PATCH $SZ PZ

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  5%

A>GENCMD PATCH 8080

BYTES READ    012F
RECORDS WRITTEN 71


A>DDT86
DDT86 1.1
-RPATCH.CMD
  START      END
2000:0000 2000:387F
```




```
-^C
A>DDT86
DDT86 1.1
-RDDT86.CMD
  START      END
2000:0000 2000:367F
-S0
2000:0000 01
2000:0001 60 79
2000:0002 03
2000:0003 00
2000:0004 00
2000:0005 66 7F
2000:0006 03 .
-S1BE
2000:01BE 36 37
2000:01BF 20 .
-S39B
2000:039B DF 60
2000:039C 02 36
2000:039D CD .
-WDDT87.CMD,0,387F
-^C
A>DDT87
DDT87 1.1
-Z
  CW   SW   TW           ST0-ST2                 ST3-ST5                 ST6-ST7
 03FF 4100 FFFF  FFF6D5555575115C7555 FFF6D5555575115C7555 FFF6D5555575115C7555
  IP   OC   OP   FFF6D5555575115C7555 FFF6D5555575115C7555 FFF6D5555575115C7555
00000 D800 00000 FFF6D5555575115C7555 FFF6D5555575115C7555
-^C
A>
```

USING THE APPLE GAME PORT

by Art Carlson

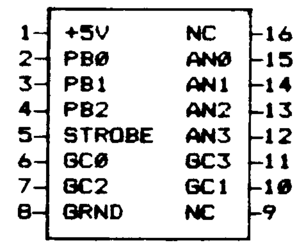
When Woz designed the Apple® he included a very useful port to interface the joysticks, paddles, and switches used for games, and named it the game port. This port can be used for many other applications, and it is unfortunate that because of its name people do not consider it for other uses.

The game port, which is available from a 16 pin DIP (Dual Inline Package) socket on the motherboard, contains four analog inputs which respond to variable resistance, four one bit outputs called "annunciators" which can be used as an input to some other device, three one bit inputs which can be used to sense the position of a switch or the state of an electronic device, and a strobe output. These features can be used from either assembly language or BASIC and can provide many interfacing functions without the added expense or slot space of a plug-in board.

The pin-out for this connector is shown in Figure 1 as viewed from the top of the motherboard. You can make the connections directly to the socket, but for experimental work where you will be changing the parts frequently, it is more convenient to use a 16 conductor jumper cable (Radio Shack #276-1976A) to bring the connections outside of the computer to a prototyping breadboard (Radio Shack #276-174). This also saves wear and tear on the motherboard socket. A permanent circuit for a small device could be assembled on a 16 pin header and plugged directly into the socket after being tested on the breadboard.

A Simple Starter Project

The first project, as shown in Figure 2, demonstrates the use of the annunciator outputs to control a device. It is a very simple project which almost anyone can do. The experienced hardware hackers can just skim over this section, but if this is your first attempt, roll up your sleeves and get started.



	FUNCTION	HEX	DEC	READ OR WRITE
PBØ	FLAGINPUT ZERO	$CØ61	49249	R
PB1	FLAGINPUT ONE	$CØ62	4925Ø	R
PB2	FLAGINPUT TWO	$CØ63	49251	R
ANØ	ANN.ZERO OFF	$CØ58	4924Ø	R/W
ANØ	ANN.ZERO ON	$CØ59	49241	R/W
AN1	ANN.ONE OFF	$CØ5A	49242	R/W
AN1	ANN.ONE ON	$CØ5B	49243	R/W
AN2	ANN.TWO OFF	$CØ5C	49244	R/W
AN2	ANN.TWO ON	$CØ5D	49245	R/W
AN3	ANN.3 OFF	$CØ5E	49246	R/W
AN3	ANN.3 ON	$CØ5F	49247	R/W
GCØ	ANALOG INPUT	$CØ64	49252	R
GC1	ANALOG INPUT	$CØ65	49253	R
GC2	ANALOG INPUT	$CØ66	49254	R
GC3	ANALOG INPUT	$CØ67	49255	R
	STROBE	$CØ4Ø	49216	R
	ANALOG CLEAR	$CØ7Ø	49264	R/W

Figure 1

Our philosophy is to have you start at a level where you can succeed and build from there, rather than start you on something way over your head and have you fall flat on your face.

The first step is to acquire a prototyping breadboard, a 16 pin jumper cable, two LEDs, and two 330 ohm resistors. Although I have occasionally used wires poked into the socket with the other end clipped or soldered to the components, I do encourage you to get the breadboard and cable. They can be used over again for other projects, and are a small investment (about $15) which will make things much more convenient.

It may appear to be overkill to use a computer to flash a couple of LEDs, but they are used here because they are cheap, easy to get, and provide visual feedback on what is happening while avoiding the additional complications involved in driving more demanding devices. In an actual application you would be controlling a motor, relay, heater, or some other device.

The annunciators are controlled by soft switches, with two memory locations assigned to each annunciator. Reading or writing to one location will turn the switch on, and reading or writing to the second location will turn the switch off. The value written to or read from the location is meaningless; it is the action of referencing the location which sets the switch.

In this example we use AN0 to control an LED connected to pin 15 of the game port, and AN1 to control an LED connected to pin 14 (see Figure 1). AN0 is turned on from location 49241 (Hex $C059), and is turned off from location 49240 (Hex $C058). The Apple uses the $ symbol to identify a hexadecimal number. AN1 is turned on from 49243 ($C05B), and off from 49242 ($C05A).

The annunciator outputs are standard 74LS series TTL (Transistor-Transistor Logic) outputs from a 74LS259 addressable latch on the motherboard, and the Apple reference manual states that the outputs must be buffered if used to drive other than

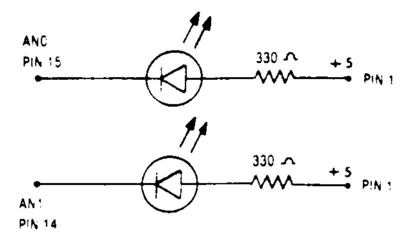


Figure 2

TTL inputs. TTL is called *current-sinking* logic because it can absorb or sink current to ground in the low state, but it can source or supply only a very limited current in the high state. The 74LS259 is rated as being able to source 0.4ma in the high state or sink 8ma in the low state. Since common LEDs require about 6ma, you have to either use the current sinking capability of the low state or use a buffer to drive the LED from the high state. Because of my early training on tube type equipment, I have had difficulty adjusting my thinking to turning something on with the logic in the low state, which is normally considered "off." Another alternative would be to use an inverter so that the device would be turned off with the TTL output low, but that also adds more parts for a simple LED driver. I have chosen to use the low state LED driver as shown in Figure 2. For more information on TTL logic, refer to page 7 of **Interfacing Microcomputers to the Real World** by Sargent and Shoemaker, page 6 of **TTL Cookbook** by Lancaster, or page 394 of **The Art of Electronics** by Horowitz and Hill.

The Applesoft BASIC program in Figure 3 will flash the two LEDs with the on and off times determined by the FOR–NEXT delay loops in lines 40 to 50, 70 to 80, 100 to 110, and 130 to 140. While this demonstrates the use of POKE statements to control the annunciators it doesn't have any useful applications. However, by using an analog input to read the value of a variable resistance you can vary the on and off times depending on this resistance.

```
]LIST

10  REM  CYCLE ANØ AND AN1
20  REM  1/16/85
30  POKE 49240,Ø: REM  TURN ANØ OFF
40  FOR I = 1 TO 250
50  NEXT
60  POKE 49241,Ø: REM  TURN ANØ ON
70  FOR I = 1 TO 250
80  NEXT
90  POKE 49242,Ø: REM  TURN AN1 OFF
100  FOR I = 1 TO 250
110  NEXT
120  POKE 49243,Ø: REM  TURN AN1 ON
130  FOR I = 1 TO 250
140  NEXT
150  GOTO 30
```

Figure 3

Assembly Language Programming of the Annunciators

Before tackling the game controller input, I want to cover using the annunciators from assembly language. I know that there were a lot of moans and groans when you saw the frightening words "assembly language," but it is not the intimidating beast that everyone thinks it is, and the speed of assembly language programs will be necessary for real time control in more advanced projects. Another advantage of assembly language programs is that you can place small driver programs in memory below HIMEM and call them from BASIC or other assembly programs.

Most assemblers can use either decimal or HEX addresses, but HEX addresses are much easier to use, especially since the memory pages break on HEX boundaries. A page in memory is the first byte of a two byte number. In other words, $0300 to $03FF is page three and covers $100 bytes (remember the Apple convention of defining a number preceeded by the $ symbol as a HEX number). The game port soft switches are located in page $C0 and are shown along with the decimal numbers in Figure 1 so that you can use them without doing any conversions.

The assembly language program in Figure 4 includes a nested delay loop so that the LEDs flash slowly enough for you to see. The HEX dump listing can be entered directly from the monitor if you do not have an assembler. The assembler source code is for the S-C Macro Assembler, but should work with most assemblers by changing the pseudo-opcode directives to suit your assembler. Lines 1050 thru 1080 establish the equates for addressing the four annunciator switches so that you can use the labels instead of the addresses in the source code. The assembler replaces the label with the

```
                  1000 *--------------------------------
                  1010 *      CYCLE VERSION 1.2
                  1020 * PROGRAM TO CYCLE ANØ AND AN1
                  1030 *      1/16/85   RAC
                  1040 *--------------------------------
CØ58-             1050 OFFØ     .EQ $CØ58
CØ59-             1060 ONØ      .EQ $CØ59
CØ5A-             1070 OFF1     .EQ $CØ5A
CØ5B-             1080 ON1      .EQ $CØ5B
Ø8ØØ- 8D 58 CØ    1090 CYØ      STA OFFØ     TURN ANØ OFF
Ø8Ø3- 2Ø 1B Ø8    1100          JSR DELAY
Ø8Ø6- 8D 59 CØ    1110          STA ONØ      TURN ANØ ON
Ø8Ø9- 2Ø 1B Ø8    1120          JSR DELAY
Ø8ØC- 8D 5A CØ    1130 CY1      STA OFF1     TURN AN1 OFF
Ø8ØF- 2Ø 1B Ø8    1140          JSR DELAY
Ø812- 8D 5B CØ    1150          STA ON1      TURN AN1 ON
Ø815- 2Ø 1B Ø8    1160          JSR DELAY
Ø818- 4C ØØ Ø8    1170          JMP CYØ
Ø81B- A2 FF       1180 DELAY    LDX #$FF
Ø81D- AØ FF       1190 LOOP1    LDY #$FF
Ø81F- 88          1200 LOOP2    DEY
Ø82Ø- DØ FD       1210          BNE LOOP2
Ø822- CA          1220          DEX
Ø823- DØ F8       1230          BNE LOOP1
Ø825- 6Ø          1240          RTS


SYMBOL TABLE

Ø8ØØ- CYØ
Ø8ØC- CY1
Ø81B- DELAY
Ø81D- LOOP1
Ø81F- LOOP2
CØ58- OFFØ
CØ5A- OFF1
CØ59- ONØ
CØ5B- ON1

ØØØØ ERRORS IN ASSEMBLY


*8ØØ.825

Ø8ØØ- 8D 58 CØ 2Ø 1B Ø8 8D 59
Ø8Ø8- CØ 2Ø 1B Ø8 8D 5A CØ 2Ø
Ø81Ø- 1B Ø8 8D 5B CØ 2Ø 1B Ø8
Ø818- 4C ØØ Ø8 A2 FF AØ FF 88
Ø82Ø- DØ FD CA DØ F8 6Ø
```

Figure 4

data established by the equate at assembly time as you can see in the HEX code in Figure 4. If you are not familiar with assembler listings, the first column is the target address, the next three columns are the actual HEX data being stored in those locations, the fifth column contains the line numbers used by the assembler for editing purposes (these line numbers are not a part of the program as in BASIC), the sixth column is the label, the seventh column is the op-code, the eighth column is the operand, and anything after that is a comment — similar to a REM statement in Basic.

Lines 1000 thru 1040 are comments to identify the program, and lines 1050 thru 1080 set up the equates. Line 1090, which is the start of the actual program (located at $800 in this example), uses the op-code STA (STore the Accumulator) to write to $C058 and turn off AN0. The HEX code for this is in columns two through four, where 8D is the code for STA, and 58 C0 is the address. The 6502 CPU used in the Apple stores two byte addressess, with the low byte (58) first followed by the high byte (C0). This is the opposite of the way you enter it in the assembler source code. You don't have to worry about this — the assembler takes care of it for you — but it can be confusing when you look at the HEX code memory dump. Line 1100 uses JSR (Jump to SubRoutine) to transfer operation to the address in the operand field. The assembler very conveniently replaced our DELAY operand with the address for DELAY in the label field. Lines 1180 and 1190 load the X and Y registers with the value $FF (when the operand is preceded by the symbol # it means load this number instead of the number in this address). Line 1200, DEY (DEcrement the Y register), subtracts one from the Y register. Line 1210, BNE (Branch Not Equal) LOOP2, executes the branch if the preceeding operation did not result in zero. If the result is zero, the program continues on to the next line. After the Y register has been decremented to zero line 1220, DEX (DEcrement the X register), subtracts one from the X register, and line 1230 loops back to line 1190 to reload the Y register with $FF and keeps on repeating this sequence until the X register has been decremented to zero. Then in line 1240, RTS (ReTurn From Subroutine), the program goes back to where the subroutine was called. After the first delay period, we store the accumulator in $C059 to turn AN0 on, and jump back to the delay loop. I'll leave it to you to follow the rest of the program.

The program is not elegant — for example, the only way to stop it is to use the reset — but it does show how to program the annunciators from assembly language.

Using The Analog Inputs

The four analog inputs were designed to read the position of joysticks by using variable resistance potentiometers attached to the joystick. Each analog input is connected to one section of a 558 quad timer integrated circuit on the motherboard (see the article "555 Timer Breadboard" in issue 12 for information on the timer IC). The reading subroutine simply counts the number of cycles required for the 558 to time out with the time determined by the variable resistance. Woz designed the circuit with a 0.022 microfarad capacitor in each section so that the count can be varied between zero and 255 with a 150 kilohm potentiometer.

To use the input, you connect a resistance between CG0 (pin 6) and five volts (pin 1), and read the value from BASIC with the command Y = PDL(0). Then the value in Y can be used by your program. The BASIC program in Figure 5 shows how a variable resistance, such as a thermistor, can be used to control temperature regulating devices.

One of the problems in controlling temperature is that if you use large heaters to bring the load up to temperature rapidly from a cold start and then switch the heater off when the proper temperature is reached, the temperature will continue to rise past the set point because of the stored energy in the heaters. When the temperature drops below the set point and you switch the heaters back on, the temperature will rise above the set point again. In a typical on-off application the temperature will continue to oscillate above and below the set points, and the fluctuations can be large. In servo control parlance this type of system where the power is either completely on or completely off is called a bang-bang control, and I like to compare it with trying to drive in city traffic with either the accelerator to the floor or the brakes completely locked with no in-between partial control. The program in Figure 5 provides much more advanced control than a simple on-off switch such as a thermostat because it allows you to proportion the response to the amount of

```
]LIST

100  REM  CYCLE AN0 AND AN1 PROPORTIONAL TO ERROR
110 RAC -   - 1 / 19 / 85
120  POKE 49241,0: POKE 49243,0: REM  START WITH BOTH LEDS OFF
130  REM  THE LEDS ARE OFF WHEN THE ANNUNCIATORS ARE ON
140 X =  PDL (0): REM   READ GC0
150  PRINT X
160 M = 5: REM  MULTIPLIER
170  IF X > 125 GOTO 210
180  IF X < 120 GOTO 300
190  FOR I = 0 TO 500: NEXT I
200  GOTO 140
210 H = M * (X - 125)
220 J = M * (255 - X): PRINT H,J
230  POKE 49240,0: REM  TURN AN0 OFF
240  FOR I = 0 TO H
250  NEXT I
260  POKE 49241,0: REM  TURN AN0 ON
270  FOR I = 0 TO J
280  NEXT I
290  GOTO 140
300 L = M * X
310 N = M * (120 - X): PRINT L,N
320  POKE 49242,0: REM  TURN AN1 OFF
330  FOR I = 0 TO N
340  NEXT I
350  POKE 49243,0: REM  TURN AN1 ON
360  FOR I = 0 TO L
370  NEXT I
380  GOTO 140
```

Figure 5

error.

Line 140 sets X equal to the reading from GC0, and line 150 prints the value to the screen. Line 160 is a multiplier to increase the length of the cycle. In a real world control situation, you will probably want to control from both sides of a set point to correct errors in either direction, so lines 170 and 180 determine if the result is above or below the set point, with a "dead band" created between 120 and 125. This "dead band" means that no control action will take place between 120 and 125, allowing us to avoid rapid cycling from heating to cooling. If the value is between 120 and 125 the program advances to the delay loop in line 190, then returns to line 140 to repeat the cycle.

If the value is above 125 the program goes to line line 210 where we set H equal to M times X minus 125 to establish the ON portion of the cycle. In line 220 we set J equal to M times 255 minus X to establish the OFF portion of the cycle, and print the values of H and J to the screen. The purpose of these two lines is to maintain the total length of the cycle equal to M times 255, but vary the ratio of the ON time to the OFF time with the ON time proportional to the difference between 125 and X. Line 230 POKES location 49240 to turn AN0 off, lines 240 and 250 are the delay loop using the value of H determined in line 210. Line 260 POKES location 49241 to turn AN0 on, lines 270 and 280 are the delay loop using the value of J determined in line 220, and line 290 returns the program to line 140 to repeat the cycle. I'll let you trace the section of the program for values less than 120 which starts in line 300.

You can tailor the program for your application by making a few minor revisions. For example, if controlling a relay you'll probably want to increase M in order to lengthen the cycle to avoid rapid cycling of the relay. You could raise H and J to some power so that the correction would increase more rapidly than the linear proportional control in the listing. You could also establish additional set points to activate an alarm or initiate some other action if the error exceeded certain limits. This BASIC program is fine for experimenting with the LEDs or for controlling something slow like a heater, but you'll need to use assembly language to control high speed motors, so fire up your assembler and tackle the next section.

In order to read GC0 from machine language, you load X with a number from 0 to 3 to determine which controller to read, and then use the monitor routine PREAD at $FB1E, which returns with a number between $00 and $FF in the Y register. (Note: the contents of the accumulator are scrambled during this process.) The assembly language listing in Figure 6 demonstrates reading GC0 and using the value to control an LED attached to AN0. Lines 1060 to 1080 establish the equates for the PREAD routine and the AN0 soft switches. Line 1090 calls PREAD to read the controller input, and line 1100 stores the value in a memory location. Line 1110 writes to $C058 to turn AN0 off (which turns the LED on), and line 1120 jumps to the delay routine. Line 1200 loads the value $FF in to X register, and enters a delay routine similar to that already encountered in Figure 4 except that we use the value in register Y obtained from the PREAD routine.

The portion of the program starting with line 1130 controls the off period of the LED. The first step is to turn the LED off by writing to $C059. Next we want to get a value equal to the difference between $FF and the value obtained from PREAD, so we load the accumulator with the value $FF in line 1140, set the carry in line 1150, and then in line 1160 we subtract the value in location TEMP. In line 1170 we transfer the value in the accumulator to register Y, and then jump to the delay routine. We had stored the value obtained from PREAD in the temporary memory location TEMP so that it would be available for use in CYOFF because both the X and Y registers would be decremented in CYON and we needed to load $FF into the accumulator before the subtraction. There are other ways to accomplish this, but I chose this method because it is simple and easy to follow.

```
1000 *-------------------
1010 *
1020 *  CYCLE.PROP.ML1.6
1030 *  1/19/85
1040 *
1050 *-------------------
1060 PREAD   .EQ $FB1E
1070 OFF     .EQ $C058
1080 ON      .EQ $C059
1090 CYCLE   JSR PREAD
1100         STY TEMP
1110 CYON    STA OFF
1120         JSR DELAY
1130 CYOFF   STA ON
1140         LDA #$FF
1150         SEC
1160         SBC TEMP
1170         TAY
1180         JSR DELAY
1190         JMP CYCLE
1200 DELAY   LDX #$FF
1210 LOOP    DEX
1220         BNE LOOP
1230         DEY
1240         BNE DELAY
1250         RTS
1260 TEMP    .BS 1
```

Figure 6

```
FB1E-   AD 70 C0    LDA   $C070
FB21-   A0 00       LDY   #$00
FB23-   EA          NOP
FB24-   EA          NOP
FB25-   BD 64 C0    LDA   $C064,X
FB28-   10 04       BPL   $FB2E
FB2A-   C8          INY
FB2B-   D0 F8       BNE   $FB25
FB2D-   88          DEY
FB2E-   60          RTS
```

Figure 7

One of the nice things about the Apple is the many subroutines in the monitor which are available for our use. In the above example we just used PREAD without concerning ourselves with how it works, but if you really want to learn assembly language you should examine the routine in order to understand what it does. You can use the monitor to display the routines in ROM by entering the monitor from Applesoft with the command CALL-151. The prompt will change to an asterisk and you can enter FB1EL (you don't enter the $ symbol because the monitor only understand HEX). This command will disassemble and list 20 memory locations starting with location FB1E as shown in Figure 7, and you can send the listing to your printer by entering the slot number for your printer card followed by a "control P" before the list command.

The first command at $FB1E is LDA $C070 which resets bit seven of the four locations $C064 thru $C067 to 1. Then it loads the Y register with 00 and follows with two NOP (No OPeration) statements to pad out the routine for the desired time. Next it loads the accumulator with the value in location $C064 indexed by the value in the X register which selects the controller to read. The line BPL FB2E returns to the calling routine if bit seven of the value

continued on page 13

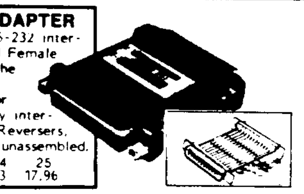
CJ1

L-COM® Shopper's Guide

Products for the ELECTRONIC & COMPUTER Industry

RS-232 DATA JUMPER BOX ADAPTER

A Data Jumper Box used to customize RS-232 interface devices. All 25 pins of the Male and Female connectors terminate to 25 solder pads. The PC Board is already wired to the D-Subs Supplied with 25 wires already stripped for permanent wiring to suit your needs. Many interfaces could be built - Null Modems, Pin Reversers, you name it! Supplied with all hardware unassembled.

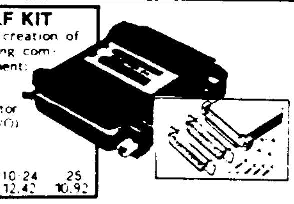


MODEL	DESCRIPTION	1-9	10-24	25
DJB	Jumper Box	22.45	20.43	17.96

RS-232 DO IT YOURSELF KIT

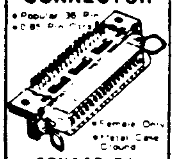
This Do It Yourself Kit allows the creation of many RS-232 variations. The following components are included in the assortment:

- 1ea. Male 25 pin D-Sub Connector
- 1ea. Female 25 pin D-Sub Connector
- 2ea. Snap-on half covers (UL 94 VO)
- 1 set Hardware for Male & Female Connectors



MODEL	DESCRIPTION	1-9	10-24	25
DIY	Do It Yourself Kit	13.65	12.42	10.92

CENTRONICS RT ANGLE PC CONNECTOR



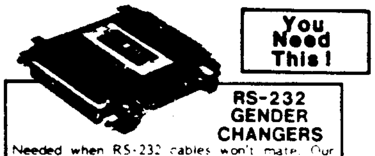
•Female Only
•Metal Case Ground

CEN36F-RA

1-9	10-24	25
8.95	7.88	8.50

You Need This!

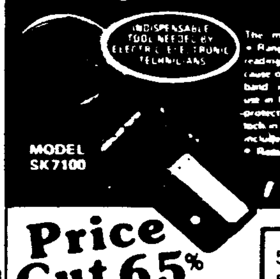
RS-232 GENDER CHANGERS



Needed when RS-232 cables won't mate. Our DC Series are compact and feature trouble free PC Board construction and lower cost. Choice of shielded or unshielded. Keep some for emergency needs.

MODEL	GENDER	1-9	10-24	25
DC-25F	(F-F)Unshielded	16.95	15.42	13.56
DC-25M	(M-M)Unshielded	16.95	15.42	13.56
DCS-25F	(F-F)Shielded	18.65	16.97	14.92
DCS-25M	(M-M)Shielded	18.65	16.97	14.92

Rotary-Scale Clamp-On Instruments



The most advanced Clamp-On Tester with many performance features • Range selection knob automatically advances the specific scale • Easy reading no confusion • Accurate AC current measurements assured because of round clamp core with built in balanced coils • Most reliable taut band internal core magnet meter • Trigger lock device retains reading for use in hard to get places • Automatic Zero Adjust ohms scale • Overload protection up to 150 for one minute ON ALL RANGES • Test leads lock in for added safety • Accuracy ±3 full scale on all ranges • Accessories included carry case test leads and instructions • Size 8.25×3.25×1.4 • Reasonably priced

10 USEFUL RANGES	
Current	6 15 60 150 300 600A
Voltage	150 300 600V
Resistance	0-20

Price Cut 65%

57.95/ea	47.60/10
55.55/3	42.85/25

Reg. $119.00

THIS MONTH'S SPECIAL!

RS-232 DATA LINE MONITOR

A miniature Data Line Monitor that allows the user to determine the status of the seven key signals of the RS-232 data path. All 25 pins wired through, and dedicated Red LED's report the status of:

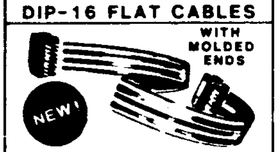
- Transmit Data (TD) pin 2
- Receive Data (RD) pin 3
- Request To Send (RTS) pin 4
- Clear To Send (CTS) pin 5
- Data Set Ready (DSR) pin 6
- Data Carrier Detect (DCD) pin 8
- Data Terminal Ready (DTR) pin 20

LED's glow at 3V & draw only 3 ma at 12V

MODEL	DESCRIPTION	1-9	10-24	25
DLM	Data Line Monitor	37.35	33.99	29.88

SUPERIOR GRADE DIP-16 FLAT CABLES

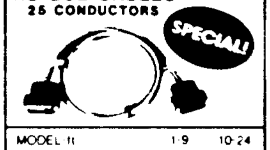
WITH MOLDED ENDS



MODEL-ft	1-9	10-24	25
MLF16PP-0.5	3.30	3.00	2.84
MLF16PP-1	3.60	3.28	3.10
MLF16PP-2	4.20	3.82	3.61
MLF16PP-3	4.80	4.37	4.13
MLF16PP-4	5.40	4.91	4.64

RS-232 CABLES

25 CONDUCTORS



MODEL-ft	1-9	10-24
C25-2.5	19.88	18.09
C25-5	21.16	19.26
C25-10	23.76	21.62
C25-15	26.36	23.99
C25-25	31.56	28.72

SPECIFY: Male-Male or Male-Fem

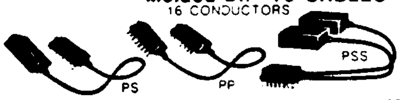
COAXIAL CABLES

W/Molded BNC Connectors On Each End

MODEL-ft	1-9	10-24	25-99
RG58C/U - for Instrumentation Use			
CC58C-1	3.55	3.23	3.05
CC58C-2.5	4.10	3.73	3.53
CC58C-5	4.65	4.23	4.00
CC58C-10	6.05	5.51	5.20
CC58C-15	7.45	6.78	6.41
CC58C-25	10.25	9.33	8.82
CC58C-50	17.25	15.70	14.84
CC58C-100	31.25	28.44	26.88
RG59B/U - for Video & General Use			
CC59B-1	3.65	3.32	3.14
CC59B-2.5	4.00	3.64	3.44
CC59B-5	4.75	4.32	4.09
CC59B-10	6.25	5.69	5.38
CC59B-15	7.75	7.05	6.67
CC59B-25	10.75	9.78	9.25
CC59B-50	18.25	16.61	15.70
CC59B-100	33.25	30.26	28.60
RG62A/U - for Computer Applications			
CC62A-1	3.60	3.28	3.10
CC62A-2.5	3.95	3.59	3.40
CC62A-5	4.70	4.28	4.04
CC62A-10	6.15	5.60	5.29
CC62A-15	7.60	6.92	6.54
CC62A-25	10.50	9.56	9.03
CC62A-50	17.75	16.15	15.27
CC62A-100	32.25	29.35	27.74
RG174U - for General Use			
CC174-1	3.60	3.28	3.10
CC174-1.5	3.75	3.41	3.23
CC174-2	3.90	3.55	3.35
CC174-2.5	4.05	3.69	3.48
CC174-4	4.50	4.10	3.87
CC174-5	4.80	4.37	4.13
CC174-10	6.35	5.78	5.46
CC174-15	7.90	7.19	6.79
CC174-25	11.00	10.01	9.46
CC174-30	12.55	11.42	10.79

Molded DIP-16 CABLES

16 CONDUCTORS



MODEL-ft	GENDER	1-9	10-24	25-99
ML16PP-1	M-M	6.85	6.23	5.89
ML16PP-2	M-M	7.15	6.51	6.15
ML16PP-3	M-M	7.45	6.78	6.41
ML16PS-1.5	M-F	6.95	6.32	5.98
ML16PSS-1.5	M-(2)F	11.45	10.42	9.85
ML16PSC-4*	M-F	8.95	8.14	7.70

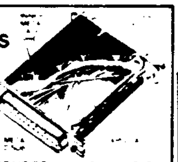
*Coiled Cord 22" Unstretched

BNC TEST ADAPTERS

MODEL	TYPE	TERMINATION	1-9
BC30	Plug	Banana Plug	7.75
BC40	Plug	Test Clip	8.95
BC50	Plug	Alligator	7.10
BC60	Jack	Banana Plug	5.75
BC70	Jack	Test Clip	6.95
BC80	Jack	Alligator	5.75

Order 10 each type - DEDUCT 10%

GENDER CHANGERS

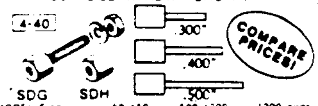


MODEL	GENDER	1-9	10-24
MG-25F	F-F	23.95	21.79
MG-25M	M-M	23.95	21.79

ATTRACTIVE Prices for CTY Users

RS-232 LOCK-DOWN SCREWS

SUPPLIED W/HDWR SHOWN



MODEL Size	50-450	500-1200	1200-over
SDC-300	.18	.17	.16
SDC-400	.19	.18	.17
SDC-500	.20	.19	.18
SDH	.11	.10	.09

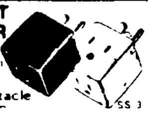
NOTICE: Multiples of 50 Only

Molded DIP-16 CABLES

16 CONDUCTOR-One end open

MODEL-ft.	GENDER	1-9	10-24	25-99
ML16P-1	Male	3.95	3.59	3.40
ML16P-2	Male	4.25	3.87	3.66
ML16P-3	Male	4.55	4.14	3.91
ML16S-1	Female	3.95	3.59	3.40
ML16S-2	Female	4.25	3.87	3.66
ML16S-3	Female	4.55	4.14	3.91

AC LINE TRANSIENT SURGE PROTECTOR



- Absorbs Spikes
- Safety Voltage Level
- Polarized Gnd'd Receptacle
- Rated: 15A 125VAC

MODEL	#taps	1-4	5-9	10-24
SS-1	Single	17.95	15.98	13.82
SS-3	Triple	19.95	17.56	15.36

Discounts offered in larger quantities

- POPULAR - GENDER CHANGERS

COMPARE

BNC

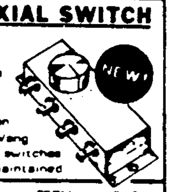
MIX 10 $2.80
MIX 25 $2.60

UHF

YOUR CHOICE 2.95

DATA COAXIAL SWITCH

- Rugged Build
- Fully Shielded
- AB or ABC Types
- Low Contact Res
- Mounting Flanges
- Metal Construction
- BNC or TNC & Wang
- Center conductor switches
- Shield isolation maintained



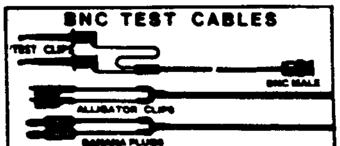
MODEL	FUNCTION	TERM	EACH
DS12B	SPDT AB	BNC	65.00
DS22B	DPDT AB	BNC	79.00
*DS22BT	DPDT AB	BNC-TNC	85.00
DS13B	SPDT ABC	BNC	75.00
DS23B	DPDT ABC	BNC	94.00
*DS23BT	DPDT ABC	BNC-TNC	99.00

*For Wang Devices-Switches Transmit & Receive Cables Simultaneously

Qty. Discounts & Custom Types Available

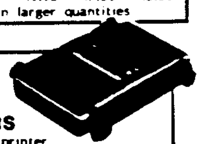
DEALER REQUESTS ATTENDED

BNC TEST CABLES



MODEL	TERMINATION	1-9	10-24
BC-01	Test Clip	8.95	8.06
BC-02	Alligator Clip	7.10	6.39
BC-03	Banana Plug	7.75	6.98

PARALLEL (Centronics) GENDER CHANGERS



Mates Centronics parallel printer cables that are of the wrong sex. All 36 pins are wired through. Bright metal construction. Mating Centronics cables are shown below.

MODEL	GENDER	1-9	10-24	25
DCS-36F	Female-Female	36.95	33.62	29.56
DCS-36M	Male-Male	36.95	33.62	29.56

GENDER CHANGERS

MODEL	GENDER	1-9	10-24
MG-9F	F-F	15.95	14.51
MG-9M	M-M	15.95	14.51
MG-15F	F-F	18.95	17.24
MG-15M	M-M	18.95	17.44

MINIMUM ORDER $25.00 • ADD $4.50 Shipping & Handling • You're Welcome to Visit Us

L-COM DATA PRODUCTS
1755 Osgood St. Rte 125
NO. Andover MA 01845

CHECK or M.O.

Toll Free Ordering
800-343-1455

MA Customers
617-682-6936
& Add 5% Sales Tax

BASE

A Series on How To Design and Write Your Own Database

By E.G. Brooner

Last month we covered the task of defining the file structure to our needs. Now, we can assume that files exist for our custom-designed storage system and we are ready to make some data entries. This is actually the easy part to program and to use. The DATA INPUT portion of the sample program is shown in Listing 1.

It opens by zeroing some arrays into which the program will read some of the housekeeping information that it previously stored, regarding such things as the field names for this data collection. Remember, we are accessing several of these little databases from the same program so it has to 'customize' itself for the files we have chosen each time we use it.

It will then present us (on the screen) with a prompt for the information it wants until the record has been entered in the otherwise conventional manner. It will name the field and tell us what kind and amount of information will be acceptable. The data will be stored in an array until we approve the entry. As is customary with this kind of operation, the program will give us a chance to correct the data, or back out, before filing it on the disk. It's that simple.

Next, we might want to edit an existing record. This section functions much like the data entry portion, except that we are presented with existing data rather than a blank record. We step through the fields in a similar manner, hitting RETURN to leave them unchanged or typing in the new, corrected information where necessary. (Note: Some BASICs do not accept a carriage return as an input variable.)

For simplicity we choose to call up the desired record by its relative number rather than by some key. This is fast and it is possible because if the record exists we probably already know the record number. If we don't, we can find it by other means in the 'search' section of the program; that will be taken up as our next subject. What is illustrated here is just a method of changing one or more fields of a record that has been called back and displayed on the screen.

There is not a great deal more to say about this section. The program already knows which base it is dealing with (from the opening section) and can pull out the necessary housekeeping and data files from among whatever else is there on our disk directory. Again, it reconstructs the file names, based on the core name we specified

```
REM     ******************
REM     *DATA INPUT MODULE*
REM     *******************

3000    GOSUB 9999
REM     PREPARE ARRAYS                REM FILL WITH BLANKS
        FOR X%=1 TO 12
                FIELD.NAME$(X%)=""
                FIELD.LENGTH%(X%)=0
                DATA$(X%)=""
        NEXT X%

        FILE$="B"+NAME$+".DEF"
        OPEN FILE$ AS 19
        IF END #19 THEN 3100

        FOR X%=1 TO 12                REM DISPLAY FIELD NAMES
                READ #19;FIELD.NAME$(X%),FIELD.LENGTH%(X%)
                IF FIELD.LENGTH%(X%)=0 THEN 3100
                PRINT X%;"  "; FIELD.NAME$(X%);
                PRINT TAB(20);FIELD.LENGTH%(X%);" CHAR ";
3050            INPUT LINE DATUM$
                IF LEN(DATUM$)<=FIELD.LENGTH%(X%) THEN 3060
                PRINT "DATA TOO LONG FOR FIELD * RE-ENTER"
                GOTO 3050
3060            DATA$(X%)=DATUM$   REM SAVE IN ARRAY
        NEXT X%

3100    CLOSE 19

        GOSUB 9999
        FOR X%=1 TO 12                REM DISPLAY FOR APPROVAL
                IF FIELD.LENGTH%(X%)=0 THEN 3200
                PRINT FIELD.NAME$(X%);TAB(20);DATA$(X%)
        NEXT X%

3200    INPUT "APPROVED ? (Y/N)";APPROVE$
        IF APPROVE$<>"Y" THEN 3000       REM FILE IT


        FILEXT$="B"+NAME$+".EXT"
        OPEN FILEXT$ RECL 5 AS 20
        READ #20,1;LAST%              REM PREVIOUS FILE LENGTH
        FOR X%=1 TO 12                REM ALL POSSIBLE FIELDS
                IF FIELD.LENGTH%(X%)=0 THEN 3500
                FILE$="B"+NAME$+STR$(X%)+".DAT"
                OPEN FILE$ RECL FIELD.LENGTH%(X%)+5 AS X%
                PRINT#X%,LAST%+1;DATA$(X%)
                CLOSE X%
        NEXT X%                       REM DATA FILED
3500    PRINT #20,1;LAST%+1           REM & 'EXT' FILE UPDATED
        CLOSE 20

REM     CONTINUATION OPTION           REM IF MORE TO BE ENTERED
        PRINT "TYPE <M> FOR MAIN MENU, OR <CR> FOR MORE ENTRIES"
```

Listing 1

```
         INPUT LINE OPTION$
         IF OPTION$="M" THEN 1300
         GOTO 3000

REM      *************
REM      *EDIT MODULE*
REM      *************
4000     GOSUB 9999
         FILE$="B"+NAME$+".EXT"      REM FIND FILE LENGTH
         OPEN FILE$ AS 19
         READ #19;MAX%:CLOSE 19      REM CALL IT 'MAX'
4050     PRINT "RECORD TO CHANGE - MUST BE BETWEEN 1 AND ";MAX%
         INPUT REC.NBR%
         IF REC.NBR%<1 OR REC.NBR%>MAX% THEN 4050
         FILE$="B"+NAME$+".DEF"      REM  DEFINE THE FIELDS
         OPEN FILE$ AS 19
         IF END #19 THEN 4100

         FOR X%=1 TO 12              REM & READ THE DETAILS
               NBR.OF.FLDS%=X%-1
               READ #19;FIELD.NAME$(X%),FL%(X%)
               IF FL%(X%)=0 THEN 4100
         NEXT X%

4100     PRINT "AFTER EACH FIELD IS DISPLAYED:"
         PRINT "IF CORRECT PRESS <CR> OR ENTER NEW DATA":PRINT

         FOR X%=1 TO NBR.OF.FLDS%:F.L%=FL%(X%)
               PRINT FIELD.NAME$(X%);TAB(20);
               GOSUB 4200            REM YOUR CHANCE TO CHANGE IT
         NEXT X%

         PRINT "<CR> = ADDITIONAL CHANGE - <M>= RETURN TO MENU"
         INPUT LINE CHOICE$
         IF LEN(CHOICE$)=0 THEN 4100      REM ALL FINISHED?
         CLOSE 19:GOTO 1300

4200     FILE$="B"+NAME$+STR$(X%)+".DAT"
         OPEN FILE$ RECL F.L%+5 AS X%
         READ #X%,REC.NBR%;DATUM$:PRINT DATUM$;
         INPUT LINE CHANGE$               REM NEW DATA (IF ANY)
         IF LEN(CHANGE$)=0 THEN 4300      REM OR KEEP THE OLD
         PRINT #X%,REC.NBR%;CHANGE$
4300     CLOSE X%:RETURN
```

Listing 2

before making our main menu choice. See Listing 2.

Reviewing the BASE series to date: after the introductory discussion we covered opening the program, then the definition and creation of the related files and their record structure. In this issue we entered some data into the files that were created earlier, and showed how any portion of any record, in any of our data collections, could be edited or altered after it was already in the file. In the next column we'll take up some of the more database-like features, i.e: searching the files for particular key information. We will be able to relate these searches to any of several criteria that might be part of the record. While many simpler applications retrieve records only by relative number, or by a single 'key' field, we'll construct this program so as to permit 'finding' data by more flexible methods. We may also discuss additional methods just as a subject of interest. As this is a learning project, an attempt has been made to keep the sample program relatively uncomplicated while illustrating other techniques which may sometimes be desirable. ■

Apple Game Port, *continued*

read in the previous location is zero, which would indicate that the timer has completed its cycle. If bit seven is still one, the next operation (INY) increments the Y register to count the number of cycles, and the following operation (BNE FB25) terminates the loop if the Y register has been incremented past $FF, which would indicate that the resistance is in excess of 150 kilohms. The following operation (DEY) decrements the Y register so that it will contain the maximum value of $FF if it has been incremented to zero. It may be confusing when we talk about increasing a positive number to zero by adding another positive number to it, but the contents of an eight bit register will overflow and "wrap around" to 00 if one is added to the maximum value of $FF. The final command RTS at FB2E returns control to the program which called the subroutine, with the value in register Y containing the number of cycles required for the capacitor on the 558 timer to charge.

The assembly language routines should not be considered the best examples of programming practice—I just hacked out something to do what I wanted. I'm sure that Don Lancaster (Synergentics) or Bob Sander-Cederlof (S-C Software) could write much better code! I tried to include enough information on the assembly language routines to help people not familiar with them, but did not write them as a complete tutorial. I need your feedback on how much assembly language detail should be included in future articles.

Going Further

There are many cards available for the Apple which offer more advanced interfacing capabilities than can be obtained from the game port, but it is an interesting challenge to learn what can be done with the little-used game port. Jan Eugenides is working on an article about using the game port to drive a printer, and others have articles in progress that deal with using it to control stepper motors, DC motors, and other devices. This article has not mentioned the flag inputs or the strobe; I intend to cover both topics in a future article, and would like to include information on what you are doing with the game port. Your articles, letters, and comments are welcome. ■

References:

- **AppleII Applications** *by Marvin L. De Jong, published by Howard W. Sams.*
- **The Art of Electronics** *by Horowitz and Hill, published by Cambridge University Press.*
- **TTL Cookbook** *by Don Lancaster, published by Howard W. Sams.*
- **S-C Macro Assembler** *sold by S-C Software Corporation, P.O. Box 280300, Dallas, TX 75228.*

USING THE S-100 BUS AND THE 68008 CPU

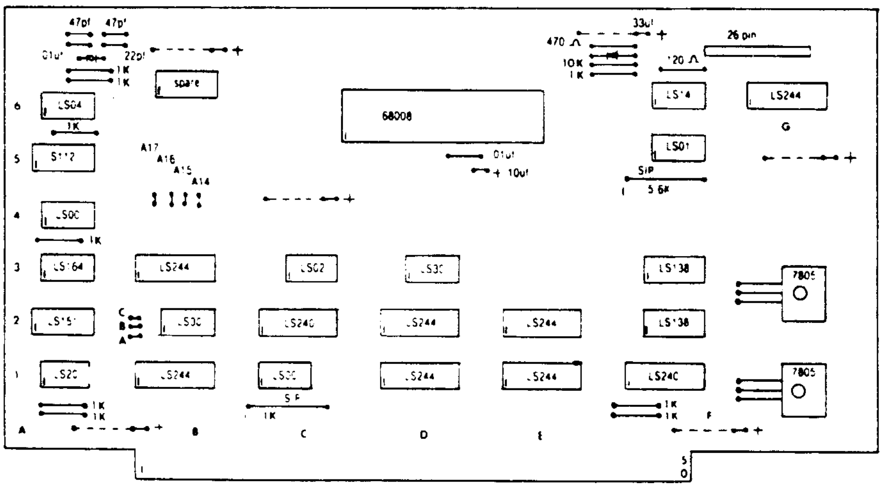
by Joseph Kohler, Kevin Jackson, and Bob Buckman
Wright State University

In 1978 the Computer Science department at Wright State University decided to develop a computer engineering program. To us, this meant "hands on the hardware." At that time we had an Intel system and several PDP-11 computers. All of the hardware had large PC boards with integrated circuits soldered to the boards. Repairs were either quite costly or very time consuming—i.e. expensive no matter how you looked at it. At this point we decided to investigate other systems which might be suitable for use by students in a hands-on computer lab.

At about that time I purchased a Cromemco S-100 kit which included a box with power supply, and a CPU card. The next step was a serial I/O card followed by some Seattle Computer 16K RAM cards. I could now run out of ROM—not much of a computer system, to be sure. The next step was a disk system. This started with the purchase of a pair of Innotronics floppy disk drives. I purchased a floppy disk controller board—it failed to work and the same thing happened when I tried again with a different vendor. I decided to build my own. The necessary parts consisted of a prototyping board, a 1771 disk controller chip and an assortment of TTL IC's. Many long hours were spent getting that board with the 1771 chip to work properly. With this accomplished, everything necessary for CP/M 1.4 was at hand. Once CP/M was up and running I felt I had a complete computer system. I was also confident that a laboratory of the kind we wanted could be put together at a reasonable cost.

The schematics for the disk controller board were given to an individual with a part time business in his home laying out PC boards. Then, with artwork in hand, I marched off to the PC house and had circuit boards made. This was followed by a simple serial I/O board. At about that time Digital Research of Texas was marketing a Z80 CPU board which proved to be perfect for our applications. The final step was the design and manufacture of a prototyping card with provisions for power and ground buses, bypass caps, regulators, connectors and easy access to the S-100 bus.

We finally put together a laboratory with seven S-100 systems using our own disk controller and serial I/O boards, Seattle Computer RAM cards, Digital Reaearch CPU boards and Innotronics disk drives. The operating system was CP/M 1.4, later upgraded to CP/M 2.2.



Board Layout

Why S-100?

Each quarter a class of relatively inexperienced students build circuit boards of their own design and plug them into the S-100 systems. In order to survive this heavy use the systems must be physically rugged and easily repairable. At the time the lab was set up, the choices available were multibus or S-100. The cost ratio was nearly four to one. Also, rugged mainframes of the S-100 style were not readily available for the multibus. The choice at the time was clear even considering the superior design of the multibus signal set.

The power distribution system on the S-100 bus ensures a certain amount of safety for all cards in the system. Switching power supplies, present in many computer systems, are efficient and compact, but they are more complicated and more easily damaged than the simple supplies in the S-100 systems. They are certainly more difficult to repair.

The reasons for using the S-100 bus as opposed to the better designed buses such as the multibus or IBM PC bus haven't changed much. An IBM PC would not survive the hard usage received by our present S-100 systems. If an IC on the PC motherboard were damaged, the whole system would go down. With an S-100 system, cards are swapped until the offending card is located, a spare is plugged into it's place, and we are up and running again. The bad cards are repaired when several of a kind are damaged.

Our reasons for continuing to use the S-100 systems are:

1. An inexperienced student can acquire a complete grasp of the hardware in a short period of time.

2. The systems are rugged, reliable and easily repaired.

3. The mainframes have large simple

linear power supplies and many slots for cards.

4. A completely new system can be put together in a matter of minutes by merely swapping a few cards.

On the negative side:

1. The S-100 bus has speed limitations.
2. The assertion levels of some of the signals are wrong.
3. The layout of the signals could be better.

Choosing a 16 Bit CPU

We decided to try one of the newer microprocessors but to continue to use our 8 bit hardware. Two choices were available, the Intel 8088 and the Motorola 68008. The latter was chosen because of its superior architecture. Here is a list of reasons for choosing the 68008:

Hardware considerations:

1. The hardware signals issued by the 68008 are straightforward and systematic.
2. Only a 5 volt supply and a simple, single phase clock are necessary to drive the 68008.

Programming considerations:

1. The chip has 8 address and 8 data registers.
2. The 68008 has two modes of operation; user mode, and supervisor mode. Address register A7 is used as the stack pointer for both modes. When the 68008 is in user mode A7 points to the user stack, and when in supervisor mode it points to the supervisor stack.
3. Except for A7 used as both a supervisor and user stack pointer, the address registers do not have special properties; i.e., instructions which use address registers are not tied to a particular address register.
4. Address registers can point to any memory location in the entire 1 megabyte address space of the 68008.
5. The data registers may be used interchangably; i.e., instructions which use data registers are not tied to a particular data register.
6. A wide range of addressing modes is available.
7. A wide range of opcodes is available.

A 68008 CPU Board Design

In the following paragraphs you will find the the technical details for a 68008

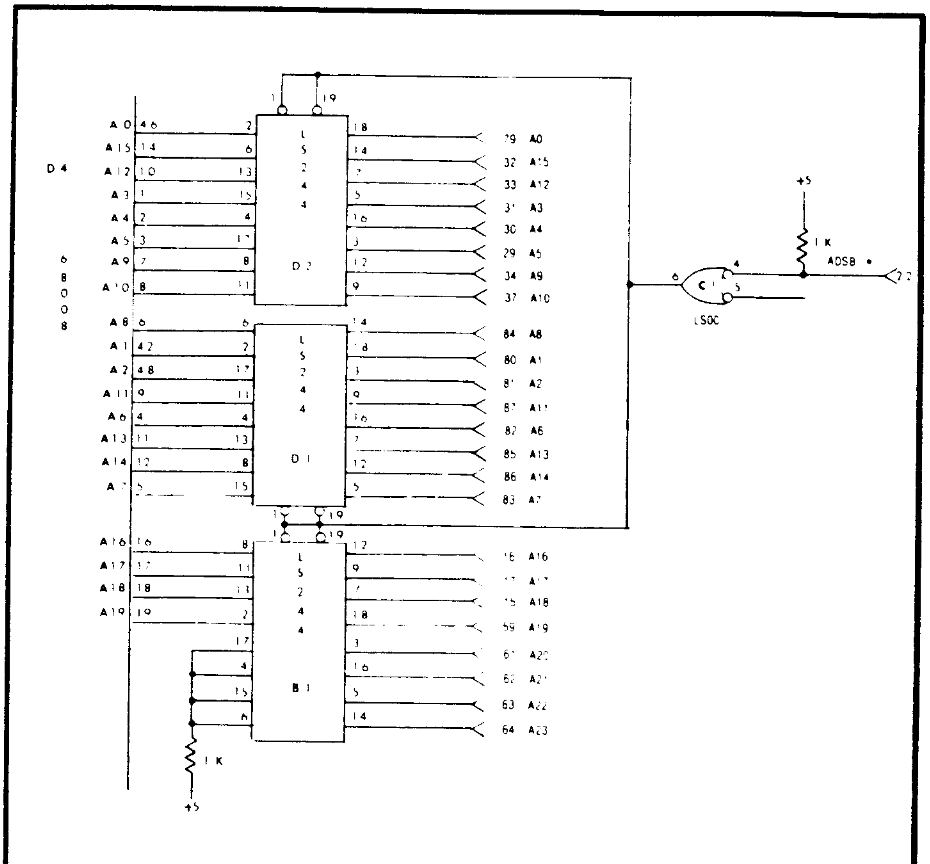


Figure 1

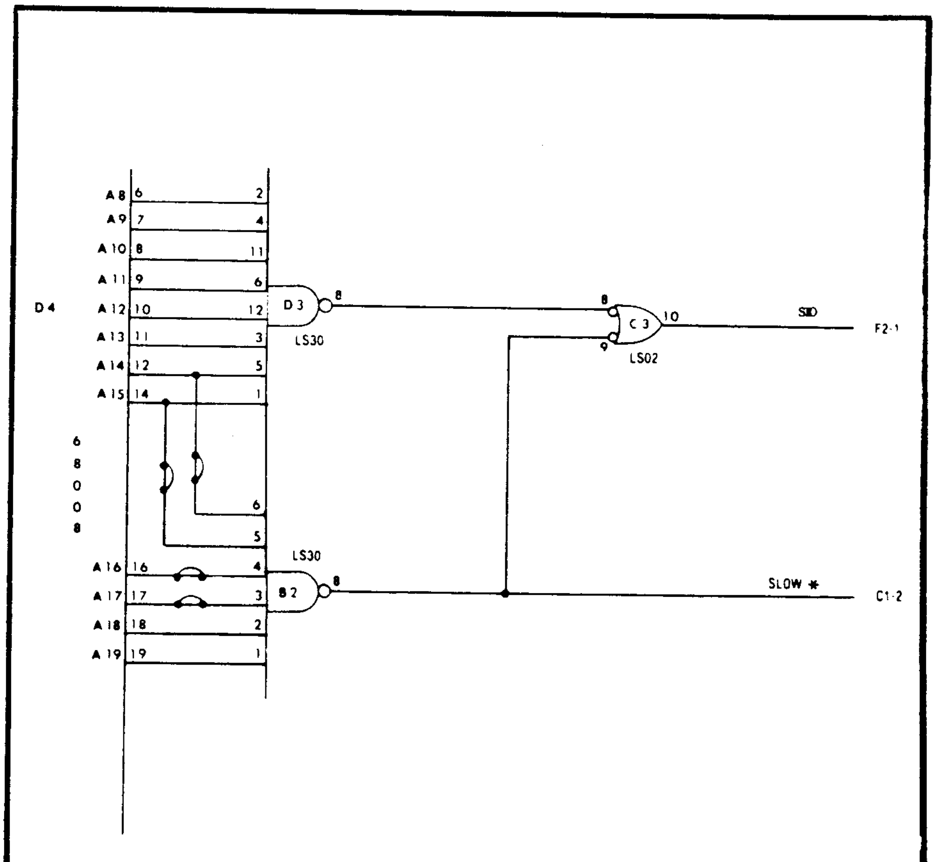


Figure 2

CPU board which resulted from a student design project. It is not IEEE-696, but it generates enough of the S-100 signals to fit many systems. The chief virtue of the board is its simplicity, its lack of PLA's or hard-to-find IC's, and the fact that it uses only 24 chips. You will probably notice that a number of the S-100 signals are not generated. Our goal was to build an S-100 board which worked with the boards we already had and to generate only those S-100 signals which seemed to derive in a reasonable way from the 68008 signal set. Actually this results in a fairly large subset of the S-100 signals being generated. When referring to signals by name, no assertion levels are included, for example NMI is written rather than NMI*. However, the schematics do include the assertion levels with the names.

Address lines A0 to A23. The S-100 bus has 24 address lines but the 68008 has only 20 address lines. First A0 to A19 of the 68008 are passed through LS244's to drive S-100 address lines A0 to A19. Lines A20 to A23 of the S-100 bus are forced high by tying the inputs of half of an LS244 high and attaching the outputs to lines A20 to A23. Since A20 to A23 are always high, no further mention of them will be made. All address buffers are enabled unless ADSB is asserted. Detailed schematics for the signals discussed in this paragraph are given in Figure 1.

Input and output. The 68008 has no special input or output instructions, and no part of its address space is set aside for I/O. The signal SIO (see Figure 2) is generated whenever address lines A8 to A19 of the 68008 are all high. Assertion of SIO will be necessary for the assertion of SINP or of SOUT (see Figure 3). The addresses $FFF00 to $FFFFF will be set aside to be used as I/O ports.

Data Out Lines DO0 to DO7. The lines DO0 to DO7 (see Figure 4) are driven by an LS244 whose inputs are tied to D0 to D7 of the 68008. This LS244 drives DO0 to DO7 whenever W (see Figure 3) on the 68008 is asserted and DODSB is negated.

Data In Lines DI0 to DI7. The lines DI0 to DI7 (see Figure 4) are inputs to an LS244 whose outputs are tied to D0

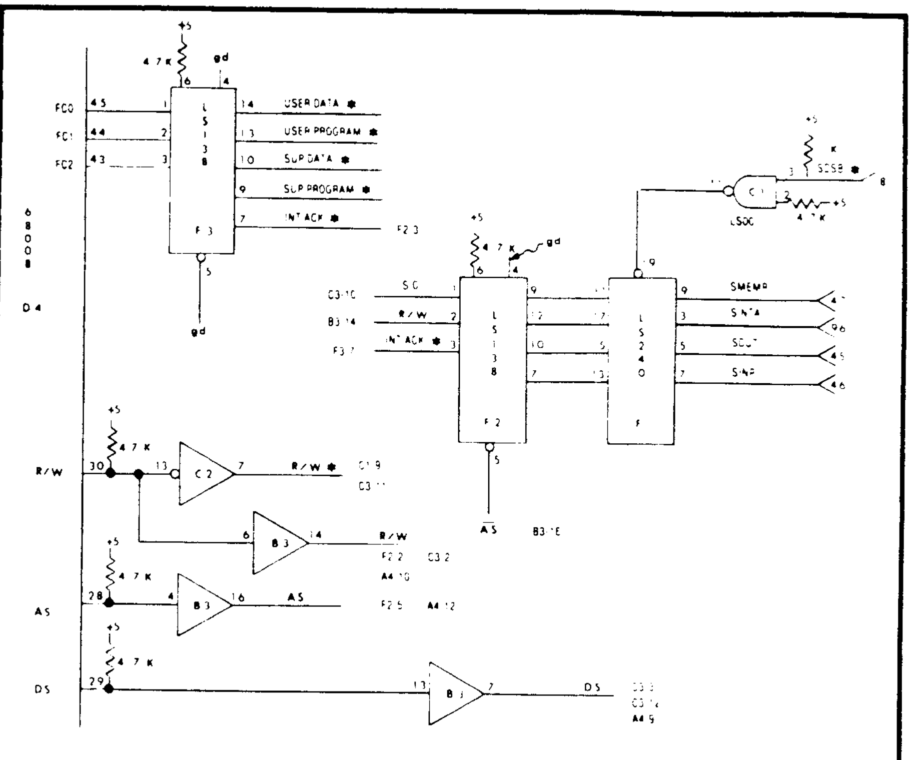


Figure 3

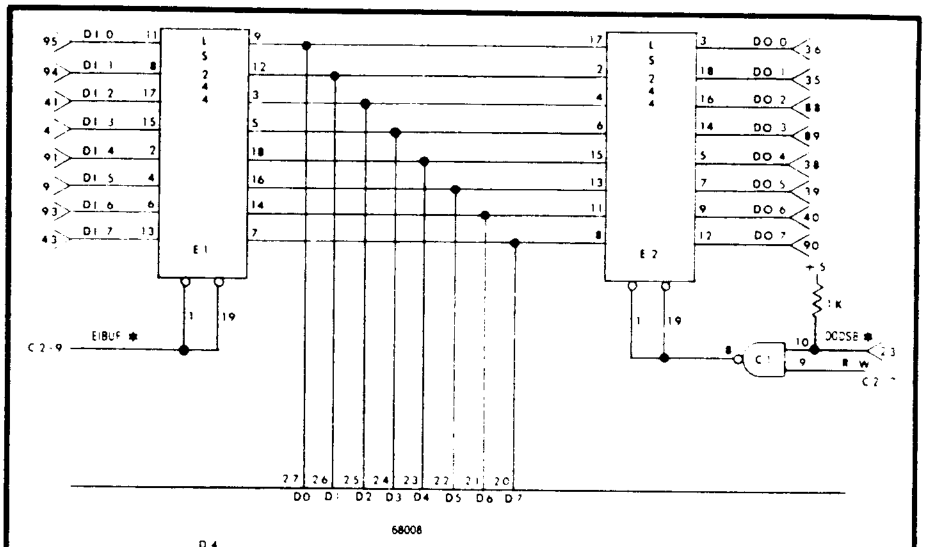


Figure 4

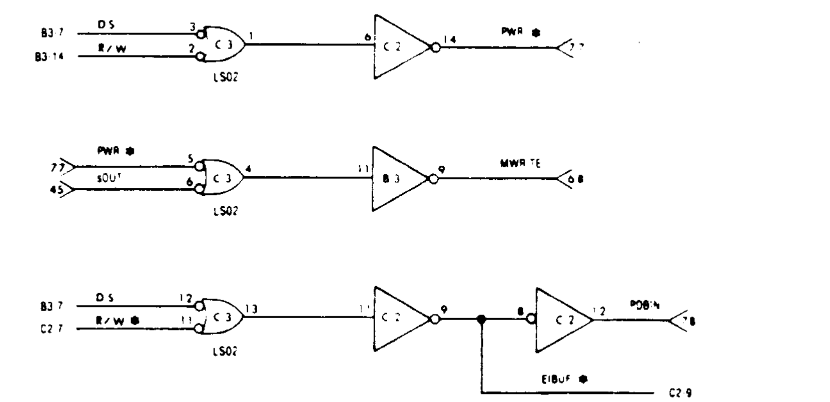


Figure 5

to D7 of the 68008. This LS244 is enabled by EIBUF which is asserted whenever both R and DS (see Figure 5) are asserted by the 68008.

Status Lines SINP, SOUT, SMEMR, SINTA. At the beginning of each bus cycle the 68008 tells us what it wants to do during the next bus cycle by placing a value of 0 to 7 at its function code outputs FC0 to FC2. The only one used on this board is interrupt acknowledge denoted by INT ACK (see Figure 3).

The three signals SIO, R/W and INT ACK are fed to an LS138 whose outputs are the status signals SINP, SOUT, SMEMR and SINTA in inverted form. This LS138 is enabled only when AS is asserted by the 68008 so these signals are asserted only when there is valid address information on the S-100 bus. These signals reach the S-100 bus with proper polarity by passing through half an LS240. This half LS240 is enabled except when SDSB is asserted.

Control Lines PWR, PHLDA, PDBIN. PWR (see Figure 5) is asserted whenever the 68008 asserts both DS and W. PDBIN (see Figure 5) is asserted whenever the 68008 asserts both DS and R. PHLDA (see Figure 6) is asserted whenever the 68008 asserts BG and negates (actually tristates) AS. The three control signals reach the S-100 bus with proper polarity by passing through one half an LS240. The half LS240 is enabled except when CDSB is asserted.

Reset. Whenever the reset switch is depressed a low is delivered to the RESET and HALT (see Figure 7) inputs of the 68008. The low is held for a period determined by the time necessary for the 33ufd capacitor to charge through the 10K resistor. The feedback circuit causes some hysteresis to be added but is overly conservative. The 68008 seems to work well enough without it.

Clock. The clock circuit (see Figure 8) has appeared in the literature. The S112 was used to achieve sharp rise and fall times as well as a 50 percent duty cycle (see Reference 1 at the conclusion of this article).

Interrupts. Assertion of NMI (see Figure 6) causes the 68008 to see a low at inputs IPL1 and IPL2/0 which it

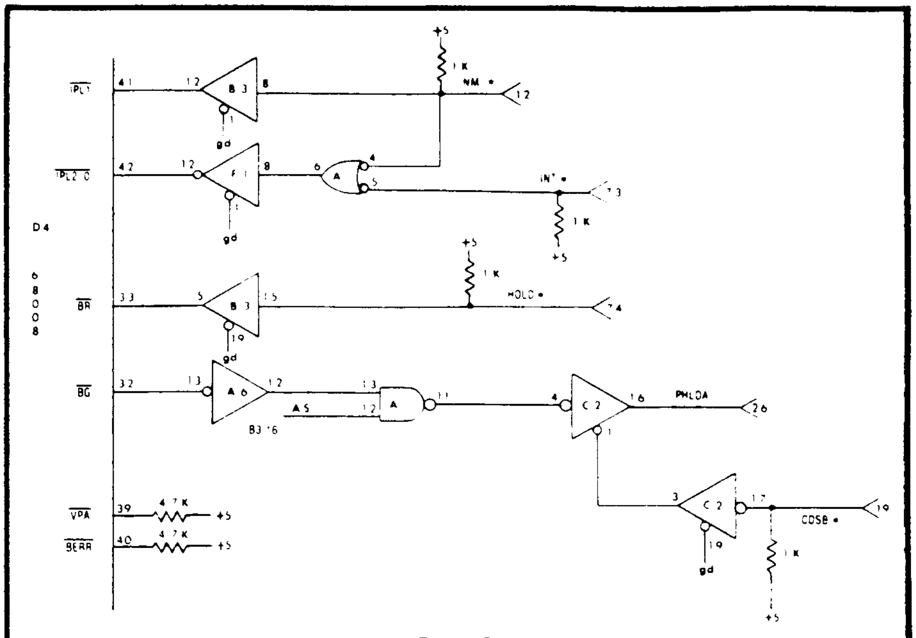


Figure 6

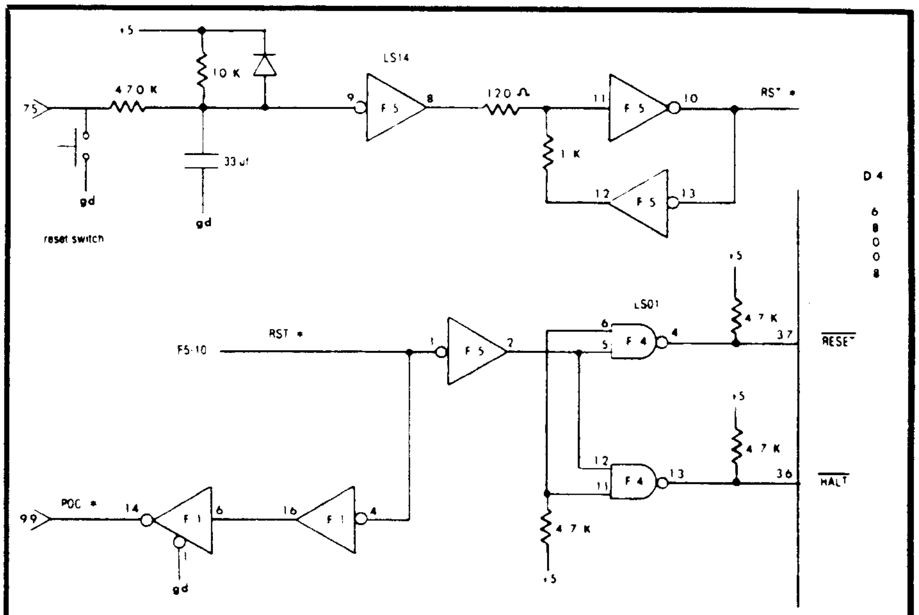


Figure 7

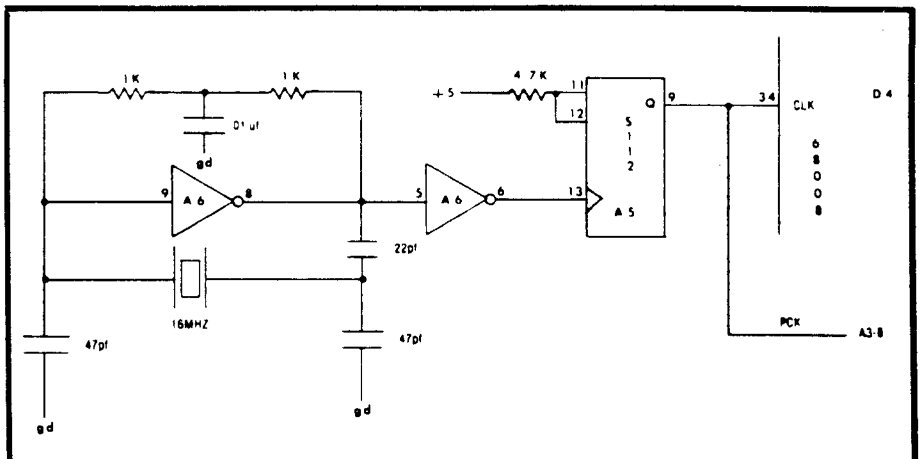


Figure 8

regards as a level 7 interrupt. To the 68008, this is an edge triggered non-maskable interrupt. Assertion of INT causes the 68008 to see a low at its ILP2/0 input, which the 68008 regards as a level 2 interrupt. To enable interrupts set the interrupt mask less than 2. To disable interrupts set the interrupt mask greater than 2.

SLOW and PHANTOM. Assertion of either PHANTOM or SLOW (see Figure 9) forces the insertion of wait states. SLOW (originates in Figure 2) is dependent upon placement of jumpers to determine which of A14 t0 A17 must be high in order to contribute to the assertion of SLOW. Assume A14, A15, A16, A17 are all connected (i.e. no trace under a jumper has been cut or if it has, jumpers reconnect the cut traces). Memory references to addresses with A14 through A19 all set to 1 will have wait states. The duration of the wait is determined by A, B, C. All other memory references run at the maximum speed of the processor. Thus memory references from $00000 to $FC000 run at maximum speed and any reference to the 16K of address space from $FC000 to $FFFFF has the assertion of DTACK delayed by an amount determined by A, B, C. The slow memory space can be increased to 32K by cutting the A14 trace, to 64K by cutting the A14, and A15 traces etc. Whenever PHANTOM is asserted memory references will have wait states, again the number determined by A, B, C.

Our reason for having a slow area of memory is to allow the use of ROM chips which cannot be correctly read by a 68008 running at 8mhz. Also the I/O space is in the slow area of memory because many ICs such as timers, disk controller chips, etc. have slow read/write times.

Memory references to the upper 256 bytes i.e. to $FFF00 to $FFFFF assert SIO which results in the assertion of SINP or SOUT and the negation of SMEMR. Thus references to the I/O space are always slow. If a RAM or ROM board overlaps the I/O space, the overlap becomes write only to those boards that respond to PWR. The signal MWRITE (see Figure 5) is negated by references to the I/O space.

STRT. The assertion of STRT (see Figure 9) indicates the beginning of a read or write operation. You might expect AS to indicate the beginning of a read or write cycle. It does, but there is a problem for the read-modify-write cycle. During such a cycle DTACK must be negated following the read portion of the cycle and reasserted

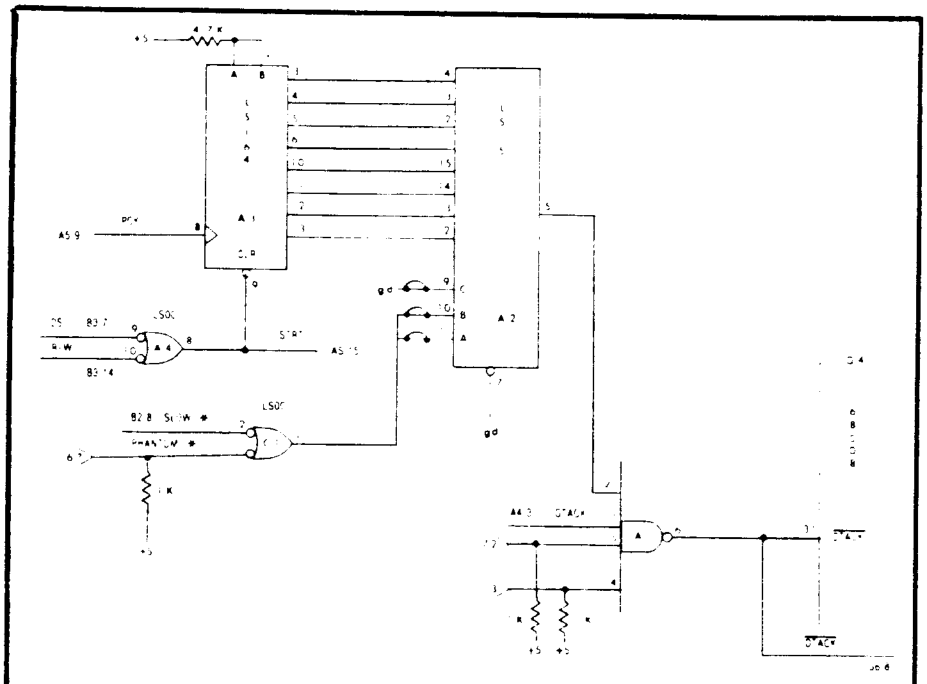


Figure 9

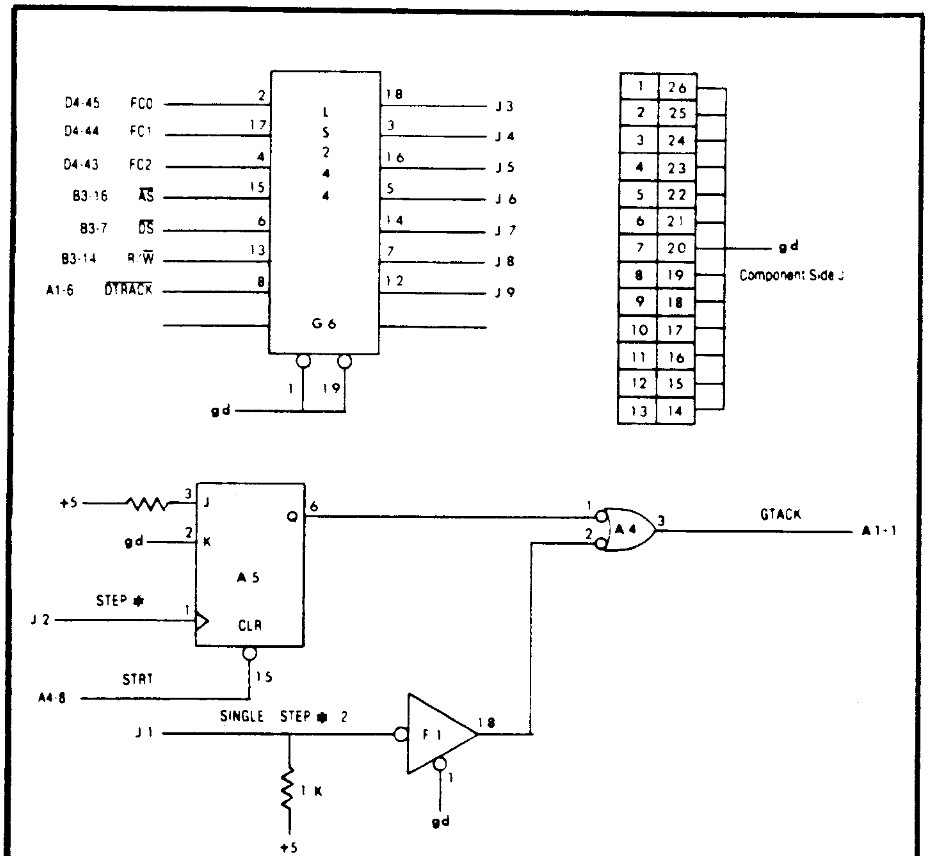


Figure 10

during the write portion of the cycle. If STRT were asserted only when AS was asserted then the read-modify-write cycle would not work properly, because AS is asserted throughout the read-modify-write cycle.

STRT is asserted following the assertion of either DS or W. Note that for a read transaction DS is asserted at the beginning of the transaction (same time as AS is asserted) and for a write transaction W is asserted at the beginning of the transaction (same time as AS is asserted). The only write transaction for which AS and W are not asserted at the same time is during a read-modify-write cycle. Until STRT is asserted the shift register is held in the clear state. Following the assertion of STRT each rising edge of the clock sets successive outputs of the shift register high. After a number of clocks a high will appear on pin 5 of the LS151. The exact number of clocks is determined by the A, B, C values. The inputs at A, B, C yield a binary value and the number of shifts required to set pin 5 of the LS151 high is this binary value plus one.

SINGLE STEP. If J1 is grounded, then SINGLE STEP (see Figure 10) is asserted and pin 2 of A4 is high. As long as as pin 2 of A4 is high GTACK is the inversion of the Q* output of A5. Now a debounced switch connected to J2 can be used to single step the 68008.

At the beginning of every 68008 bus transaction STRT is negated so the Q* output of A5 is high and therefore GTACK is negated. The assertion of either DS or W forces the assertion of STRT which removes the low from the clear input of A5. At this time GTACK is still negated so the 68008 is waiting for the assertion of DTACK (see Figure 9).

Now, if the debounced switch is grounded (i.e., STEP is asserted), the Q* output of A5 goes low so GTACK is asserted. This results in the assertion of DTACK. The 68008 performs the read or write and then negates both DS and W so STRT is again negated. This returns the 68008 to the state described at the beginning of the previous paragraph.

Connector J. The LS244 G6 (see Figure 10) buffers a number of the 68008 control signals so that they may be available to another board. We have also built a companion display board which displays these signals, allows single stepping of the 68008 and displays a number of the S-100 signals. The display board shows the S-100 address, data in, data out in hex displays and control signals in discrete LEDs. ■

Reference 1: "Basic Circuit-design Techniques Yield Stable Clock Oscillators," by Jim Williams EDN Magazine, August 18, 1983.

Note: A PC card for this 68008 CPU design can be purchased for $60 from Intellicomp Inc., 292 Lambourne Ave., Worthington, Ohio 43085.

Interfacing Tips and Troubles

A Column by Neil Bungard

One of the best ways to interface to any computer is through its serial communication port, especially if speed is not a critical factor. I personally use this method anytime I can because the serial communication port is usually a standard configuration regardless of the type of computer being used. In addition, the serial port can usually be accessed through a higher level language (like BASIC) as opposed to writing machine language drive routines. Even in the cases where machine language routines are required, computer manufacturers are good about documenting the required software for using their serial port.

I said that the good thing about the serial communication port is that it is usually a standard configuration. This is good because you know what to expect when connecting to the computer. But it can also be annoying because the logic level voltage standard for a serial communication port and the logic level voltage values in the interface circuit are usually mismatched. Most computers' serial ports adhere to the RS232C standard. This standard assumes a logic 1 to be between − 3 and − 15 volts, and a logic 0 to be between + 3 and + 15 volts. The logic level voltages in the interface circuit (assuming TTL logic) are 0 volts for a logic 0 and + 5 volts for a logic 1.

The discrepancy in logic level voltages between the serial communication port and the interface circuit is eliminated by conditioning the interface signals to agree with the RS232C standards. This typically requires a dual power supply, special driver and receiver ICs, special connectors, etc. After completing several serial interface projects, I decided that I needed a "jelly bean" logic-to-RS232 converter which could be pulled out of my hacker's tool box and placed between any interface circuit and a serial communication port. Figure 1 is such a device. This circuit operates from a single 5 volt supply and conditions TTL logic signals to agree with RS232 standards.

Circuit Description

The logic to RS232 signal conditioning circuit in Figure 1 is extremely straightforward and utilizes an interesting IC manufactured by Intersil Corporation, the ICL7660 (IC1). The ICL7660 is a voltage converter IC which requires only + 5 volts to operate, and produces a − 5 volts on its output (pin 5). This IC eliminates the need for a dual power supply, which greatly simplifies this circuit's design. The other two ICs in Figure 1 are the RS232 driver (LM1488,IC2), and the RS232 receiver (LM1489,IC3). For proper operation, the LM1488 requires + 3 to + 15 volts on pin 14 (which is obtained from the interrface circuit), − 3 to − 15 volts on pin 1 (which is obtained from the ICL7660), and ground on pin 7. The LM1489 only requires + 5 volts on pin 14, and ground on pin 7 for its operation.

Serial information being generated by the builder's circuit enters and signal conditioning circuit on pin 2 of the LM1488 (IC2). The signal is conditioned by this IC to meet RS232 standards, and is output to pin 2 of a standard DS-25 connector (This connector is standard on most computer systems). Information coming from a computer's serial interface port enters the conditioning circuit on pin 1 of the LM1489 (IC3), where it is converted to TTL level signals before it is sent to the builder's circuit.

This signal conditioning circuit can easily be wirewrapped, or for convenience and reliability a circuit board can be manufactured. A foil pattern for the signal conditioning circuit is provided in Figure 2 for those who wish to make a circuit board. If you do not wish to manufacture your own board, I have a few and would be glad to send you one for $10.00. Regardless of whether you produce a board or wirewrap this circuit, I'm sure that you will find this device a worthy addition to your interfacing toolbox. ■

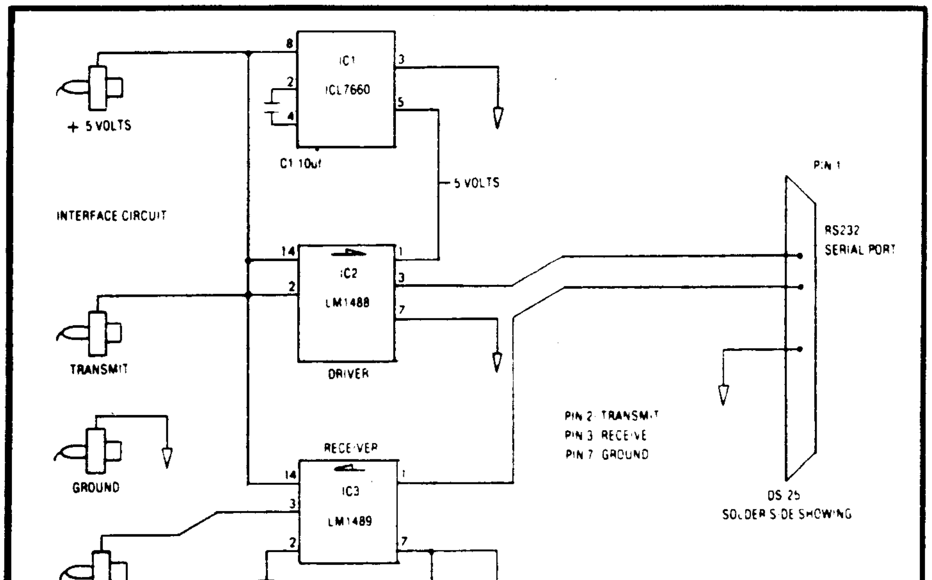


Figure 1

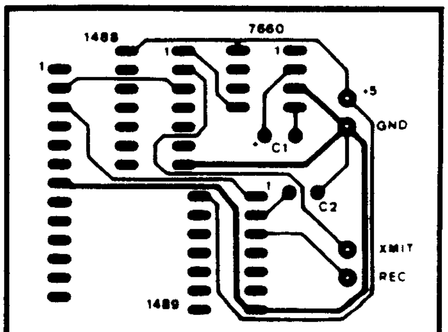


Figure 2

Books of Interest

Structured Microprocessor Programming

by Morris Krieger, Charles Popper, Robert Radcliffe, and David Ripps.
Published by Yourdon Inc.
1133 Avenue of the Americas
New York, NY 10036
230 pages, 6" x 9"

This book is not new (the copyright date is 1979), but it is so unusual that I felt it should be brought to the attention of our readers. Most books on assembly language are either very simple with side-by-side examples of BASIC and assembly language routines, or they are written on a doctorial thesis level and only the experts can understand them. You'd expect a book about structured programming at the CPU level to be difficult to follow, but this book is written for the beginner. On page one they state "We assume no prior knowledge on the part of the reader about programming (structured or otherwise). This book is for the complete novice. It starts at square one."

The book is intended to accompany SMAL/80 (Structured Macro Assembly Language 80) which will be the subject of a separate review, but the detailed programming information will be helpful to anyone programming in assembly language. The contents of the book are as follows:

The book contains a lot of down-to-earth advice. For example, in chapter 2 on structured programming principles, they state "In structured programming, every statement must always have only one entry point and one exit point." They also advise "When a program segment has more than one entry or exit point, the programmer will very likely find he has lost control of his program."

I found chapter 11 on files, counters, and markers, and chapter 12 on storage and retrieval to be especially helpful on a program I was working with using 8080 code and ASM. ■

New Products

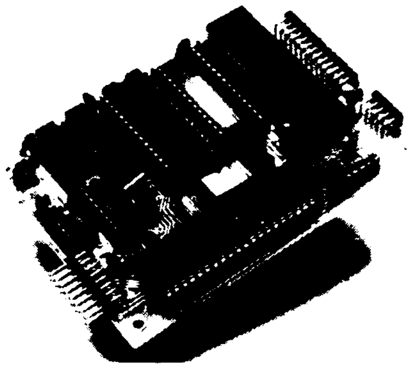
Full BASIC Microcontroller

Basicon's MC-li microcontroller uses the full BASIC language to program directly through a terminal with an RS-232 connection. Designed around Intel's BASIC 52 chip, the self-contained microcontroller offers exceptional programming ease.

Priced at $349, the MC-li uses low power to offer 36 input/output lines, 8K × 8 RAM, 3 timer/counters, 9 interrupts, wide ranging autobaud and a real time clock. An EPROM programmer and other peripherals are also available.

Intended uses include instrumentation, process control, research and development, and even personal computing.

For additional information, write to Basicon, Inc., 11895 N.W. Cornell Road, Portland, OR 97229, or phone 50326012. ■



IBM Compatible SBC

Davidge Corporation announced their DPC-1000 IBM-PC compatible single board computer which they claim matches the IBM-PC form factor right down to the location of the mounting holes.

The DPC-1000 features IBM-PC compatibility on a single board without the need for video and disk controller cards, and they state that it will run all the popular IBM software.

The 5MHz 8088 CPU can be augmented by an optional 8087 math coprocessor. The board, which is available with either 64K or 128K of RAM, can be expanded to 256K with an optional memory expansion module.

There are two 28-pin EPROM sockets, one of which is utilized by the BIOS ROM, and four DMA channels are used for the floppy disk controller, memory refresh, and two user defined functions. Two software controlled timers and five user available interrupts are also provided. I/O includes a monochrome video controller with TTL compatible output, IBM-PC compatible keyboard port, two RS-232 serial ports, Centronics parallel printer port, and a floppy disk controller for up to four 5¼" or 3½" drives. Five IBM-PC compatible I/O slots are also available. A specially written BIOS provieds maximum IBM-PC compatibility and supports PC-DOS, MS-DOS, CP/M-86, Concurrent CP/M and PC/iRMX (Intel) operating systems.

Single quantity evaluation samples are available for a limited time starting at $625, and quantity discounts are provided. This board can be ordered from Davidge Corporation, 292 E. Highway 246, P.O. Box 1869, Buellton, CA 93427 (phone 805/688-9598). ■

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

Interfacing to S-100 (IEEE 696) Microcomputers

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

Microprocessors for Measurement and Control

You'll learn to design mechanical and process equipment using microprocessor-based "real time" computer systems. This book presents plans for prototype systems which allow even those unfamiliar with machine or assembly language to initiate projects. By D.M. Auslander and P. Sagues. 310 pages, 7 3/8x9 1/4, softbound. $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 minicomputers, Tandy Corporation's famous TRS-80, and others). By Murray Sargent III and Richard Shoemaker. 288 pages, 6¼x9¼, softbound. $15.55

Mastering CP/M

Now you can use CP/M to do more than just copy files. For CP/M users or systems programmers – this book takes up where our CP/M handbook leaves off. It will give you an in-depth understanding of the CP/M modules such as, CCP (Console Command Processor), BIOS (Basic Input/Output System), and BDOS (Basic Disk Operating System). Find out how to: incorporate additional peripherals with your system, use console I/O, use the file control block and much more. This book includes a 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, 5¼x8¼, softbound. $20.00

TTL Cookbook

Popular Sams author Don 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

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

Book Total ______

Shipping ______

TOTAL ______