```
2000   3E 0F          LD A,0F        ;load 0F into accumulator
2002   D3 85          OUT (85),A     ;scan 0F displays
2004   06 00          LD B,00        ;load zero into register B
2006   78      LOOP1  LD A,B         ;put contents B into accumulator
2007   D3 84          OUT (84),A     ;output A to segment latch
2009   04             INC B          ;increase value in B by 1
200A   CD 00 21       CALL DELAY     ;go to delay subroutine at 2100
200D   C3 06 20       JP LOOP1       ;unconditional jump to address 2006

2100   11 FF 60 DELAY LD DE,60FF     ;load 60FF into DE registers. Note address 2100.
2103   1B       LOOP2 DEC DE         ;decrement DE by 1
2104   7B             LD A,E         ;load contents of E reg into A
2105   B2             OR D           ;logically OR A with D
2106   C2 03 21       JP NZ,LOOP2    ;if not 0 jump to address 2103
2109   C9             RET            ;when 0 return to main program
```

Table 5. Second Program.

```
2000   3E 46   START  LD A,04        ;define a pattern
2002   D3 84          OUT (84),A     ;& output it to segment latch
2004   0E 05          LD C,05        ;start counter at 5
2006   3E 01          LD A,01        ;load 1 into accumulator
2008   D3 85   LOOP1  OUT (85),A     ;only turn on 1 display
200A   47             LD B,A         ;save contents of A
200B   CD 00 21       CALL DELAY     ;delay routine
200E   78             LD A,B         ;restore A. Delay changed it
200F   CB 07          RLC A          ;rotate register left
2011   0D             DEC C          ;decrease the counter
2012   C2 08 20       JP NZ,LOOP1    ;not zero jump back
2015   0E 05          LD C,05        ;time to go back. reset counter
2017   D3 85   LOOP2  OUT (85),A     ;turn on 1 display
2019   47             LD B,A         ;save A and call
201A   CD 00 21       CALL DELAY     ;delay
201D   78             LD A,B         ;restore A
201E   CB 0F          RRC A          ;rotate right circular
2020   0D             DEC C          ;decrease counter
2021   C2 17 20       JP NZ,LOOP2    ;At right side yet? No, so jump
2024   C3 00 20       JP START       ;Yes. Start again

2100   11 FF AA DELAY LD DE AAFF     ;load 16 bit register pair. Note address 2100
2103   1B       LOOP2 DEC DE         ;decrement DE
2104   7B             LD A,E         ;load E into accumulator
2105   B2             OR D           ;logic OR accumulator with D
2106   C2 03 21       JP NZ LOOP2    ;stay in loop if not zero yet
2109   C9             RET            ;zero. Delay finished.
```

Table 6. Back & Forth.

3. Movement around a Display

The next program to enter is given in Table 8. If you entered the program correctly then one segment should move around three of the displays. Note that we placed the OUT (84),A into the two bytes before the delay subroutine at address 2100. This was done so the delay subroutine would not have to be moved if you already have it at 2100 from the previous programs. The subroutine can be placed anywhere in RAM; there is nothing 'magic' about address 2100.

```
2080   2E 01          LD L,01
2082   2C             INC L
2083   7D             LD A,L
2084   C3 02 20       JP 2002
```

Table 7. Code Modification to Table 6.

Elegant Code. It should be clear to you that this program is not written 'efficiently'. The code works and does the job but the repeated loads and calls are not 'elegant'. The 'better' way is to use a lookup table. (Remember the tune you made yourself in the previous Chapter - the Tune Table is a lookup table.) This trade off between code which works but which is not elegant and code which is tight, elegant and 'good' is something you will have to work out for yourself if you do more programming.

Of course, this trade off is not an excuse not to learn better programming techniques and skills.

```
2000  3E 2A     START   LD A,2A       ;use 3 displays &
2002  D3 85             OUT (85),A    ;light them up
2004  3E 01             LD A,01       ;load segment a into A
2006  CD FE 20          CALL 20FE     ;turn it on then delay
2009  3E 02             LD A,02       ;load segment b into A
200B  CD FE 20          CALL 20FE     ;turn it on then delay
200E  3E 40             LD A,80       ;load segment g into A
2010  CD FE 20          CALL 20FE     ;
2013  3E 10             LD A,10       ;load segment e into A
2015  CD FE 20          CALL 2OFE     ;
2018  3E 08             LD A,08       ;load segment d into A
201A  CD FE 20          CALL 20FE     ;
201D  3E 04             LD A,04       ;load segment c into A
201F  CD FE 20          CALL 20FE     ;
2022  3E 40             LD A,80       ;load segment g into A
2024  CD FE 20          CALL 20FE     ;
2027  3E 20             LD A,40       ;load segment f into A
2029  CD FE 20          CALL 20FE     ;
202C  C3 00 20          JP START      ;and do it all again

20FE  D3 84             OUT (84),A    ;turn on segment in A. Note address 20FE.
2100  11 FF AA  DELAY    LD DE AAFF    ;our standard delay
2103  1B        LOOP2    DEC DE        ;code
2104  7B                 LD A,E
2107  B2                 OR D
2106  C2 03 21           JP NZ LOOP2
2109  C9                 RET
```

Table 8. Light up & Move Display Segments.

4. Hints on Writing your own Programs.

By now you should be starting to modify the programs given here and experimenting on your own. You may have scraps of paper surrounding you with code fragmants on them. It is suggested to you that right from the start you get a book to document all this program development. Second, when you write a bit of code put some comments on it about what it is and what you are trying to do. I am sure you will find that when you try a new program you will always want to be looking back to what you did 20 minutes ago. If it is in a book you will have no trouble finding it. Use a different colour pen when you go back to it so you can tell the changes you made when you return to it.

Use the five fields format to write your programs.

In the last group of programs we will introduce some new techniques and Z80 mnemonics.

5. Looking for a Keypress

The program in Table 9 shows how to look for a keypress then display it using a lookup table. When you think you understand the program try to add display of the four remaining keys - the '+', '-', 'Fn' and 'AD' keys. You will have to look at the 923 Data sheet in the Technical Manual to work out the return codes. You will have to change the number of bits to clear. Then you will have to work out how to physically display the + and A/D keys on the LED displays.

This program introduces the 'db' define byte assembler directive. The actual code in the table is keyed into successive memory locations. The db is for when you use your computer to assemble the code instead of doing it by hand. A public domain Z80 assembler is included on the floppy disk. This program and the remaining ones in this Chapter were developed and tested on an EPROM emulator and the Z8T assembler. We have left in some of the pseudo operation codes (like db)so you gradually become familiar with them. See Appendix II for more details.

6. Introduction to Counting

The program in Table 10 extends the previous program into a one digit counter which displays the count both as a hex digit and as a binary nibble. The '+' key is used to increment the count. Try to write the code to use the '-' key to decrease the count to make an up/down counter. (Look at the data sheet on the 923 chip to find the code returned from the '-' keypress.)

Change the counter to a decimal counter. Can you make a two digit counter counting from 00-99 decimal?

7. More on Keypress Usage

These three programs are similar but there is a lot to learn from the differences between them.

Move 1. Table 11. A shape is put on a display. The 'C' key moves it to the left, the 'F' key moves it right. The shape 'falls off' the display into the unused bit and the speaker. The speaker can be heard to click as the shape moves through it.

Move 2. Table 12. The same as Move 1 but the shape does not fall off the displays.

```
2000  DB 86        waitlp    in a,(86h)          ;get status from input latch
2002  CB 6F                  bit 5,a             ;test status bit (bit 5)
2004  28 FA                  jr z,waitlp         ;if zero then NO KEY pressed
2006  E6 0F                  and 0fh             ;else got key clear bits 4-7
2008  21 17 20               ld hl,dis_table     ;now point HL to display table
200B  85                     add a,l             ;use input nibble as offset
200C  6F                     ld l,a              ;put the result back in HL
200D  7E                     ld a,(hl)           ;get display byte
200E  D3 84                  out (84h),a         ;output to segments
2010  3E 01                  ld a,00000001b      ;turn on right-most display
2012  D3 85                  out (85h),a         ;via the display commons
2014  C3 00 20               jp waitlp           ;all done do it agian
2017  3F 06 5B 4F  dis_table db 3fh, 06, 5bh, 4fh ;0,1,2,3
201B  66 6D 7D 07            db 66h, 6dh, 7dh, 07h ;4,5,6,7
201F  7F 6F 77 7C            db 7fh, 6fh, 77h, 7ch ;8,9,A,B
2023  39 5E 79 71            db 39h, 5eh, 79h, 71h ;C,D,E,F
```

Table 9. Display Hex Key Pressed.

Move 3. The same as Move 2 but the shape will move as long as the keyswitch is held down. We have put Move3.txt on the floppy disk which comes with this Kit. Please print it out and enter it. It appears in several forms. See Appendix II for more details about the different formats.

8. Review

These programs should have started to show you the capability of the Southern Cross. We have introduced more of the Z80 mnemonic codes.

- - - - - - - - - - - - - - - - - - -

```
2000  3E 01                  LD A,1              ;set count to 0
2002  32 00 21               LD (buffer),A       ;and save count
2005  3A 00 21     mainlp     LD a,(buffer)       ;get count on each loop
2008  D3 85                  OUT (85h),A         ;turn on commons
200A  21 2F 20               LD HL,dis_table     ;now convert count
200D  85                     ADD A,L             ;to display code
200E  6F                     LD L,A              ;by using a conversion
200F  7E                     LD A,(HL)           ;table
2010  D3 84                  OUT (84H),A         ;output figure to segments
2012  DB 86        key_loop   IN A,(86H)          ;get key status
2014  CB 6F                  BIT 5,A             ;test it
2016  28 FA                  JR Z,key_loop       ;jump if no key
2018  E6 1F                  AND 1FH             ;clean off unused bits
201A  FE 12                  CP 12H              ;is it "+"
201C  20 F4                  JR NZ,key_loop      ;jump if not "+"
201E  3A 00 21               LD A,(buffer)       ;get our rolling bit
2021  3C                     INC A               ;up count
2022  E6 0F                  AND 0FH             ;but keep it to 4 bits
2024  32 00 21               LD (buffer),A       ;save new count
2027  DB 86        release    IN A,(86H)          ;wait for key release
2029  CB 6F                  BIT 5,A             ;test key status
202B  20 FA                  JR NZ,release       ;jump if key still down
202D  18 D6                  JR mainlp           ;else jump to main loop
202F  3F 06 5B 4F  dis_table db 3fh, 06, 5bh, 4fh ;0,1,2,3
2033  66 6D 7D 07            db 66h, 6dh, 7dh, 07h ;4,5,6,7
2037  7F 6F 77 7C            db 7fh, 6fh, 77h, 7ch ;8,9,A,B
203B  39 5E 79 71            db 39h, 5eh, 79h, 71h ;C,D,E,F
```

Table 10. One Hex Digit Counter.

```
2000  3E 58          ld a,88          ;"your shape"any value 1-FF. We chose 88h
2002  D3 84          out (84h),a      ;output to segments
2004  3E 04          ld a,04          ;this value turns on
2006  D3 85          out (85h),a      ;middle display
2008  32 00 21       ld (buffer),a    ;save rolling bit
200B  DB 86  mainlp: in a,(86h)       ;get key status
200D  CB 6F          bit 5,a          ;test it
200F  28 FA          jr z,mainlp      ;jump if no key
2011  E6 1F          and 1fh          ;clean off unsed bits
2013  FE 0C          cp 0ch           ;is it "C"
2015  20 0A          jr nz,notleft    ;jump if not "C"
2017  3A 00 21       ld a,(buffer)    ;get our rolling bit
201A  CB 07          rlc  a           ;shift it left
201C  32 00 21       ld (buffer),a    ;save it
201F  18 0C          jr out_new       ;jump to output it
2021  FE 0F  notleft: cp 0fh          ;test for "F" key
2023  20 E6          jr nz,mainlp     ;jump to loop if not
2025  3A 00 21       ld a,(buffer)    ;else get rolling bit
2028  CB 0F          rrc a            ;move it to the right
202A  32 00 21       ld (buffer),a    ;save it
202D  3A 00 21 out_new: ld a,(buffer) ;get new rolling bit
2030  D3 85          out (85h),a      ;output it to commons
2032  DB 86  release: in a,(86h)      ;wait for key release
2034  CB 6F          bit 5,a          ;test key status
2036  20 FA          jr nz,release    ;jump if key still down
2038  18 D1          jr mainlp        ;else jump to main loop
```

Table 11. Move1. Shape Falls Through Speaker.

```
2000  3E 58          ld a,58          ; "your shape"any value 1-FF. We chose 58h
2002  D3 84          out (84h),a      ;output to segments
2004  3E 04          ld a,04          ;this value turns on
2006  D3 85          out (85h),a      ;middle display
2008  32 00 21       ld (buffer),a    ;save rolling bit
200B  DB 86  mainlp: in a,(86h)       ;get key status
200D  CB 6F          bit 5,a          ;test it
200F  28 FA          jr z,mainlp      ;jump if no key
2011  E6 1F          and 1fh          ;clean off unsed bits
2013  FE 0C          cp 0ch           ;is it "C"
2015  20 12          jr nz,notleft    ;jump if not "C"
2017  3A 00 21       ld a,(buffer)    ;get our rolling bit
201A  CB 07          rlc  a           ;shift it left
201C  FE 40          cp 40h           ;is it outside left
201E  20 04          jr nz,not40      ;jump if not
2020  CB 07          rlc a            ;else move it to right
2022  CB 07          rlc a            ;display
2024  32 00 21 not40: ld (buffer),a   ;save rolling bit
2027  18 14          jr out_new       ;jump to output it
2029  FE 0F  notleft: cp 0fh          ;test for "F" key
202B  20 DE          jr nz,mainlp     ;jump to loop if not
202D  3A 00 21       ld a,(buffer)    ;else get rolling bit
2030  CB 0F          rrc a            ;move it to the right
2032  FE 80          cp a,80h         ;test for right fall out
2034  20 04          jr nz,not80      ;jump if not
2036  CB 0F          rrc a            ;else move bit to
2038  CB 0F          rrc a            ;left display position
203A  32 00 21 not80: ld (buffer),a   ;save it
203D  3A 00 21 out_new: ld a,(buffer) ;get new rolling bit
2040  D3 85          out (85h),a      ;output it to commons
2042  DB 86  release: in a,(86h)      ;wait for key release
2044  CB 6F          bit 5,a          ;test key status
2046  20 FA          jr nz,release    ;jump if key still down
2048  18 C1          jr mainlp        ;else jump to main loop
```

Table 12. Move2. Shape Stays on Displays.

CHAPTER 6

Z80 THEORY

The Z80 has a very large number of instructions compared to later uP and uC chips. This large number plus the theory of how to use them can be very frightening to the student. That is why we went straight into program the computer in the last two chapters with real examples.

We briefly mention here some of the topics of machine language programming and the Z80 in preparation for more programs and theory of programming we will soon start to discuss. More detailed information can be found in the Technical Manual. A complete listing of the Z80 Instruction Set alphabetically & numerically is given on a separate sheet for your reference during programming.

General Meaning of the More Common Codes.

ADD	Add a byte in a register to byte in another.
AND	Logically AND accumulator contents with another byte.
BIT	Find if bit number x in a byte is a 1 or 0.
CALL	Call a subroutine.
CP	Compare 2 bytes then do something
DEC	Decrement a byte specified by 1.
EX	Exchange the contents of some registers with other registers.
HALT	Z80 stops operation.
IN	Input byte from a Port to the Z80.
INC	Increment a byte specified by 1.
JP	Jump to somewhere.
JP cc	Jump to somewhere after a condition is tested.
JR	Relative jump.
LD	Load somewhere with something.
NOP	Waste time, do nothing.
OR	Logically OR accumulator contents with another byte.
OUT	Output byte from Z80 to a Port.
POP	Store a register pair on the Stack.
PUSH	Push a register pair on the stack.
RET	Return from a subroutine.
RL	Rotate byte left.
RR	Rotate byte right.
RES	Reset bit x of register specified. Make it 0.
SBC	Subtract a register from another.
SET	Set bit x of register specified. Make it 1.
SRL	Byte in register is shifted right one bit.
SUB	Subtract one registers contents from anothers.
XOR	Logically XOR accumulator contents with another byte.

Altogether there are 696 individual instruction in 158 different instruction types. The Technical Manual gives a full listing and some theory of their use. As mentioned before there are many books available which give the detailed theory of microprocessors in general and the Z80 in particular. Please obtain some of these books and look up and read more about the following topics:

- Instruction Types. The Technical Manual gives information on this.
- the Stack. The PUSH and POP instruction uses the stack. The SP, Stack Pointer.
- Flags. Using the flag register is another important part of programming to master.
- Addressing Modes. The same code can usually be done in a variety of ways. You must become familiar with the different adressing techniques available in the Z80 instruction set.
- Ports. The IN and OUT instructions.
- Interrupts and zero page usage. RST instruction.
- Parameter passing.
- Program Counter, PC. We have been using this in our examples without actually mentioning it.
- other Registers, IX, IY

The programs in the next Chapters will start to look at these topics and their usage. We ask that you review these topics for yourself.

Reference Books

Here are some of the books we have used in this project:

1. Data Books from Zilog. Essential for the serious programmer. They have changed their format several times over the last few years. What you want is the Manual containing the data sheets on the Z80 cmos chip Z84C0006 and a separate Technical Manual on the Z80. In the mid-1980s both were published in the one book.

2. Programming the Z80, by Rodney Zaks. Sybex. About the best book to get.

3. Z80 Assembly Language Programming, by Lance Leventhal. Osborne McGraw-Hill. Also good.

CHAPTER 7

THE MONITOR and HOW TO USE IT

The Southern Cross hardware can do nothing on its own, it requires a set of instructions in the form of a program to tell it what to do. The program for the Southern Cross is stored in the 27C64 EPROM and is called a Monitor program. The Monitor's basic function is to allow memory locations to be viewed and changed, and to allow program execution. It also contains a variety of useful routines to aid in the development of programs.

Outline of the Monitor

Please print out the Monitor SCMV1_2.PRN. This is in the format we have used in earlier Chapters to show the original source code, the object code translation and comments. We have given below an outline of the Monitor with some of its major aspects explained. You should follow each of these points then explore the details of the Monitor for yourself to try to understand it. Remember this as about a 'easy' a Monitor to get without sacrificing power or philosophy of Monitor operation.

Making the Link. One of the problems for beginners is that they have worked through and understood basic examples like those presented in the earlier Chapters yet when confronted by the Monitor it looks far too complicated and daunting. (So that when we say it is an 'easy' Monitor this just can depresses the student more.) We do not want this to happen. So what we have done is we have taken a subroutine from the Monitor and worked it through in 3 steps from very simple code which everyone will understand to the same code as used in the Monitor. We ask that you work through the steps and see how each step simply saves code and is quite logical. We have selected the Scan Display (SCAND) subroutine on p11. & 12 of the print-out. The 3 steps are in **multi.z8t** on the floppy disk. You can take the individual code steps out, assemble them and run them to prove that each step works. Or just download the whole file and run the three code fragments at 2100h, 2200h and 2300h. (Serial port downloading is discussed in the next Chapter.) Please print out the total file and study it. Note that to run the second and third steps you must enter data into the addresses 2000h through 2005h otherwise garbage will be displayed.

A similar sequence of gradually more efficient code could be shown for all the other subroutines which compose the Monitor. In many cases there are additional steps which could be taken to make the code even more efficient (or more complicated depending on your point of view!)

More on SCAND. When you run one of the multi code fragments change the speed switch to S)low and vary the potentiometer. Note that at any one time only one display is turned on. This is called multiplexing the displays. At the normal F)ast speed the individual displays are turned on in turn so fast that you get the impression that they are all on simultaneously. Multiplexing displays allows the programmer to save on both the number of components needed and their power rating. The 8x8 add-on board uses multiplexing.

Monitor Outline. The first two pages of the print-out of the Monitor are used to assign variables. EQUate is used to give a specific memory location or binary number a name which can be used in later program instructions. These two pages are very important. They are the basis of the header file which we will discuss below when you write your own programs and wish to use the code already written in the Monitor. We will call these two pages the header file in anticipation of this later discussion.

RAM above 3F00 is used to store global variables defined in the header file. The first two lines assign the start and end of user RAM. RAMEND is used to calculate the CHECKSUM on user RAM done on page 6 of the printout.

The SYSTEM EQUate on the third line is another way of storing variables in RAM. For the Secret Number (p 32) and the Kaleidoscope (p36) those variables used only within them (local variables) are stored temporarily at 3F00 and above. After you leave the subroutine the varables are lost. There is no need to store these variables globally.

The Baud Rate constants are delay times for use in the subroutine MAIN initialized on p6 and used later in the transmit & receive data bit routines. The default baud value of 4800 is assigned to BAUD in the MAIN subroutine on p6. You can change this value by going to BAUD & BAUD+1 and changing the delay times in those two bytes. (Look up the location in the printout and you will find it is 3FC0 in this version of the Monitor. It will probably be a different location in later versions which is why we used the general label BAUD and not the absolute address.)

The I/O Addresses you will already be familiar with.

The Smartwatch variables are for use with the Dallas DS1216 chip. This is fully discussed in the next Chapter.

The Block EQUates are used for Fn 2, 3 and 4.

The last group of EQUates allow access to the power of the Z80 chip set for interrupts and user defined purposes. Only two RST instructions are used. RST 6 is is how you can use the Monitor subroutines for your own programs (see the next section.) RST 7 allows access to routines to help debugging programs (see below.)

The slowest part of the Southern Cross on power up is the switch over of power on the Dallas chip, if present, from the internal battery to the external power supply. The subroutine on p5 tests for when the change has been made.

Initialization of the Southern Cross then takes place. Read through the complete listing and try to pick out bits you understand. Follow the logic where you can. Do not expect to follow all of it in the first few attempts. Keep coming back to it every few days and read more. Enter fragments of it on the keyboard and test out. Read the rest of this documentation with some reference books.

A version number of the Monitor (ASCII 31 is '1', 32 is '2') is assigned on p9. It is not used but may be of use in later versions for testing purposes.

A final question: after you press reset and '2000' appears in the Address displays where in the Monitor are you? The answer is at the end of the Chapter.

The routines which make up the Monitor can be used by the programmer in their own programs. This means that instead of writing code to turn on the displays, read from the keyboard etc. as we demonstrated in earlier Chapters you may use the code already written in the Monitor to do it for you. This is a very important concept to realize. You do not have to keep rewriting the same code fragments in your programs to do routine tasks; you can use the code already available to you in the Monitor.

RST 30H & the System Call Handler

To allow the Monitor routines to be used in your programs there is a single entry point to the Monitor. This is done to ensure that you will not have to rewrite your programs when new versions of the Southern Cross Monitor are issued which change the absolute addresses of the individual subroutines. Each routine available has been assigned a System Call Number. The System Call Jump Table on p5 lists the subroutines available. Also see the listing in the header file. They are:

0	**MAIN**	Restarts Monitor
1	**VERS**	Returns Monitor Version Number
2	**DISADD**	Converts word to 7 segment code in address display buffer
3	**DISBYT**	Converts byte to 7 segment code in data display buffer
4	**CLRBUF**	Clears display buffer
5	**SCAND**	Scans display
6	**CONBYT**	Converts byte to 7 segment code
7	**CONVHI**	Converts hi nibble to 7 seg code
8	**CONVLO**	Converts lo nibble to 7 seg code
9	**SKEYIN**	Scans display until keypress
A	**SKEYRL**	Scans display until released
B	**KEYIN**	Waits for key press
C	**KEYREL**	Waits for key release
D	**MENU**	Compare & jump routine
E	**CHKSUM**	Calculates checksum
F	**MUL16**	16 x 16 bit multiplication
10	**RAND**	Generates random number
11	**INDEXB**	Indexes 8 bit/byte table
12	**INDEXW**	Indexes 16 bit/word table
13	**MUSIC**	Plays music table
14	**TONE**	Plays a tone or note
15	**BEEP**	Beeps the buzzer
16	**SKATE**	Scans 8x8 display
17	**TXDATA**	Send a byte serially on DOUT
18	**RXDATA**	Receive a byte serially on DIN1
19	**ASCHEX**	Convert ASCII character to hex
1A	**WWATCH**	Write to Smartwatch
1B	**RWATCH**	Read from Smartwatch
1C	**ONESEC**	1 second delay using Smartwatch
1D	**RLSTEP**	Relay board sequence test
1E	**DELONE**	1 second delay loop

The subroutines are accessed by first setting up the registers to be passed to it. That is, loading Registers with the values you want the subroutine to use in the calculation. Then the system call number is loaded into the C register immediately prior to executing a RST 30H or RST 6 instruction. The comments in the Monitor at the beginning of each subroutine explains these Entry requirements and also tell you where to find the result after the subroutine has been executed. Go through the Monitor printout and find each of the subroutines which can be called.

This is the general form of programs to use the subroutines in the Monitor:

```
; set up EQUates
call name       EQU    call number
; set up registers
        LD A,nn
        LD HL,nnnn
; system call
        LD C,call number
        RST 30H
```

The system call itself uses 3 bytes, the same as the CALL instruction so there is no penalty in program size. Most subroutines use the A register to pass 8 bit values and the HL register pair to pass 16 bit values. Some also use the DE registers. The RST 6 instruction is the same as RST 30H. (However, z8t will not recognise RST 6. Other assemblers will recognise either RST 6 or RST 30H as being the same. This is a limitation of z8t assembler.) Table 13 is an example of a program to generate and display random hex numbers written entirely using system calls. The simplicity of programs written this way is obvious. You can now see and understand why many of the

```
2000  0E 04     START   LD C,CLRBUF
2002  F7                RST 30H
2003  0E 10             LD C,RAND
2005  F7                RST 30H
2006  0E 02             LD C,DISADD
2008  F7                RST 30H
2009  0E 09             LD C,SKEYIN
200B  F7                RST 30H
200C  0E 15             LD C,BEEP
200E  F7                RST 30H
200F  0E 0A             LD C,SKEYRL
2011  F7                RST 30H
2012  C3 00 20          JP START
```

Table 13. Random Number Program Using System Calls.

programing techniques we developed in earlier Chapters should now be abandoned. You must now start to look on the Monitor as a programming tool to save time and effort. Later you may find that there are subroutines you want to use which are not in our Monitor. (For example, routines to scroll the 8x8 displays, do mathematical calculations.) The solution is to write the subroutines you want and then add them into the EPROM in the unused space above the Monitor. In the next Chapter we have described the EPROM emulator which will allow you to directly test programs in this space. The final step is to write your own Monitor.

The Header File

When you write your own programs and if you want to use the subroutines already written in the Monitor you do not have to look up the EQUates necessary to put in your program (with the possibility that you may leave out one.) All of this information for the complete Monitor is in a header file. All you need to do is include this file with your program and all of that overhead work is taken care of. Print out the 3 pages of **header12.z8t** and compare it to the first 2 pages of the Monitor. Practice on the Random Number program in Table 13. Make a copy **header.z8t** and call it **random.z8t**. Fill out the time, date and description about the program is in the space provided. Then add the mnemonic code from Table 13 between ORG 2000H and END. (Remember the START label must begin in column 1.) Assemble and looking at the .prn file. You can use the serial downloading described in the next Chapter to download the **random.hex** program and run it. Or just enter in the code beginning at 2000H by hand.

RST 38H Using Software Interrupts

The most common problem you will face when learning to write software will be to find out why that piece of code you wrote (which you 'know' is OK) failed to run as expected. Usually the program gets stuck in a loop somewhere. But it may crash which means it has unpredictably destroyed an unknown number of memory locations and maybe even itself. A program which has an error in it is said to have bugs. Hence debugging is the process of removing those errors.

A simple form of debugging is included in the Monitor. An instruction may be inserted into the program code which will allow the registers to be viewed or changed at that point in the logic. This is called a software interrupt. It is done by inserting a RST 38H (ReSTart) instruction into the program. When this instruction is found the current PC contents are pushed onto the stack and substituted by the memory location held in 38H. 38H contains the address of the next opcode to be fetched. At the top of p6 RST38H restarts at 3FFA which has just been loaded in the previous line with the location of the Single Stepper routine SSTEP (056D on p19 of the printout for v1.2 Monitor.) (The other RST instructions 1 through 5 reset the code to the start of the main loop on p6. They are available for use by you for your code.)

All the register contents are displayed on the ADDRESS displays and the name of the register is displayed on the DATA displays in the following order: PC, AF, BC, DE, HL, IX, IY, SP. The '+' key cycles forward and the '-' key cycles backwards. The AD key will return execution to the instruction after the RST 38H instruction. The Fn key will return to the Monitor. You may change any of the registers displayed (be careful in case you get lost and crash the system.) Once the program is working you may substitute a NOP (NO oPeration) instruction for the RST 38H, or squash the program together.

In Table 14 presents a small program to demonstrate the use of the software interrupt.

```
2000  21 AA AA          LD HL,AAAAH   ;load HL with AAAA
2003  06 FF             LD B,FFH      ;load B with FF
2005  2B          LOOP  DEC HL        ;decrement HL by 1
2006  FF                RST 38H       ;software interrupt
2007  10 FC             DJNZ LOOP     ;B also decremented
2009  0E 00             LD C,00H      ;MENU call
200B  F7                RST 30H       ;jump to monitor
```

Table 14. Example of Software Interrupt.

When the program is run the display will show 2007 PC which is the contents of the Program Counter. This is the address the program will start execution at when the AD key is pressed. Use the + & - keys to cycle though the registers. Check that HL contains AAA9. HL has already been decremented once. Step to the B register. It should contain FF. Press AD. B should now contain FE. Enter 3333 on the hex keyboard and press AD. B should decrement to 32. Keep pressing the AD key to see the decrementation process. When you are tired of that enter 0100 on the keyboard, press AD and the program should jump out to the Monitor (RST 6, call 0.)

Using Hardware Interrupts

In the previous section we saw how to use software interrupts; you place a RST 38H instruction in the pro-

gram whenever you wish to examine and change registers. We can use hardware to do the same thing. We can use one IC to generate a hardware interrupt after each instruction is executed . It automatically insert an RST 38H into the code.

The INT pin on the Z80 when pulled low is used to interrupt the program after the current instruction is completed. When this occurs the Program Counter is loaded with 38H and program execution continues from there.

Hardware Single Stepping can be done in two ways. It can be done after every instruction, or it can be done either between two instructions inserted into the code or using the Function 7 toggle.

A 74HC74 IC (dual flip-flop with preset and clear) has been supplied with this Kit. It is suggested that you piggy-back the IC onto IC10 (they are both 14 pin IC's) by (quickly) soldering the power connecrtions (pins 7 & 14) and then running the various jumper connection from it.

Basic Single Step Hardware. Figure 7 shows the hardware required to generate an interrupt after every instruction. M1 (machine cycle 1) is an output from the Z80 that indicates that it is fetching an instruction. The M1 signal is used to clock one of the D flip-flops in the 74HC74.

The Data input is connected to the Chip Select of the EPROM (Pin 20, IC 6) to achieve two things. Firstly, when the code in RAM is being executed, the EPROM Chip Select is High, which means that M1 will clock a High into the latch making Q high and Q bar Low. This generates an INT when the current instruction is complete. Secondly, we need a way to prevent the single stepper from stepping through the monitor, as you cannot use a program that is trying to single step through itself! Each Read from the EPROM clocks a low into the flip flop (makes INT High) and ensures that no interrupts are generated when code in the EPROM is being executed. This occurs straight after an INT has been detected when an instruction is executed at 0038H.

Improved Single Step Hardware. Figure 8 is an enhancement to allow software control of the single stepper. If you want to have single stepping capability on your Southern Cross then this is the modification you should make. IC1:B is configured to provide a change of state each time IO7 is accessed. This output is used to hold the single stepper reset when the it is not required. The connection to the system RESET ensures that the latch always starts out with the stepper disabled.

With this enhancement you can now insert **OUT (IO7),A** into your code to enable and disable the single stepper. This is useful if you are working on a large program and need to trace only a small part of it. You put **(IO7),A** at the beginning and at the end of the particular piece of code you wish to single step through.

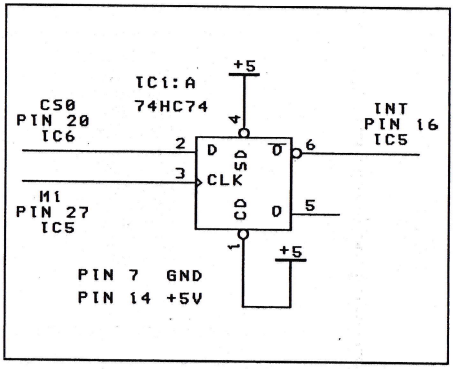

Figure 7. Basic Hardware Single Stepper Circuit.

If the single step hardware is attached and you are in the Monitor then Function 7 will toggle it on/off. A 't' will very briefly appear on the displays when it is turned on/off.

Start to experiment with a simple program like move3. Turn on the single stepper with Fn 7 then run move3 and follow through the various registers as they change. Single stepping either in hardware or software is a very important tool to know how to use when you have to debug your own programs.

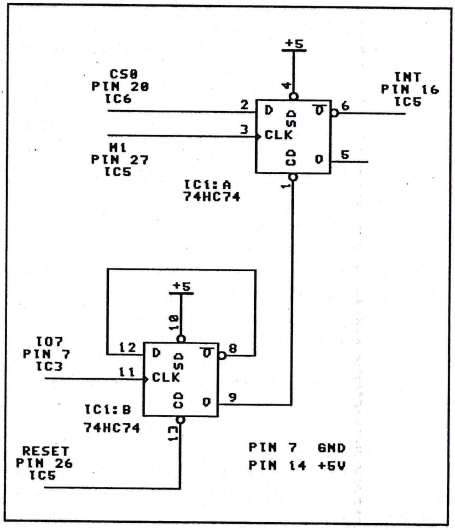

Figure 8. Improved Hardware Single Step Circuit.

Monitor Function Key Assignments

Here is a complete list of Function key assignments made in the Monitor. Go through the Monotor print out to see the code relating to them. This can be found in SCMV1_2.PRN on the floppy disk which came with the Southern Cross.

- **Fn 0.** Start program execution at the address displayed.
- **Fn 1. Ready to Receive Serial Download.** This is explained more fully in the next Chapter. It enables the Southern Cross to receive an Intel hex formatted file from the serial port of a PC.
- **Fn 2. Block Start.** The address displayed is saved for later use. '[' appears briefly in the left data display. It is the beginning block marker.
- **Fn 3. Block End.** The address displayed is saved for later use. ']' appears briefly in the right data display. It is the end block marker
- **Fn 4. Block Move.** Move a block of memory defined by Fn 2 & 3 to the displayed memory position. The move overwrites; it does not insert.
- **Fn 5. Checksum.** All the bytes between Block Start & Block End are added. The low byte of this calculation is called a checksum. It is used to make sure that the bytes in a program block have not changed. 'CS' appears briefly in the Data Display followed by the checksum. The number of bytes in the block appears in the Address displays. 'E' is displayed if the End address is less than the Start.
- **Fn 6. Relative Branch Calculator.** The JR instruction requires a value that is added to the Program Counter to determine where the instruction will branch to. This function calculates the value required and puts it in memory. Step to the address where the offset is to go and press Fn 2. Now step to the address to branch to and press Fn 5. 'RB' is displayed briefly in the Data Displays, then the address of the offset is displayed and the offset value is stored there. 'E' is displayed if the 'branch to' address is out of range.
- **Fn 7. Toggle Hardware Single Stepper** if attached.
- **Fn 8. Play Tune 1.**
- **Fn 9. Play Tune 2.**
- **Fn A. Play Tune in RAM.** A tune you enter is played.
- **Fn B. Toggle Keypress Beep.** The speaker on the Southern Cross can disabled/enabled using this Function. This can be personal preference (you get sick of the noise) or it can be because you want to use the speaker bit on CN4 and you do not want the speaker function. When the speaker is diabled a delay is run in its place to provide a visual cue or feedback that you have actually pressed a key. The delay value can be changed to suite your preference. The address of this delay is KEYTIM & KEYTIM + 1.
- **Fn C. Secret Number.** This is explained in Chapter 4.
- **Fn D. Relay Board Sequencer.** Tests the relays on the Relay Board.
- **Fn E. Kaleidoscope for the 8x8 Display.** Routine to test the 8x8 Display Board works.
- **Fn F. Clock/Calendar Routines for DS1216B.** The optional DS1216B Intelligent socket provides non-volatile RAM backup & date & clock functions. This Monitor function accesses the clock & calendar. See the next Chapter for more explanation.

Answer to Question

We asked the question above about where in the Monitor the Southern Cross after you press reset. Look at the bottom of page 6,

```
MAIN2 CALL SKEYIN  ;WAIT FOR A KEY.
```

The Monitor is waiting for a keypress. It is in a loop continually calling the subroutine **SKEYIN** waiting for a keypress. When it gets one it immediately loads the key table menu at 0294 and the program goes on. Follow through the various possibilities.

CHAPTER 8

NON-VOLATILE RAM

AND THE

PC CONNECTION

You will have discovered that one of the annoying things about entering programs into the Southern Cross is that the program is volatile. When you turn the power off the program disappears from memory. You will also have found that it is long and tedious to hand enter the program using the hex keypad on the Southern Cross, particularly when you want to change the program. There are several ways to overcome these problems. They are:

- use a non-volatile RAM socket.
- connect the Southern Cross to a PC. The program is permanently kept in a PC. It can be downloaded from the serial port of the PC to CN4; from the parallel port of the PC to CN3 or to an EPROM emulator where it can be run, tested and modified.

All of these options are available on the Southern Cross. Our purpose is to introduce & teach the techniques of modern uC/uP code development. We have assumed that a PC - an 8086/80286/386/486 based computer running DOS - is available to you.

Non-volatile RAM.

Dallas Semiconductor make two Intelligent Sockets which are ideally suited for our use. They are the DS1213B & DS1216B. Both fit between the 6264 RAM IC itself and the IC socket. Both contain a battery which keeps power supplied to the data in the RAM when power to the Southern Cross is removed. The battery life is over 10 years. No circuit modifications are required to use the DS1213B. You just plug it in between the RAM chip and the IC socket.

The DS1216B contains an additional feature - a self contained **date & clock** function. It keeps track of hundredths of second, seconds, minutes, hours, days, date, month & year. Accuracy is better than 1 minute per month.

This is an extremely powerful programming tool. Think of the program features which it opens up to you:

- program different messages to come on at different times of the day, or week using our 8x8 Display Board
- program a sequence of relays to open & close at different time of the day or week or month using our Relay Board

Hardware Modification. The DS1216B chip requires two small hardware modifications to be done to the back of the board. These are easy to do. One link has to be cut or desoldered and two jumper wires have to be added.

- Add a jumper wire between pin 1 of the RAM socket IC7 and the RESET pad. This connects the Reset of the DS1216B to the SC Reset. It ensures that data in the DS chip is not corrupted if you press Reset while the data in the DS chip is being updated. The RAM chip has no connection to pin 1.
- 2 pads have been provided at pin 21 of the Z80 and pin 22 of the RAM. Add a jumper wire between these 2 pads. This connects the Z80 Read to the RAM & EPROM Output Enable pins. At the moment these pins are connected to Ground by a link. Follow the track from pin 22 of the RAM & EPROM to the link which connects them both to ground. That is the link to cut. It is also indicated by a dotted line on the overlay of the PCB.

If you cannot easily buy the Dallas Smart Chips then they may be bought from DIY Electronics.

The Monitor has a routine built into it to show the Clock/Calandar. This is accessed by pressing Function F. The time is displayed in 24 hour mode. To change the time press AD. The clock will stop and the displays will become brighter indicating time setting mode is on. Enter the correct time in HH MM SS format. The numbers will scroll in from the right. When you have finished press AD and the clock will restart.

Press the + or - key to display the calendar information in day, month, year format. The day of the week is indicated by the position of the decimal point. Sunday on the left most display, friday on the right. No decimal point indicates saturday. You can change the calendar information by pressing AD and changing the information by keying in the correct numbers. Change the day of the week by pressing the + key. Press AD when you are finished. To return to the monitor press the Fn key.

We have provided for the date to be displayed in either DD MM YY or MM DD YY formats. CALMDE has been provided in the Smartwatch registers. If bit 7 is high the MM DD YY format is displayed. If bit 7 is low then the alternate format is used. Go to the memory location of CALMDE (which is 3FB6 in V1.21 of the Monitor) and write bit 7 as you require.

You can develop your own programs using the Smartwatch using system calls 1A and 1B as described in the previous Chapter.

Connection to a PC.

You will have realized by now that entering in programs by hand and doing hand assembly is time consuming and it is very easy to make an error. We will now discuss using a Personal Computer to automate many of the routine tasks of assembling and entering programs into the target system (the SC in our case) for us. The aim is to use the power of the PC to do all the routine work - program typing, program assembly, downloading, error checking - for us.

There are basically 2 ways to connect the SC to a Personal Computer (PC.)

- connect it to the serial or parallel port of the PC and download the assembled program into the RAM space (2000h to 3FFFh.)
- use an EPROM emulator and move the assembled program into the emulated ROM space. Version 1.2 of the Monitor occupies almost 4K of EPROM so there is 4K of emulated ROM free to use.

We can teach both ways on the SC. Which method you use in any particular sitation will depend on the hardware available to you. For example, in many uC systems there is no RAM available so you must use an emulator. However, in our SC system RAM is available so the first method is more convenient. Either method can download assembled files in the Intel hex format.

1. EPROM Emulator. We have developed a state-of-the-art 'intelligent' EPROM emluator for the Southern Cross. In addition to 8K EPROMs (the size used on the Southern Cross) it will also emulate EPROMs of 16K & 32K. It has an on-board uC. The emulator is now available from DIY Electronics in either Kit or assembled & tested form. The documentation which comes with it discusses how to use it starting from basic principles. We will not discuss it further in this Manual.

2. To the Serial Port of a PC. Most PCs have at least one serial port designed to be compatible with a standard called RS232C. Modems, trackballs and mouses connect to this port. Software like LapLink use it (in its original release) to transfer data from one computer to another. The bit port CN4 of the Southern Cross can be connected to the serial port to transfer data in exactly the same way.

They cannot be connected directly for the simple reason that the PC serial port operates at +/- 12V while the Southern Cross operated at +5V. An interface circuit is required. Two interface circuits are shown in Figs 9 & 10. The 'guaranteed' way to do it is shown in the first Figure 9. A 'less correct' but easier way is shown in the Fig. 10. The latter circuit will probably work in most cases but is not guaranteed to work. In both cases three wires are required between the PC and the interface circuit. Four wires are needed from the interface to the Southern Cross. A packet of components and a PCB to build the transistor interface board has been provided in the Kit.

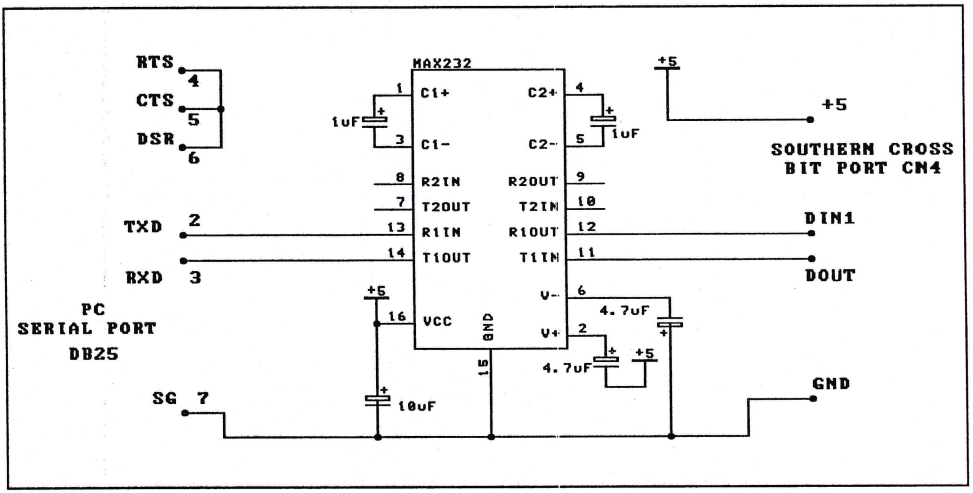

Figure 9. Guaranteed Serial Interface Circuit.

Intel Hex Downloading. Function 1 puts the Southern Cross in 'ready to receive Intel hex formatted files' mode at the address which was displayed when you pressed Fn 1. The Z8T public domain assembler provided with this Kit produces Intel hex formatted output (see Appendix 2 for a review of the assembler, what it does and the files it produces.) When you are ready to transfer a file press Function 1 on the Southern Cross. It is now ready to receive the file.

You can use a communication program on the PC to send out data through the serial port (eg, Telex, Procom Plus) or you can use DOS to do it. Setup the port with the command

MODE COM1: 4800,N,8,1

(Make a batch file setcom.bat for it.) Then use the DOS copy command

COPY filename.hex com1:

If you use Norton then set it up so that when you press Enter on a file with the .hex suffix it will aoutomatically do the DOS copy to com1. If the transfer was successful 'C' will be displayed on the Southern Cross display. Press any key to return to the Monitor. The file should be in RAM at the address (usually 2000h) where you pressed Fn 1. If a checksum error occured an 'E' will be displayed. Press any key to exit the ready to receive state in which case an 'A' will be displayed.

When the file is received by the Southern Cross the Monitor will check it using the chechsum and other information contained in it then convert the actual program to machine code in the correct memory locations.

If it Does Not Work

Did you run setcom.bat to reinitializt the com1 port? Maybe you have to delete any mouse setup line in your autoexec.bat (or config.sys) file and reboot before the setcom.bat file will capture the serial port. The mechanical connection into CN4 may be faulty. Wiggle it during a data transfer to see. Use a logic probe or CRO to check that the signal is getting to the Southern Cross board from the PC. Trace the DIN1 path to IC8, pin 17. Also check that the movement of the 3 wire connection into the PC has not fatigued and broken one of the wires.

You will soon see the benefit of using the PC to develop your programs. We have put many of our example files on the floppy disks in Intel hex format so they can be downloaded.

Baud Rates. The baud rate can be changed. As we saw in the previous Chapter this is set in the monitor by a delay time at BAUD. The delay value for common baud rates are:

300	0220H
1200	0080H
2400	0030H
4800	001BH
9600	000BH

If you change the baud rate remember to reset the com1 port on the PC to the new baud rate.

Remember that BAUD contains the low byte & BAUD + 1 the high byte.

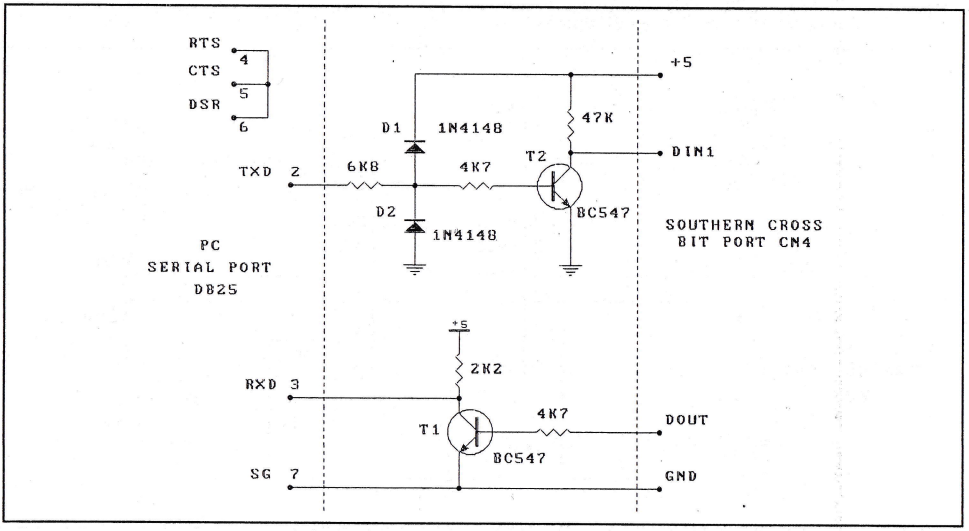

Fig. 10. Probably Sufficient Interface Circuit.

APPENDIX 1

TECHNICAL NOTES

We have tried to learn from the mistakes and deficiencies of earlier SBC's. The main mother board has been kept deliberately simple so as not to intimidate the beginning student with too much on it, yet powerful enough so the student can really progress before they move to the add-on boards using the expansion sockets. The Southern Cross is supplied in kit form so that the student learns to be unafraid of touching the various integrated circuits and components which make a computer.

A big problem with some earlier SBC's was that the Monitor program in EPROM was regarded as 'secret' and 'Copyright' and was generally not published, let alone explained. We have adopted the opposite approach. The annotated Monitor program is supplied on a floppy disk to you. It is a relatively simple Monitor with no program short cuts or tricks.

We have gone straight into running programs (Chapters 4 & 5) and explaining how they work before we have said anything about Z80 codes, the stack, flags, addressing modes, etc (Chapter 9.) We believe that this soft-sell, hands-on approach is better at teaching programming than the traditional approach of doing pages and hours of theory first then being 'allowed' to key in a program.

'Why Didn't you....'. More on our Design Philosophy.

The design philosophy of the Southern Cross was that it had to be simple for beginners to learn from yet powerful enough so that it was infinitely expandable upwards; that is, it could progress with the student to use a PC for code development. It was judged essential that a hex keypad and 6 LED displays be on the board rather than use a PC to plug into right from the start. This way the student could truly feel that they had a real, complete computer in their hands. The PC when it comes to being used with the Southern Cross is a tool to speed up code development and not a 'real' computer somehow attached to the Southern Cross to usurp its role.

We resisted the temptation to add more chips to the main board. Add-on boards using the 16 and 40 pin IDC and the CN4 connections were the way we decided would be most instructive. The board is single sided with links rather than double sided. To a new student having to put on 54 links on a single sided board is less intimidating than a double sided board. Of course by handling the board students loose their fear of it which is our aim. The PCB could have been made smaller but again we thought this would be at the expense of user friendliness. Too many professionals forget the real fears that beginners (and many not so beginners) have about touching IC's and computers.

A hardware speed control has been incorporated together with the usual crystal controlled oscillator. Most programs the student will write will use software delays. However, to show using hardware that the Z80 can be turned down is to teach in a tangible way the benefits of software versus hardware control. (And there are times when a hardware speed control is handy to have immediately available just by moving a switch.)

The above notwithstanding comments about the Southern Cross are invited.

PCB History.

V 1.0	August 1992. First release.
V 1.1	December 1992. Added CN4 connection, increase some pad diameters & minor modifications. Added pads to allow easy DS1216B Dallas Smart socket addition.
V 1.2	April 1993. 22K SIL to replace 5 resistors. Changed to mini-electrolytics. 5V pads for logic probe. Dotted the link to remove for DS1216B. Minor overlay/track/pad changes.

Monitor History.

V 1.0	August 1992.
V 1.1	January 1993. Added support for Relay board, 8x8 Display board, connection to PC serial port from CN4 & routines to use DS1216B Dallas Smartwatch/RAM chip.
V 1.2	February 1993. See V1_21REV.TXT on the floppy disk for details.
V 1.21	July 1993. Fixed wrong byte in Relay testing routine. D4 and D6 were not being tested.

Production Details.

PCB's were designed using Protel® Autotrax. Schematic diagrams were drawn using Protel Schematic. This documentation was produced using Ventura® Publisher V2. Schematic and PCB files were plotted to file using Protel Schplot or Traxplot (HP-GL, 1 pen), renamed .HPG then imported into Ventura Publisher as line art in the normal way.
