Table 9.2. Divisors for even-tempered tones

Note	Frequency	Divisor
D#	156	15798
E	165	14911
F	175	14074
F#	185	13285
G	196	12539
G#	208	11835
A	220	11171
A#	233	10544
B	247	9952
C	262	9394
C#	277	8866
D	294	8369
D#	311	7899
E	330	7456
F	349	7037
F#	370	6642
G	392	6269
G#	415	5918
A	440	5585
A#	466	5272
B	494	4976
C	523	4697
C#	554	4433
D	587	4184
D#	622	3950
E	659	3728
F	698	3519
F#	740	3321
G	784	3135
G#	831	2959
A	880	2793
A#	932	2636
B	988	2488
C	1047	2348
C#	1109	2217
D	1175	2092
D#	1245	1975
E	1319	1864
F	1397	1759
F#	1480	1661
G	1568	1567
G#	1661	1479
A	1760	1396
A#	1865	1318
B	1976	1244

10

The Printer Interface

The Model 100 communicates with a printer according to the Centronics interface standard, which defines mechanical, electrical, and software characteristics of the interface.

Mechanical Requirements

The first requirement of the Centronics standard is the connector, a 36-pin device usually made by AMP or Amphenol. At the rear of the Model 100 is a 26-pin connector labeled "PRINTER", with the hardware designation CN5. This connector, with square pins spaced 1/10 inch apart, was probably chosen to save precious space on the Model 100 case. The printer cable 26-1409, the connections of which are shown in table 10.1, plugs into CN5 and has a connector at the other end that conforms to the Centronics standard.

Table 10.1. Printer connections

Centronics pin	CN5 pin	Function	Source
1	1	STROBE-not	computer
2	3	DATA0	computer
3	5	DATA1	computer
4	7	DATA2	computer
5	9	DATA3	computer
6	11	DATA4	computer
7	13	DATA5	computer
8	15	DATA6	computer
9	17	DATA7	computer
10	19	ignored by Model 100	
11	21	BUSY	printer
12	23	ignored by Model 100	
13	25	BUSY-NOT	printer
14	—	NC	
15	—	NC	
16	—	NC	
17	—	NC	
18	—	NC	
19	2	GND	
20	4	GND	
21	6	GND	
22	8	GND	
23	10	GND	
24	12	GND	
25	14	GND	
26	16	GND	
27	18	GND	
28	20	GND	
29	22	GND	
30	24	GND	
31	26	ignored by Model 100	
32	—	NC	
33	—	NC	
34	—	NC	
35	—	NC	
36	—	NC	

Electrical Requirements

According to the Centronics standard, the thirty-six pins at the Centronics connector fall into two groups: the ground pins (pins 19 and up) and the active pins (pins 1 to 18). Ground pins are connected to a source of zero voltage in both the printer and computer. Active pins carry variable voltages, normally within the range of 0 (LOW or logic 0) to 5 volts (HIGH or logic 1). Most of the active pins are controlled by the computer and are used as a source of information for the printer. A few are controlled by the printer and are used as an information source by the computer.

The Model 100 provides ground to most of the ground pins, and makes most of the active pins accessible to the CPU. Pins 10, 12, 14 through 18, and 31 through 36 are not connected anywhere in the Model 100. For example, many printers announce that they have run out of paper by producing a high signal at pin 12. The printer cable provides this signal to the Model 100 at pin 23, but this signal goes nowhere within the Model 100.

Software Characteristics

The Centronics standard also spells out the sequence in which the printer should energize the various lines to accomplish the printing of a character. Briefly stated, the computer determines whether the printer is able to accept another character by inspecting the BUSY and BUSY-NOT lines. The computer makes an eight-bit word available to the printer on the DATA lines, of which there are eight. It then signals the printer to read the data word by lowering the STROBE line; and leaving the data word in place long enough for the printer to read the data.

The pattern of 1's and 0's which the computer places on the eight data lines is determined largely by the ASCII character set (values 32 to 126 decimal). This is in contrast to the EBCDIC used by IBM and the Baudot code used by many telex machines. Fortunately, almost all character manipulations within the Model 100 use the ASCII set. This allows most texts to be loaded from memory to the printer without the need for translation. Most printers, however, do not respond to values between 128 and 255, let alone reproduce the novel displays the Model 100 screen gives for that range of values.

Model 100 Printer Hardware.

Table 10.2 summarizes the ports through which the CPU accesses the various printer signals. The two printer status lines, BUSY and BUSY-NOT, are made available to the CPU through bits 1 and 2 of input port BB (or B3: decimal values 179 or 187). This port is implemented in hardware through port C of the PIO chip. The two lines discussed here are often referred to as PC1 and PC2. Pull-up resistors are provided, so that without a printer cable attached both appear to the CPU as logic "1".

Table 10.2. CPU access to printer signals.

CPU source (output port)	Printer signal	CPU destination (input port)
E8 bit 1	STROBE	
B9	DATA0 through DATA7	
	BUSY	BB bit 2
	BUSY-NOT	BB bit 1

The CPU sets the eight-bit data word through output port B9 (or B1: decimal values 177 or 185), implemented in hardware as port A of the PIO and often abbreviated as PA0 through PA7. Buffer M32 provides enough power to drive a two-meter cable.

The STROBE-NOT signal, usually at a logic 1 level, must be pulled down for a brief interval. This is done through bit 1 of output port E8 (actually E0 through EF, decimal 224 through 239). Port E8 is selected through port address decode line Y6-NOT and is latched in flip-flop M14. Transistor T8 is the signal driver.

These connections are shown schematically in figure 10.1.

When the printer detects the low condition of the STROBE-NOT signal, it goes to the data lines to see what character is to be printed. Obviously, the computer must leave the data lines unchanged until the printer has done this. Some computers have a par allel buffer dedicated to the printer, so that the problem never arises. In the Model 100, however, output port B9 is used for many other functions, so it is important that none of the other functions take place until the printer has had time to read the data. The designers of the Model 100 could have relied on the ACK-NOT line at Centronics pin 19, to see when the printer has read the data, but not all printers provide the ACK-NOT signal.

Although it might appear sensible to suspend operations until the BUSY and BUSY-NOT lines indicate that the printer has finished printing the character, with a typical 80 character-per-second printer, this would mean waiting 12 milliseconds, which is an eternity to assembly programmers. As we shall see, the ROM resolves the problem by going into a 200-microsecond delay loop (during which time most interrupts are masked) after the strobe is sent. This is based on the assumption that all printers are quick enough to read the data within that time.

How the ROM Prints Characters

The printer driver routine is located at 6D3F through 6D6C. To understand the routine you should disassemble that code.

Since the B and C registers are used, the routine saves those registers on the stack using a PUSH instruction at 6D3F. The character to be printed, which was in A when the routine was called, is loaded to C.

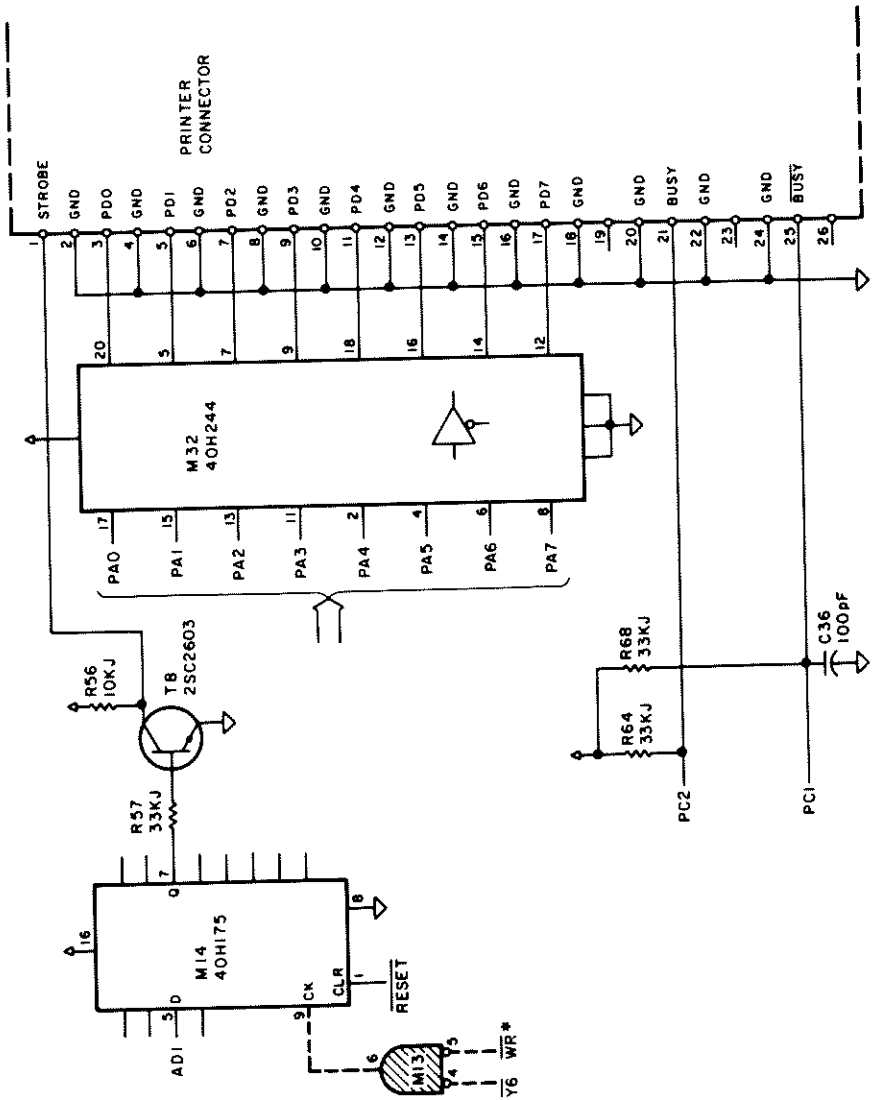

Figure 10.1. Printer hardware configuration

The CPU examines BUSY and BUSY-NOT repeatedly until the printer is ready, each time also calling subroutine 729F, which checks to see if the Break key has been pressed. If it has, the routine returns with the carry flag set. The loop is at 6D41 through 6D4D.

You want to avoid having to reset the computer if the routine is executed when no printer is attached. Model I owners will recall problems along this line.

When the printer is ready, the byte to be sent to the printer is loaded to the printer connector by means of output port B9. Then the printer strobe line is pulled low by turning on bit 1 of output port E8.

Using a single bit of output port E8 can be tricky, since it controls many other functions, such as selection of the option ROM, control of the cassette motor, and strobing of the clock/calendar chip. The Model 100 leaves a copy of the contents of output port E8 at FF45. Bit 1 is turned on and then off again, without changing the other bits as follows:

```
6D56    3A      45 FF   LDA     FF45    ;LD A,(FF45)
6D59    47              MOV     B,A     ;LD B,A
6D5A    F6      02      ORI     02      ;OR 02
6D5C    D3      E8      OUT     E8      ;OUT (E8),A
6D5E    78              MOV     A,B     ;LD A,B
6D5F    D3      E8      OUT     E8      ;OUT (E8),A
```

The port contents are placed in B and the accumulator, and bit 1 is turned on in the accumulator. The accumulator is sent to the port, restored to its previous value from B, and again sent to the port.

Using this ROM code, how long does the strobe last? The changes in output value of port E8 occur near the end of execution of each of the OUT instructions. They are separated by a MOV (LD) instruction that moves data from one register to another. The total time interval is fourteen cycles— the length of one OUT (ten cycles) and one LD B,A (4 cycles).

The Model 100 CPU crystal (hardware designation X2) oscillates at 4.9152 megahertz. The CPU divides this by two resulting in a clock period of about 0.4 microseconds. An interval of fourteen cycles lasts about 5.7 microseconds.

Typical printers (Epson MX-80, Okidata 92) require that the strobe last at least 0.5 microseconds, so this is more than adequate.

The code that follows, at locates 6D61-6D6C, leaves the parallel data in place during a 200 microsecond delay. It unmasks the clock interrupt, restores the previous values in A, B, and C, and returns control to the calling routine.

Fancier print routines.

The printer status bits available at port BB can tell much more than whether the printer is ready to receive another character. For example, with the Okidata 92 printer, the BASIC expression 6 AND INP(187) gives the following values:

6- If printer is not connected
0- If printer is connected, but the printer power is off.
4- If printer is out of paper or off-line.
2- If printer is on-line and ready.

Each model of printer, however, handles conditions such as out of paper and power off differently. In a particular application you may be able to write a program with messages like "Please turn on printer," and so on.

ROM Calls to the Printer

Some ROM printer routines have been published by Radio Shack and are unlikely to change in the event of a ROM update. After calling any of these, the user should test the carry flag upon return to see if the effort to print was successful. If the carry flag is set, it means the printer hung up and the break key was pushed. The routines are discussed in turn.

PRINTR

This is the routine discussed above. The character to be printed is placed in the A register and CALL 6D3F is executed. The BASIC LPOS value, however, is not updated. All register contents including A are preserved.

PRTLCD

This routine dumps a copy of the LCD screen to the printer. The routine is invoked by CALL 1E5E, which is the same as the address BASIC uses when executing the keyword LCOPY.

PRTTAB

This routine, invoked by CALL 4B55, sends a character in the accumulator to the printer, relying upon and updating the LPOS variable at F674. Tabs are expanded in software to spaces.

PNOTAB

This routine, invoked by CALL 1470, sends a character in the accumulator to the printer, relying upon and updating the LPOS variable at F674. Tabs are not expanded to spaces but are sent as they appear. Use this routine if the printer you are sending to has tabs set in hardware at nonstandard spacings, or if the printer itself can expand tabs to spaces.

Printing to Dot-Addressable Graphics Printers

Many printers have escape sequences that allow control of individual pins of a dot-matrix printing head. These sequences allow any of the 256 possible eight-bit words to be sent to the printer. Unfortunately, the printer driver invoked by the BASIC command PRINT expands every 09 hex into one or more spaces, depending on the print column position. This wreaks havoc on the escape sequences. In BASIC, the way to send such a character is CALL 5232, char where 5232 is the decimal equivalent of 1470 hex. The address is the routine PNOTAB, and char is a variable or constant to be sent to the printer.

Unpublished ROM Routines

The RST 4 opcode can be used for printer output if a flag at F675 is set to a nonzero value. This is discussed in detail in chapter 13.

The routine at 4BA0 sends a carriage return to the printer, updating the LPOS variable.

The Low Battery light.

When the Model 100 is turned off and the printer is turned on and connected, some printers cause the Low Battery light to turn on. A "sneak circuit" in the printer interface circuit allows this.

Referring to figure 10.1, note that transistor T8 is turned on whenever a printer strobe is desired by the CPU. This provides a ground path to the strobe line in the printer, which triggers gates in the printer to accept an incoming character.

Within the printer, the strobe line must have a pull-up resistor to plus five volts. If the computer is unplugged, the printer will not start strobing in nonexistent data. The Model 100 also has a pull-up resistor, R56, which is connected to the same 5 volt source as the Low Battery light-emitting diode.

When the Model 100 is turned on and has sufficient power in the battery, transistor T17 is off. It does not turn on T19 so that the LED is kept off. If any current passes through R56, it passes toward the printer.

However, when Model 100 power is off, any weak voltage leaking back to T17, as from the printer through R56, turns T17 on. As a result, T19 turns on, and the weak voltage lights up the LED.

The best way to prevent this phenomenon is by installing a diode in series with R56, so that current can flow only toward the printer.

Not all printers do this— only the ones in which the pull-up resistor is small in resistance.

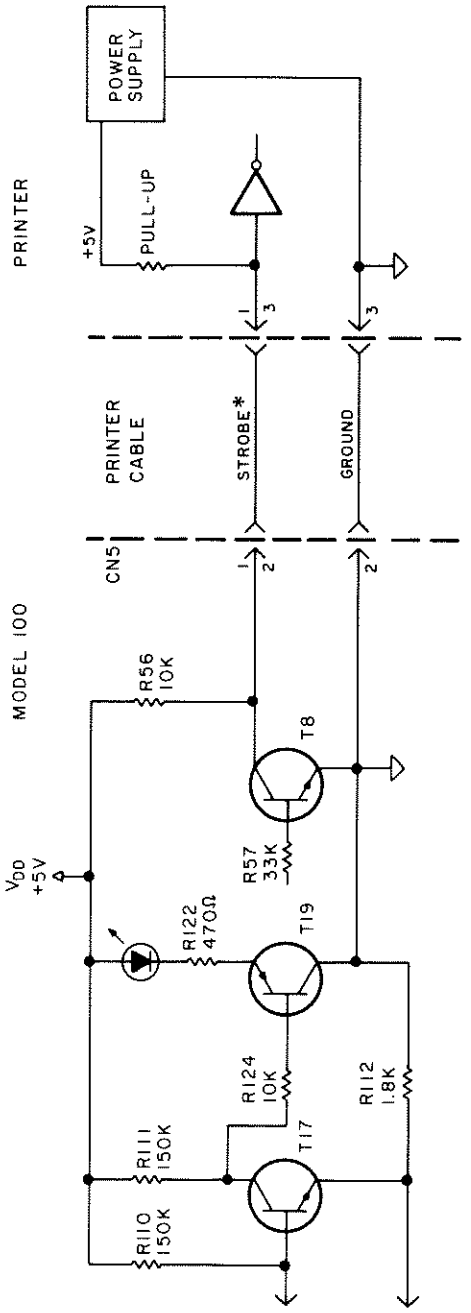

Figure 10.2. Connection between printer and low-battery light

11

Clock/Calendar

The Model 100 keeps time and date information even when it is turned off. It does this through a sophisticated CMOS integrated circuit, the Nippon Electric uPD1990AC. This chapter discusses this chip and the associated circuitry and explains how you can use it, through ROM calls and your own routines.

Terminology

Many devices in the Model 100 have signals and functions that can be referred to by the term *clock*. For example, the CPU provides a 2.4576-megahertz signal known as CLK to the PIO. Any output to port E8 provides four CLK signals to flip-flop M14, and bit 3 of output port B9 provides a signal called CLK to the *clock* chip, M18. We'll refer to the circuitry collectively as the *clock/calendar chip*.

Hardware Theory of Operation

The uPD1990AC integrated circuit, designated M18 in the Model 100, is one of the chips that receives power from the AA cells or AC adapter when the ON/OFF switch SW-5 is off. This power supply, designated VB, is also supplied to the RAM chips and is backed up by the nickel-cadmium cell, so that the clock and RAM information are not lost when the AA cells are changed. The chip draws only a few tens of microamperes when keeping time.

With its crystal X1, which oscillates at 32768 hertz, it is able to keep time and date information current. The designers of the chip chose 32768 because that frequency, divided by two fifteen times, becomes one hertz and is suitable for updating the *seconds* part of its memory. The other crystals in the Model 100 are X2, which provides 4.9152 megahertz to the CPU and X3, which provides 1 megahertz to the modem chip. Neither provides a simple power of two.

The clock/calendar chip is composed of an oscillator, divider, time counter, shift register, and associated control and switching circuitry. Once it is given the values, it maintains seconds, minutes, hours, day of the week, day of the month, and month in the time counter (shown in figure 11.1). Long and short months are properly accounted for, with the exception of February 29th in a leap year.

The year is maintained not by the chip but by the ROM operating system. You may already be familar with one bug. Occasionally the year will be incremented when it should not be; this is not the fault of the chip.

The time and date information in the time counter comprise forty bits of data. From time to time, this data must be loaded to and from the CPU. To reduce the physical size of the chip, its designers chose to use serial data transmission, which requires fewer pins than parallel transmission. The chip includes a forty-bit shift register, used for loading data into and out of the chip. Commands to the chip allow it to load serial data from the CPU into the shift register, from the shift register into the time counter, from the time counter into the shift register, or from the shift register serially to the CPU.

In addition, the chip can also programmed to provide timing pulses (TP) of 64, 256, or 2048 hertz to the CPU interrupt RST 7.5.

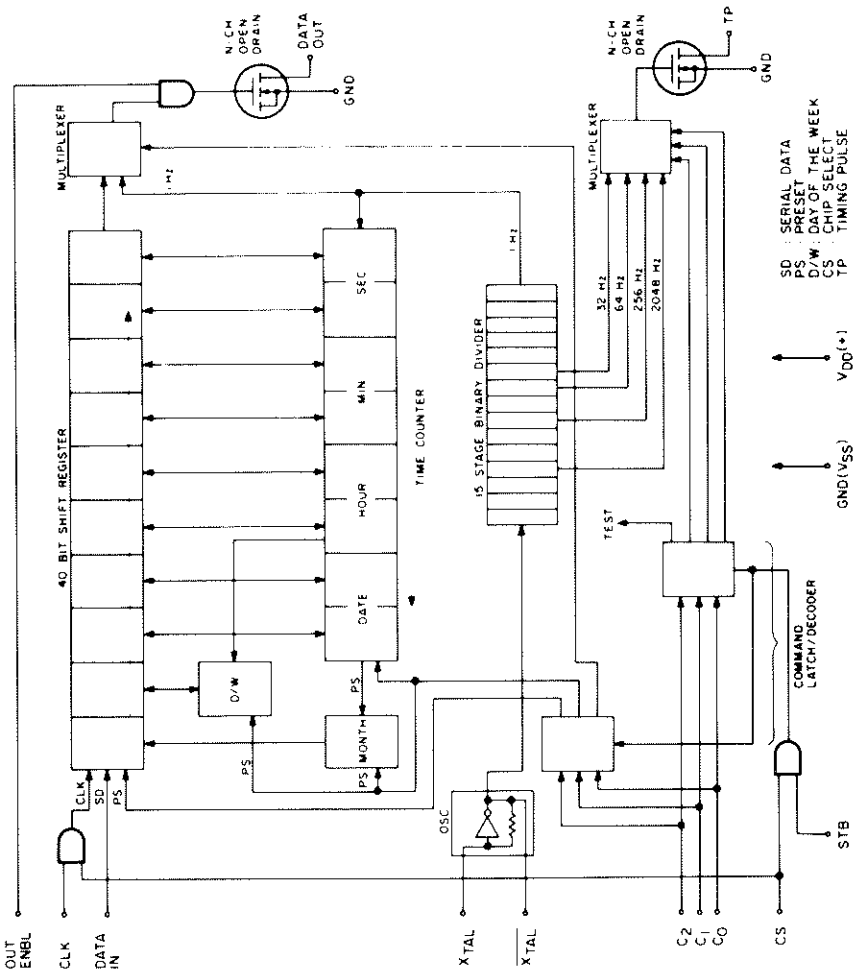

Figure 11.1. Clock/calendar architecture

The commands which may be sent to the clock/calendar chip are summarized in table 11.1.

Table 11.1. Clock/calendar set/read commands. The command value is placed in output port B9, then the clock is strobed by momentarily turning on bit 2 of output port E8 (or by using CALL 7383).

Command Value (hex)	Function
00	Register hold (normal timekeeping)
01	Commences shift register loading (both in and out)
02	Load shift register into time counter (set time)
03	Load time counter into shift register (read time)

The digits of time and date information are loaded into or out of the chip, as shown in table 11.2. The chip uses BCD (Binary Coded Decimal) format, in which 0000 means zero, 0001 means one, and so on, up to 1001 which means nine. Obviously BCD has much in common with hexadecimal notation; the major difference is that values like 1010 have no meaning to a machine using BCD.

Table 11.2. Digit sequence for chip loading

Bits	Meaning	Format
0-3	Seconds units	BCD
4-7	Seconds tens	BCD
8-11	Minutes units	BCD
12-15	Minutes tens	BCD
16-19	Hours units	BCD
20-23	Hours tens	BCD
24-27	Date units	BCD
28-31	Date tens	BCD
32-35	Day of week	0=Sunday, 6=Saturday
36-39	Month	1=January, 0CH=December

Setting the Time in the Clock/Calendar

To set the time, the shift register must be loaded with the desired time/date information. The register is loaded serially; the serial load mode is commanded by loading output port B9 with the *shift* command 01 and then strobing the clock/calendar, as shown in figure 11.3a.

Strobing the Clock/Calendar

Strobing is performed by turning bit 4 of output port E8 on and then off. Because the ROM operating system stores the contents of output port E8 in RAM at FF45, the proper way to strobe a clock/calendar command is as follows:

```
OUT B9     ; OUT (B9),A  command to port B9
LDA FF45   ; LD A,(FF45) get contents
ORI 04     ; OR 04       turn on bit 2
OUT E8     ; OUT (E8),A  strobe the chip
ANI FB     ; AND F8      turn off bit 2
OUT E8     ; OUT (E8),A  finish the strobe
RET        ; RET         wasn't that easy?
```

This routine is available at 7383 through 7390 in ROM and can be invoked by a CALL to 7383.

Output port B9, however, is used for many functions other than sending commands to the clock/calendar. These functions include LCD, LPT, and keyboard control. If interrupts are enabled, it is possible for the contents of output port B9 to change between the load in the beginning of the subroutine and the strobe at the fourth line. Thus, it is necessary to disable interrupts during the routine; all the ROM routines which use CALL 7383 have interrupts disabled at the time of the call.

Once the shift mode is selected, the clock/calendar chip pays close attention to bits 3 and 4 of output port B9. The contents of bit 4 (either 1 or 0) will be understood later by the chip to be bit 0 of the seconds (as shown in table 11.1).

Bit 3 of the port is then turned on and bit 4 is left in its previous state. Turning on bit 3 provides a so-called CLK input to the chip, which causes the contents of bit 4 to be loaded into the shift register. Bit 3 is then turned off. This is shown in figure 11.2.

The process is repeated for the remaining thirty-nine bits; after receiving the forty CLK inputs, the chip is no longer in shift mode. This is shown in figure 11.2.

Interrupts must be disabled during the entire process, so that the only changes to output port B9 are those selected by the main program (and not by an interrupt routine).

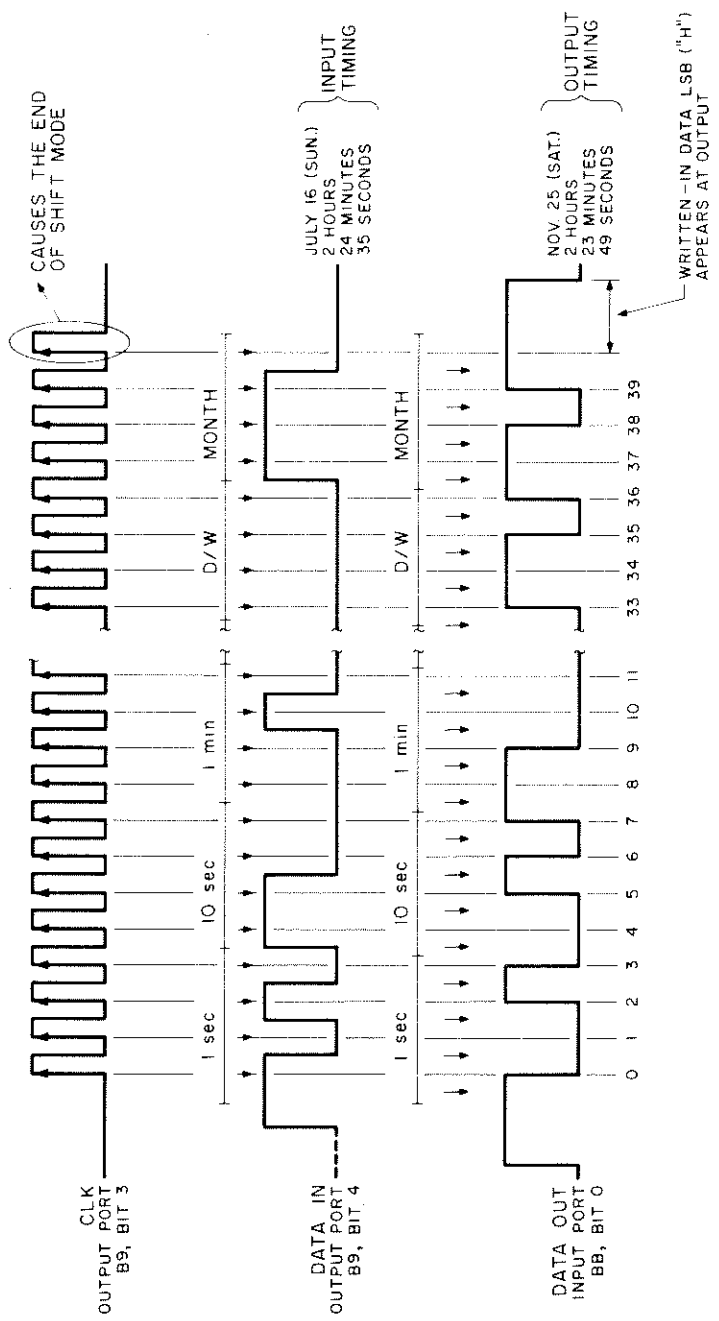

Figure 11.2. Clock calendar strobing

After the forty bits of new time/date information have been loaded to the shift register, the chip must be commanded to load the contents of the shift register into the time counter, as shown in figure 11.3b. This is accomplished by sending the 02 command, as listed in table 11.1. Finally, the chip is allowed to go back to normal timekeeping, by command 00, the register hold command.

Nothing in the hardware prevents the loading of meaningless values, such as 1111, into the shift register of the chip. Doing so would cause the chip to yield funny time values until the digit had been incremented back to zero.

Reading the Time

The process of reading the time is similar to setting the time. First, a read time command, 03, is strobed into the chip. This causes the time/date information to be loaded into the shift register, as shown in figure 11.4a. Then a shift mode command, 01, is strobed into the chip. The chip provides bit 0 of the shift register (the bottom bit of the seconds unit) to bit 0 of input port BB. As before, bit 3 of output port B9 gives a CLK signal to the chip. Turning bit 3 on and off again makes the shift register advance to the next bit. This is shown schematically in figure 11.4b.

After thirty-nine such loads and CLK signals (*see* the bottom plot in figure 11.2), the chip leaves the shift mode. The chip can be returned to normal timekeeping by issuing a "&0" command.

Selecting a TP Frequency

The Model 100 ROM relies on a 256-hertz TP signal. This is provided by the clock/calendar to the CPU at its RST 7.5 interrupt input. The arrival of the TP signal at the CPU causes it to execute an interrupt routine if the RST 7.5 interrupt is enabled and unmasked. The interrupt routine performs a number of functions, including keyboard scanning (*see* chapter 15.) By placing a vector in RAM at F5FF, the interrupt can be utilized for other purposes.

In such an event, you may wish to execute the TP interrupt more or less frequently than 256 hertz. This can be accomplished by strobing a value other than 05 to the clock/calendar, as shown in table 11.3.

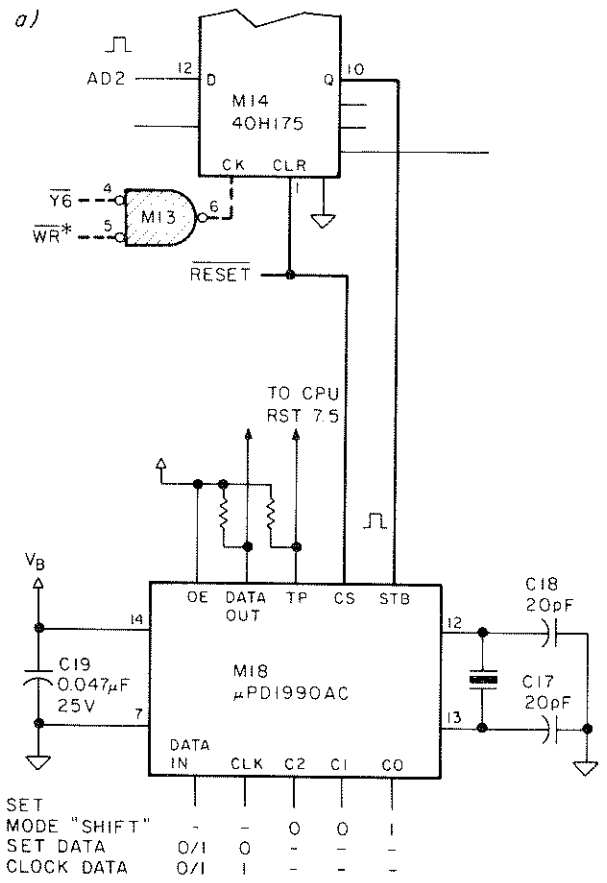

Figure 11.3a. Setting the clock calendar

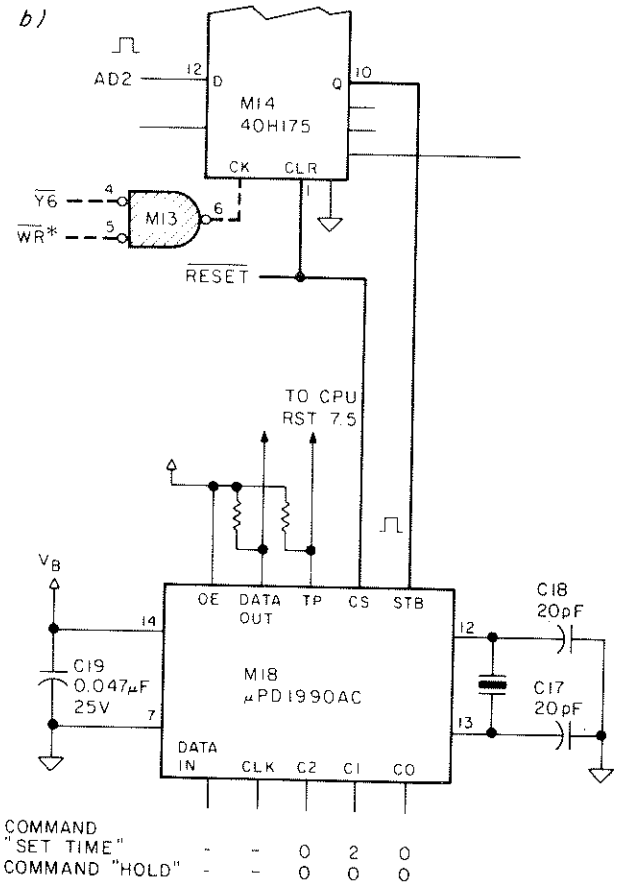

Figure 11.3b. Setting the clock calendar

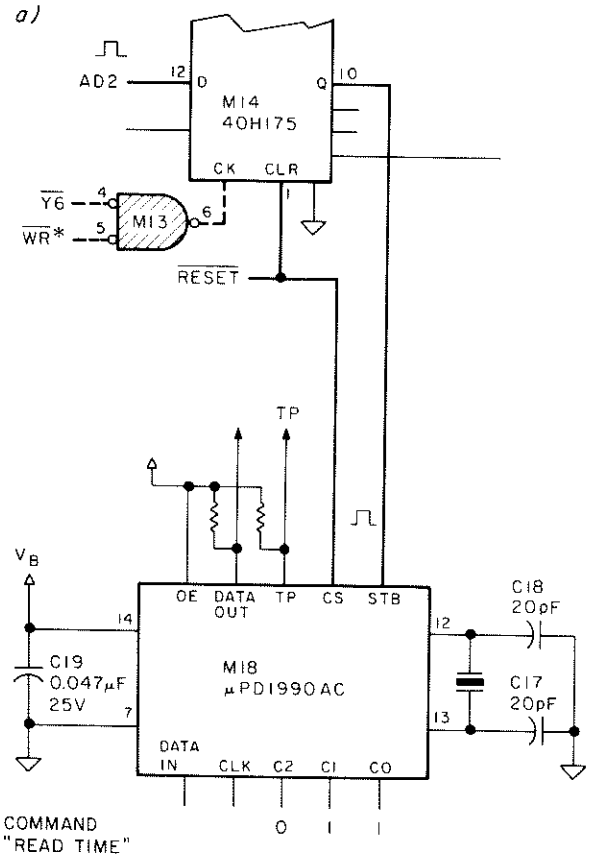

Figure 11.4a. Reading the clock calendar

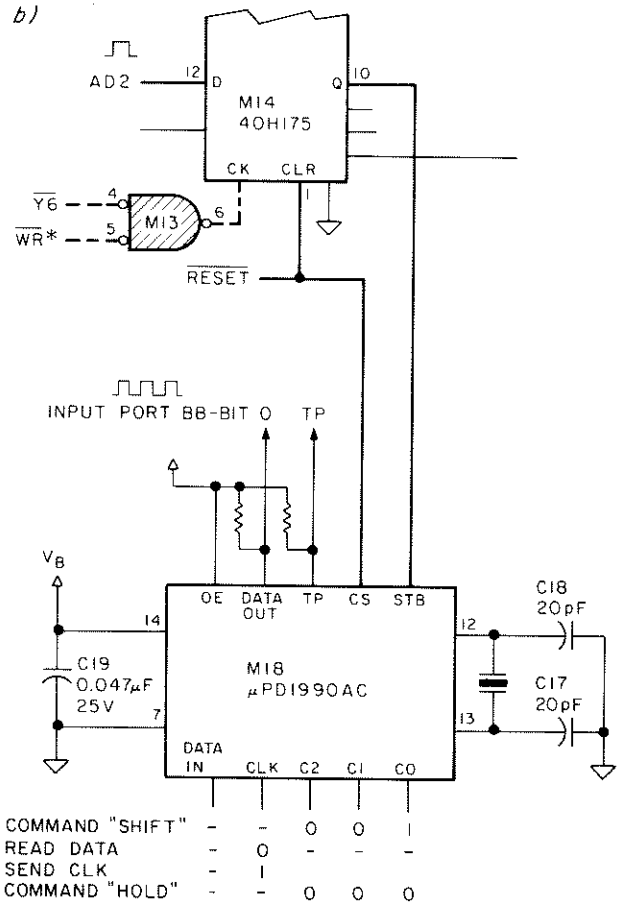

Figure 11.4b. Reading the clock calendar

Table 11.3. Clock/calendar TP commands.
The command value is placed in output port B9 and the clock is strobed by momentarily turning on bit 2 of output port E8 (or by using CALL 7383).

Command Value (hex)	Interrupt Frequency Selected
04	64 hertz
05	256 hertz
06	2048 hertz

The TP frequency is selected in ROM during the power-on sequence, at 6CEB.

Clock/Calendar Accuracy

The clock/calendar keeps time as accurately as a quartz watch. The temperature of the crystal affects timekeeping; extremes of temperature throw off the time noticeably.

Most CPU functions, including enabling and disabling of interrupts, do not affect the clock. This will be a surprise to Model I/III owners, as with these computers the clock accuracy is affected by the amount of disk usage.

In particular, the power-up routines do not affect the chip which continues to keep time whether power is on or off. It is reset only by the BASIC TIME$=, DATE$=, and DAY$= assignments, or by a reinitialization of RAM.

Published ROM Routines

Three routines have been published which allow you to determine the time and date. Prior to calling any of the routines, buffer space should be set aside for the answer, and HL should be set to the starting address of the buffer to which the information will be returned. The routines are listed in table 11.4.

Table 11.4. Routines to obtain time and date

Data Desired	Routine Location	Format	Number of Bytes
Time	190F	hh:mm:ss	8
Date	192F	mm/dd/yy	8
Weekday	1962	ddd	3

Unpublished Routines

A portion of the date or time information can be determined by accessing the location in RAM where the information is stored. The addresses are given in table 11.5. Before using these values, RAM should be refreshed by calling the ROM routine at 19A0.

Table 11.5. Date and Time locations in RAM after call to 19A0.
Values are BCD unless otherwise noted.

Address	Contents
F923	Seconds units
F924	Seconds tens
F925	Minutes units
F926	Minutes tens
F927	Hours units
F928	Hours tens
F929	Date units
F92A	Date tens
F92B	Day of week (0=Sunday, 6=Saturday)
F92C	Month (0=January, C=December)
F92D	Year units
F92E	Year tens

Note that table 11.5 resembles table 11.2. That is due to the fact that the routine called at 19A0 loads the clock/calendar shift register directly into RAM starting at F923.

Note, however, that the buffer starting at F923 is much larger than forty bits. In particular, each group of four bits from the shift register ends up in a separate eight-bit byte in RAM.

Setting the Time through ROM Calls

The actual serial loading, both to and from the clock/calendar chip, is performed by the routine at 7329 through 7390. The routine has two entry points: 7329 to read time and 732A to set time. Prior to the call, HL must point to a ten-byte RAM area, in the format shown in table 11.6.

If the time-set entry point is used, the contents of the buffer are loaded to the clock/calendar. Only the bottom four bits of each byte are loaded to the chip, and no checks are undertaken for correctness of format. For example, 00001111 could be loaded and it would not make sense to the chip, since 1111 is not a correct BCD value.

If the time-read entry point is used, the contents of the clock/calendar are loaded to the RAM buffer. The top four bits of each byte are zero.

Table 11.6. Buffer for routine at 7329 and 732A

Values are BCD unless otherwise noted.

Address	Contents
HL+00	Seconds units
HL+01	Seconds tens
HL+02	Minutes units
HL+03	Minutes tens
HL+04	Hours units
HL+05	Hours tens
HL+06	Date units
HL+07	Date tens
HL+08	Day of week (0=Sunday, 6=Saturday)
HL+09	Month (0=January, C=December)

Rotations of the accumulator are used to select bits to be loaded to and from the serial input and output of the chip. The code is fascinating and well worth disassembly and study for those who wish to understand how a CPU can undertake serial input and output without a UART.

Comments are provided in figure 11.5.

```
7329      entry point for update of RAM
732A      entry point for update of chip
732C      disable interrupts
7331      if in RAM update mode, strobe a time read command to chip
7336      strobe a shift mode command
7339      delay forty microseconds
733E      ten digits are loaded
7340      each digit is four bits
7344      RAM update mode?
7348-D    if so, get bit 0 of input port BB
7352-B    chip update mode? if so, send a bit to port B9, bit 4
735D-63   send a CLK pulse to chip
7376      chip-set mode?
7377-9    if so, set time
737C-D    let chip return to normal
7380      reenable interrupts, return
7383-90   routine to strobe the chip with command in accumulator
```

Figure 11.5. Comments for serial clock/calendar routine

12

Cassette Input and Output

The cassette interface of the Model 100 is used for storing and loading data, BASIC programs, and machine-language files.

Three file formats are used, and they correspond to the three types of RAM files: DO, BA, and CO. Data files are written on tape in 256-character blocks, each with checksum. Basic and machine language files are written in a single large (or small) block, also with checksum. A machine-language file can contain addresses for loading location and for the entry point at which execution is to begin.

Accessing Data (DO) Tape Files from BASIC

Cassette data (DO) files may be easily accessed from BASIC, using the OPEN and CLOSE commands and INPUT or PRINT statements. The ROM operating system provides RAM buffers for cassette input and output, both of which are so well integrated with BASIC that one need pay no attention to the details of the tape storage format.

Creating a CO Cassette File

The CSAVEM command or (SAVEM command with CAS: device specification) can be used if data is stored in memory which is to be stored on tape and later reloaded in the same memory location. The syntax is:

```
CSAVEM "filnam", stadd, endadd, tradd
```

where filnam is a filename of one to six characters, and stadd and endadd are the boundaries of the RUNM command. If no tradd value is given at the time of the CSAVEM then stadd will be used.

Loading a CO File back into RAM.

When the tape file is reloaded to RAM with the CLOADM (or LOADM) command, stadd and endadd, previously written to tape with the data, are used to determine where in memory the data will be loaded.

There is one other way to access a CO file on tape. The BASIC command RUNM will load the data on the tape into RAM according to the stadd and endadd stored on tape (just like CLOADM). BASIC then commences execution of the program by JUMPing to the address stored on tape as tradd.

Often one wishes to load a tape CO file into RAM at addresses other than those stored on tape for stadd and endadd. The BASIC OPEN command is no help, because it cannot be used to access a CO tape file. The ROM routines for cassette I/O happen too quickly for use in BASIC through CALL commands, so one is pretty much limited to assembly language for I/O involving machine-language cassette files.

Accessing BA Tape Files From BASIC

You really can't do it. If you try to execute a CSAVE from within a BASIC program (as distinguished from a CSAVE executed in immediate mode), you will find that BASIC returns to the "OK" prompt when the CSAVE is executed.

This happens regardless of whether the program containing the CSAVE command has finished.

A similar problem occurs when CSAVE is used in the immediate mode. For example, if you type:

```
CSAVE"A":CSAVE"A"
```

you will find the program is saved only once.

The CSAVE command can only be used to save the BASIC program presently in the BASIC work area and cannot be used to save some other BA file in RAM memory.

The CLOAD, LOAD"CAS:" and RUN"CAS:" commands can be used within BASIC. This causes the program being run to be destroyed and the new program loaded in its place. The RUN command does result in the execution of the new program and can be used for the chaining of programs.

Hardware Theory of Cassette Operation

The 8085 CPU is specially designed for CPU-controlled serial input and output. SID and SOD are dedicated chip pins used for incoming and outgoing serial data, respectively. In the Model 100 these pins are connected to the cassette interface circuitry.

Throughout the service manual the cassette circuitry bears the cryptic legend "CMT". Here the term "cassette" will be used.

Incoming Cassette Data Flow

The cassette cord 26-1207 provides a connection to pin 4 (RXC) of the CASSETTE connector (CN3), from the earphone jack of the cassette recorder through the black plug, as shown in figure 12.1. At CN3 the signal connects with the circuitry shown in figure 12.2. At first glance it appears that diodes D5 and D6 prevent signals from passing through to the operational amplifier (op amp) M30. This is because D6 will conduct, providing a short circuit if the incoming signal is positive. If the signal is negative, diode D5 will conduct.

A diode like D5 or D6, however has the interesting property that it will never provide a short circuit. Instead, the amount of current flowing through it will be limited in such a way that its presence in the circuit never pulls the voltage across it to less than about 600 millivolts. The forward-biased voltage drop is 600 millivolts.

The diodes act as a limiter or "clamp" on the amplitude of the signal that reaches the op amp.

As a result, great variations in the strength of the input signal will not have much of an effect on the ability of the computer to read the data. Any signal strong enough to saturate the diodes (more than 800 millivolts) will do just fine. Increasing the volume further will not harm things either.

The input register R95 determines the input impedence, which is 100 ohms.

The audio signal, converted to a 600 millivolt square wave by the diodes, is amplified by op amp M30 to a square wave of about five volts magnitude. The op amp inverts the signal as the clamped audio signal is fed to the inverting input of the amplifier.

This signal is inverted (and neatly trimmed to be a "clean" square wave) by invertor M34, and is made available to the CPU at pin 5 (SID).

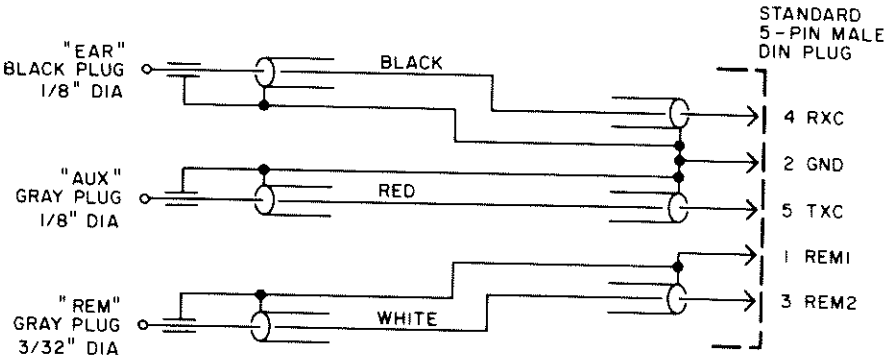

Figure 12.1. Schematic of cassette cable 26-1207

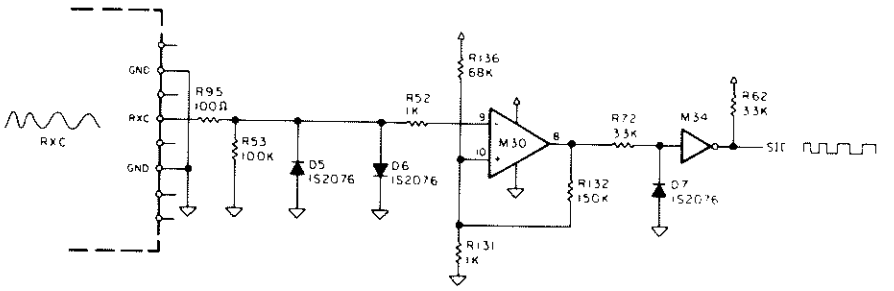

Figure 12.2. Incoming cassette data

Reading the Data at the SID Terminal

The CPU reads the contents of the SID pin using the RIM (read interrupt mask) instruction. The RIM instruction and its counterpart SIM, are the two instructions added to the 8080 instruction set by the designers of the 8085.

The RIM instruction loads the Interrupt Mask to the accumulator. For cassette purposes seven of the eight bits loaded are useless, only bit 7 is meaningful. Bit 7 is a "1" if the SID pin is receiving 5 volts, or a "0" if the SID pin is receiving zero volts.

In other words, positive and negative portions of waveforms from the cassette correspond to 1's and 0's, respectively, at bit 7 of the accumulator after the RIM.

Several programming methods may be used to handle the value at bit 7. These include an AND with 80 hex or a rotate left instruction. The latter requires fewer bytes to store in ROM and is used in the Model 100 in its cassette bit input routine at 6FDB-7015.

The routine at 6FDB watches the SID pin carefully to see how much time passes between the start and midpoint of one square wave, with the result returned in the C register.

When the start of a waveform is noted, a tight loop examines the SID pin repeatedly, incrementing C until the midpoint of the waveform is seen. The loop is 29 CPU cycles long, or about 12 microseconds.

If the incoming waveform is 1200 Hertz, the time from start to midpoint would be about 416 microseconds. One would expect C to reach a value of about 35. However, if the incoming waveform is 2400 Hertz, one would expect the value of C to reach a value of about 17.

The bit-input routine has calls to 729F in each tight loop, so that pressing the SHIFT-BREAK key will allow a graceful exit from the call. Each ROM routine that calls the bit-input routine has a RC (RET C) instruction after the call for the same reason.

The bit-input routine also contains instructions that call the beeper-toggling routine at 7676 if sound is enabled according to the flag at FF44. This is how the audio monitor of cassette loading is accomplished.

OUTGOING CASSETTE DATA FLOW

The CPU's Role

The CPU causes cassette output by wiggling the SOD line. In ROM this is performed by the bit-output routine at 6F6A-6F84. The routine produces a single square waveform (at the SOD pin) of either 2400 Hertz (if bit 0 is one) or 1200 Hertz (if bit 0 is 0), based on the value at bit 0 of the accumulator.

The timing values in the D and E registers, set at 6F6B or 6F71, determine the time delay between the upward and downward transitions of the waveform, respectively.

The transitions in the SOD pin are accomplished by the SIM instruction. Recall from the discussion in chapter 2 that the SIM instruction leads a double life. It sets the SOD value and masks interrupts depending on the condition of bits 6 and 3 of the accumulator prior to executing the SIM instruction. For control of the SOD pin, bit 6 should be on and bit 3 should be off. Bit 7 is loaded with the desired SOD status, either a 1 or a 0. The SIM instruction is then executed.

In simple terms, to turn SOD on, load D0 hex to A, then execute SIM. To turn SOD off, load 50 to A, then execute SIM.

The bit-output routine performs a left-rotate. Repeated calling of the routine sends successive bits of the value in the accumulator to the cassette. This can be seen in the routine DATAW at 6F5B-6F67, which writes to tape the character in the A register. It simply sends one 1200-Hertz cycle, then calls the bit-output routine eight times, sending eight square waves, each of which may be 1200 Hertz or 2400 Hertz.

Hardware Treatment of the SOD Signal

If you try to record a square wave and play it back, the result will no longer be a square wave. This is because the sharp corners produce unwanted signals (harmonics, in the language of Fourier analysis) as the wave passes through the system. To avoid this, the square wave emitted from the SOD pin is passed through a resistor-capacitor network designed to smooth the wave, as shown in figure 12.3.

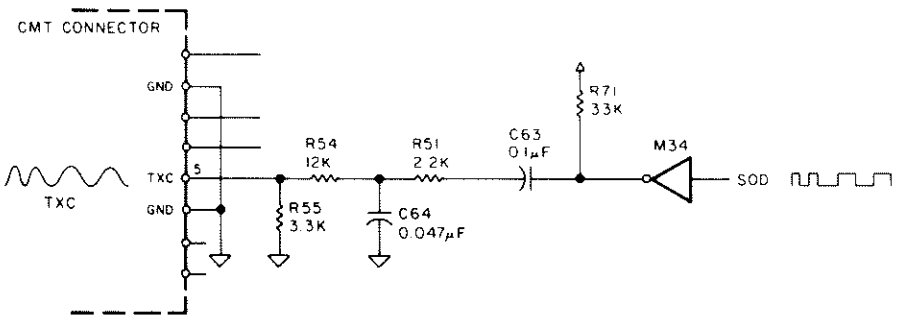

Figure 12.3. Outgoing cassette data

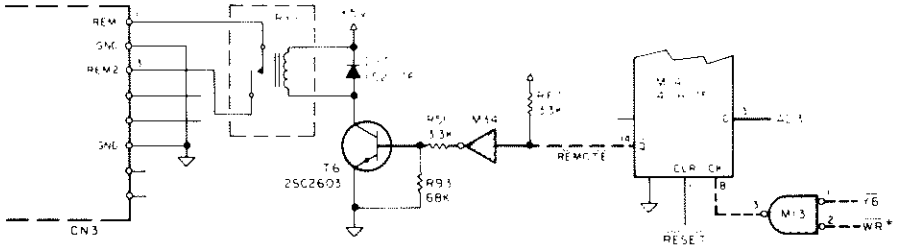

Figure 12.4. Cassette motor control

The recording level in most recorders cannot be adjusted and is instead determined by an ALC (automatic level control) circuit which adjusts the recording level according to the loudness of the incoming signal.

The ALC circuit requires a fraction of a second to stabilize when encountering a signal after a period of silence. Therefore it is good practice to write a few cycles of "leader" before each stretch of data.

Interrupts

Obviously any routine, whether a routine from ROM or one you write yourself, which is designed to read or write cassette data, must have interrupts disabled. If interrupts were handled the timing loops would be thrown off.

Motor Control

The CPU turns the cassette motor on and off by way of bit 3 of output port E8. As that port controls other functions, including the option ROM, the printer strobe, and the clock/calendar strobe, it is important to change only bit 3 when it is desired to turn the motor on or off. The ROM routines store the current contents of the port in FF45, and you should do this too.

The contents of FF45 are ORed with 08 to turn bit 3 on and ANDed with F7 to turn bit 3 off.

Once the bit is set, the accumulator may be loaded to port E8 and also to FF45.

As shown in figure 12.4, the value at bit 3 of the output port drives a flip-flop M14 to produce a signal called REMOTE*, which is low when the Motor is on and high when the motor is off. Remote* is inverted by M34 so that a "1" at bit 3 turns the transistor on. Turning the transistor on provides a ground path for the coil of relay RY1. Energizing RY1 closes the contacts (which are normally open) which shorts pins 1 and 3 of the CASSETTE connector CN3.

These two pins, designated REM1 and REM2 in the Model 100, are connected by the cord to the 3/32" gray plug, as shown in figure 12.1. That plug is intended to go in the "REM" jack of the recorder, which turns the motor on and off. (Obviously the REM1/REM2 control capability may be used for control of other things, as long as the current is not too great.)

The one place in ROM where bit 3 of output port E8 is controlled is at 740F. The routine there is jumped to by the motor-off and motor-on subroutines discussed below.

Published ROM Subroutine Calls

Before cassette data is read or written, interrupts must be disabled and the motor turned on. Both can be accomplished by a call to CTON at 14A8.

The motor is turned off, and the interrupts are enabled by a call to CTOFF at 14AA.

Writing to Cassette

The routine DATAW, called at 6F5B, writes the byte of data in the accumulator to tape. (This routine is discussed above). If the SHIFT-BREAK key is pushed, the routine will return with the carry flag set.

The routine CSOUT, called at 14C1, sends a character to the cassette, keeping track of a checksum. Prior to the call, the byte to send is in the accumulator and the current checksum is in C. After the call, the updated checksum is in C. (This routine calls DATAW, so it also returns with the carry flag set if the SHIFT-BREAK key was pushed.)

By "checksum" we simply mean a running total of bytes sent to tape. The values are added, ignoring the carry bit.

The routine SYNCW, called at 6F46, writes a header and sync byte to cassette. The header is composed of 512 bytes, each 01010101, all run together end to end. This is the steady tone you hear at the beginning of any cassette load. The sync byte is a 01111111, which is a warning to the cassette input routine that the 01's are almost at an end, and that what follows is real data.

This routine will exit with the carry flag set if the SHIFT-BREAK key is pushed.

Reading From Cassette

A single byte may be read with DATAR, called at 702A. DATAR uses the bit-input routine discussed above. Upon return from the bit-input routine, DATAR examines the C register to see how long the waveform was. The threshold applied is 21 decimal or about 2000 Hertz. The 1 or 0 is shifted, and the bit-input routine is called again until eight bits have been received (or the SHIFT-BREAK key pushed).

The result of the read appears in the D register. Like all the "read" routines, it checks to see if the SHIFT—BREAK key was pushed. DATAR responds to that event by returning with the carry flag set.

The routine CASIN, called at 14B0, reads in a character by calling DATAR. If the SHIFT-BREAK key was pushed, the routine ends by turning the motor off and jumping to the BASIC routine that displays the "10" error message on the screen. (If you don't want this response to a SHIFT-BREAK, you can adopt the code from 14B0 to 14C0).

The routine will update the checksum value residing in the C register.

The routine corresponding to SYNCW above is SYNCR. This clever routine, called at 6F85 and located in ROM at 6F85-7042, listens to the tape until it finds that the incoming signal is between about 660 and 5300 Hertz, figures out where the 01's are, and then waits for the sync byte (the 7F mentioned above). At that point, or when the SHIFT-BREAK key is pushed, it returns.

Unpublished Routines

The routine at 22B9 sends DE bytes to cassette, from a buffer pointed to by HL. This is followed by a checksum designed to add up with the previous bytes to make a total of zero, followed by twenty 0 bytes. The routine then turns the motor off and jumps to 0501. Since it never returns, it is unsuitable for assembly language use. It may, however, be adapted to make a good block-write routine.

The routine at 2413 will load up to 256 bytes from cassette into the RAM buffer pointed to by HL. The DE register-pair contains the number of bytes to load. For 256 bytes, the register should contain 00. When the routine finishes, the checksum for these bytes is in A.

By convention blocks of data are written so that the checksum is zero. When the routine is finished, if the checksum is zero, the Z flag is set, representing a successful load.

File Formats

Cassette files are composed of two or more blocks, each of which starts with a header and a sync byte.

The first block in the file contains a header (512 bytes of 55H), a sync byte (7F), the file type (9C for "DO" files, D0 for "CO" files, D3 for "BA" files), the filename (six bytes), and other information (10 bytes).

The second and subsequent blocks are composed of a header (512 bytes of 55H), a sync byte (7F), a data indicator (8D), and then the data followed by a checksum. DO files are broken up into 256-byte blocks, while BA and CO files have one enormous block for the whole file contents.

A file may be opened using the subroutines listed in table 12.1. In each case, prior to the call, the filename must be stored at FC93. Other data, if any, is stored at FACE.

Table 12.1. Call locations for file OPEN

File Type	Symbol	Open for Output	Open for Input
BA	D3	260B	2650
CO	D0	2611	2656
DO	9C	260E	2653

User Experimentation

Nothing in the hardware requires you to use frequencies of 1200 and 2400 Hertz to store data. Nor is there any requirement that the storage technique be single square waves of particular wavelengths. Feel free to modify the ROM bit-input and bit-output routines, or write your own routines for other formats.

For an example of another format, see MCS-80/85 Family User's Manual, pages A1-38 to A1-42. There, bursts of tone and periods of silence of varying duration are used to represent 1's and 0's.

It is also possible to vary the format of data within a cassette file. It is not necessary to use the BA, CO, or DO cassette file formats if you are willing to design your own format.

13

The Liquid-Crystal Display Screen

The Model 100 top panel contains a screen composed of a rectangular array of so-called "pixels", 240 across and 64 down. Each pixel is about 0.8 millimeters square. The pixels are part of what is called a liquid crystal display.

How Liquid Crystals Work

If you hold two polarizing filters between your eye and a table lamp, you will find that the amount of light that passes through is a function of the angle between the two filters.

One easy way to try this is to obtain two sets of Polaroid™ sunglasses. Hold one pair with the lenses side by side and the other pair with the lenses one above the other. No light will pass through. If the glasses are held with the lenses side by side, light will pass through.

This is due to the fact that light can be polarized. When light passes from the lamp toward you and through the furthest polarizing lens, it has been filtered so that all the light is polarized. Let's assume it is polarized vertically. When this vertically polarized light reaches the lens closest to your eye, it will pass through only if the lens is turned the same way as the first lens (so that it also passes vertically polarized light).

Some transparent substances have no effect on light passing through them, while others will twist the polarization of light passing through them. Depending on the particular substance and the distance travelled through the substance, vertically polarized light may be converted to horizontally polarized light, and vice versa.

Such substances are not hard to find. Everyday dextrose, a common sugar made from corn, will twist polarized light. Its very name was chosen because it twists light to the right. "Dextro-" is a prefix which means "to the right".

A liquid-crystal display is composed of a carefully prepared liquid placed between two glass panels. The top panel includes a polarizing filter; the top and bottom panels contain nearly transparent electrodes extending vertically and horizontally.

Room light striking the screen passes first through the polarizing panel. The furthest penetrating light is the light which is polarized, say, vertically. This light bounces off the bottom panel, and (if the polarization has not changed) passes through the polarizing panel a second time, reaching the eye. Any part of the screen where this occurs is perceived as being light in color.

The liquid is designed to twist the light when it (the liquid) is subjected to an electric field. For a certain pixel to be perceived as dark, the LCD driving circuitry must activate the row and column electrodes associated with that pixel every so often. This occurs once every fourteen milliseconds and lasts about half a millisecond.

The electric field being emitted from the electrodes causes the liquid to twist the light a certain number of degrees.

The eye's viewing angle has an effect on the amount of polarization associated with the glass panels, so that no single number of degrees of twist will produce a dark panel for all viewing angles. The rotary control DISP on the right side of the Model 100 is a potentiometer, which varies the voltage used to activate the pixels and optimizes the appearance of the screen for a particular viewing angle.

CPU Control of the Screen

CPU control of the 15360 pixels is accomplished through ports FE (LCD data) and FF (LCD status/command), as well as output ports B9 and BA. Specialized integrated circuits (HD44102 and HD44103) are used to drive the pixels, and provide the 15360 bits of RAM memory needed for LCD operation. (A detailed discussion of LCD I/O ports is beyond the scope of this book. For further information, see the Model 100 Service Manual pages 4-13, 4-14, 4-28, 4-29, 4-30, 7-1, and 7-2.)

The 15360 bit RAM memory mounted on the LCD printed circuit board is not directly addressed by the CPU. Instead, it is loaded through the I/O ports. Some of the regular RAM memory, located from 8000 to FFFF, however is allocated to LCD data. The area from FE00 to FF40, for example, contains the ASCII values presently on the screen. This is the source of information used when BASIC performs the LCOPY command.

It is clear however, that the actual screen contents (the on/off states of the pixels) are not found in RAM at FE00-FF40. For example, if PSET and PRESET are used to turn pixels on and off, the LCOPY command will not convey the results to the printer, even if the pixels form a printable character. Similarly, the patterns that reach the LCD by means of PSET and PRESET will not scroll upwards when the rest of the display does.

Character Formation

When sending data to the LCD screen, the CPU does not send ASCII values to the integrated circuits on the LCD board. Instead, it sends 1's and 0's which are to be stored in the LCD RAM. The LCD RAM is used to drive the individual pixels.

The ROM routines used by the CPU in handling screen output use several different routes for the data. In the case of pixel-specific routines like PLOT and UNPLOT (used in the BASIC commands PSET, PRESET, and LINE) the ROM routine sends addresses and data to the LCD chips to affect only the pixels in question.

When ASCII characters are printed to the screen, the routine first loads the ASCII value to the RAM area FE00-FF40. It later interprets the ASCII value according to a ROM table to determine which bits must be turned on and off to form that character.

Formation of Character Shapes

No character shape information is needed for ASCII values from 0 to 31 (decimal) as these are not printable characters. They instead merely cause cursor movement, etc.

Since each character printed on the screen lies in an array that is six pixels wide and eight pixels deep, forty-eight bits of data are required to define the whole character. (It happens that the rightmost column is always empty in the case of ASCII values 32 to 127. As we shall see, the ROM storage technique takes advantage of this fact to save ninety-six bytes of ROM.)

The character-generation table begins at 7711 and runs to 7BF0. (The ROM routine that uses it is located at 73EE). To see how the table works, print out (using the PEEK function) the five values starting at memory address 78CE (30926 decimal). The contents of these locations are 28, 160, 160, 144, and 124. Now, convert each of these numbers to binary notation. The results are 000111000, 10100000, 10100000, 10010000, and 01111100. If these binary numbers are written in a column, something interesting will emerge, as shown in figure 13.1.

```
30926   00011100
30927   10100000
30928   10100000
30929   10010000
30930   01111100
```

Figure 13.1. Lower-case "y"

Do you see it? Turn the page sideways, and you will see a lower-case "y" among the 1's and 0's.

The table begins at 7711 with the pixel information for an ASCII 32 (decimal) which is a space. As you would imagine, it is composed of all zeros. The table continues, five bytes at a time, through ASCII values 33 to 127.

At location 78F1 the table changes. Since many of the characters beyond 127 use all six columns of pixels, six bytes of data are used for each character. The character with value 128 (which looks a little like a telephone) occupies 78F1, 78F2, 78F3, 78F4, 78F5, and 78F6. From this point to the end of the table, each character uses six bytes. The table finishes at 7BEB-7BF0 for the character with value 255.

It is interesting to use this ROM table to generate the screen characters yourself. The program in figure 13.2 prints the 224 printable characters of the Model 100 to the screen. Each character is displayed twice — once in the normal way by use of the BASIC PRINT command, and a second time with pixels turned on one by one to form the characters.

The two images of the character are identical in appearance because they are both based on the bit-graphics information in the ROM table. What's different is the sort of programming that puts the bits on the screen. When the PRINT command is executed, BASIC invokes machine-language subroutines which extract the information from the ROM table and put it on the LCD screen — all the pixels turn on, forming the character, virtually simultaneously. The second character image reaches the screen much more slowly (you can see that the pixels turn on one by one) because the calculation of which pixels to turn on is done step-by-step in BASIC.

The BASIC PRINT statement gives the ASCII value to an assembly language routine in ROM, which uses the machine language subroutines PLOT and UNPLOT to turn on the proper pixels.

```
10 CLS:FOR CR= 32 TO 255: PRINT @129, CR;
   :PRINT@134, CHR$ (CR);:IFAS < 128THEN
   AD=30481+(CR-32)*5
   ELSEAD=30961+(CR-128)*6
14 FOR COL=0 TO 5: BY=PEEK(AD+COL)
   :IFCR < 128 AND COL=5 THEN BY=0
16 FOR ROW =0 TO 8:IF BY AND (2 ^ ROW)
   THEN PSET(96+COL,24+ROW)
   ELSE PRESET(96+COL,24+ROW)
20 NEXT ROW:NEXT COL:BEEP
100 IF INKEY$="" THEN 100
   ELSE NEXT CR:END
```

Figure 13.2. Program to demonstrate the ROM character-generation table.

If your printer includes bit-addressable graphics, such as the Epson MX-80 with Graftrax, you can print the CODE and GRPH characters directly at the printer. A program to print the values in table 13.1 is shown in figure 13.3.

```
10 kl=31729 : e$=chr$(27)+chr$(75)+chr$(6)+chr$(0)
   : for as=32 to 127:for kt=0 to 43: if peek(kl+kt)
   <>as then 1000 else for co=0 to 5:
   va=peek(kl+kt+co*44) :if va =0 then lprint space $(12);
   :goto 900
20 lprintusing"      ###    "
   ;va;:lprinte$;:ad=5*va+30321:
   if va>127 then ad=va*6+30193
30 for bi=0 to 4: a=peek (ad+bi) :gosub 2000
   :next bi: if va>127 then a=peek(ad+5)
   :gosub2000 else a=0:gosub 2000
900 next co:lprint
1000 next kt:next as:end
2000 a1=0:for ib=0 to 7:
   a1=a1 or ((a and 2 (7-ib))<>0) and 2 ib)
   :next ib: call 5232, a1:return
```

Figure 13.3. Program to print Model 100 characters to bit-addressable dot-matrix printer.

Let's analyze the program line by line and see how the printing is accomplished.

Line 10 sets up a FOR loop which picks an ASCII value and searches the keyboard-decoding ROM table (*see* chapter 6) for the place in the table where that lower-case key is located.

When a particular lower-case key is found, the uppercase GRPH SHIFT-GRPH, CODE, and SHIFT—CODE ASCII equivalents are extracted from the keyboard-decode table through the expression:

```
va=peek(kl+kt+co*44)
```

The resulting six ASCII numerical values are printed by lines 20, 30, and 2000. In a few cases there is no ASCII value. For example, no value is assigned to CODE-G. In such cases the program simply prints twelve spaces.

The Epson printer with Graftrax uses escape sequences to output the bit-addressable graphics. The sequence is an escape (decimal 27), the letter N (decimal 75), the number of columns to be printed bit-style (decimal 6), and a null (decimal 0). The next six values received by the printer are generated much like the array in figure 13.1. Each "1" in binary notation results in a dot on the paper from the print head.

Unfortunately, the Graftrax protocol assigns the bits to the paper "upside-down" from the way the Model 100 assigns the bits to the screen. The FOR loop of line 200 inverts the bits before printing.

The BASIC PRINT routine converts any ASCII TAB (decimal value 9) to a varying number of spaces based on where the Model 100 thinks the printer carriage is positioned on the printer. This is handy for Radio Shack printers that don't know where the next tab stop is, but can cause problems when you want to send escape sequences which sometimes contain the value "9".

The PRINT routine can be circumvented with a machine-language subroutine call as shown on line 2000.

Unshifted		SHIFTed		GRPH		SHIFT-GRPH		CODE		SHIFT-CODE	
39	'	34	"	140	▘			160	´	164	¤
44	,	60	<	153	▖	248	┴	188	ü	221	Ü
45	-	95	_	92	▚	124	|	197	¨	167	▾
46	.	62	>	151	▞	247	┘	207	ß		
47	/	63	?	138	▝			174	½		
48	0	41	)	125	▗			175	¶	166	▸
49	1	33	!	136	▗	225	▪	192	ß	208	Ä
50	2	64	@	156	▙	226	▪				
51	3	35	#	157	▗	227	▪	193	ê	209	ê
52	4	36	$	158	▙	228	▪				
53	5	37	%	159	▚	229	▪				
54	6	94	^	180	▛	230	▪				
55	7	38	&	176	▀			196	ü	212	Ü
56	8	42	*	163	£			194	ï	210	ï
57	9	40	(	123	{			195	ô	211	ô
59	;	58	:	146	▌	245	▌	173	¼		
61	=	43	+	141	±			190	·	168	†
91	[	93	]	96	`	126	~	181	~		
97	a	65	A	133	▘	235	▛	182	à	177	À
98	b	66	B	149	▖						
99	c	67	C	132	▞	255	▚	162	ç	171	⊞
100	d	68	D			237	▙	187	è	215	È
101	e	69	E	143	▝	233	▌	198	ê	214	Ê
102	f	70	F	130	▐	238	▟			191	f
103	g	71	G			253	▜				
104	h	72	H	134	▀	251	▜				
105	i	73	I	142	▗	243	┬	199	ï	213	ï
106	j	74	J			244	├	203	ü	219	Ü
107	k	75	K	155	←	250	┼	201	ï	217	ï
108	l	76	L	154	→	249	┤	202	ô	218	ô
109	m	77	M	129	▘	246	└			165	¥
110	n	78	N	150	↑			205	ñ		
111	o	79	O	152	↑	242	┐	183	ô	178	ô
112	p	80	P	128	▒	241	─	172	¼		
113	q	81	Q	147	▚	231	▬	200	à	216	À
114	r	82	R	137	▗	234	▮			170	▣
115	s	83	S	139	▘	236	▜	169	ç	185	ß
116	t	84	T	135	▣	252	◢			186	ß
117	u	85	U	145	▄	240	┌	184	ü	179	Ü
118	v	86	V					189	ê	222	Ê
119	w	87	W	148	▚	232	▬				
120	x	88	X	131	▣	239	▮	161	à	223	À
121	y	89	Y	144	▚	254	◣	204	ÿ	220	Ÿ
122	z	90	Z			224		206	ß		

Table 13.1. LCD characters

RAM Locations Relating to the Display

A number of RAM locations are set up when the Model 100 is initialized, and should be left undisturbed, as should everything above F5F0, by any user program. The most commonly used values are listed in table 13.2.

Table 13.2. Display variables in RAM

Name	Address	Description
CSRY	F639	Horizontal cursor position
CSRX	F63A	Vertical cursor position
	F63B	Number of active cursor lines
	F63C	Number of active cursor lines
	F63D	Line-8 lock flag
	F63E	Scrolling disable flag
	F648	Reverse "video" flag
	F675	Output flag 0=LCD 1=LPT
	F788	BASIC POS value
	FCCO	Beginning of alternate LCD buffer
	FDFF	End of alternate LCD buffer
BEGLCD	FE00	Beginning of LCD memory
ENDLCD	FF40	End of LCD memory

Published ROM Subroutine Calls

The most frequently used ROM call is LCD, at 4B44. The character in the accumulator is put on the LCD screen at the current cursor position, and the cursor moves to the right (and if necessary, to the next line). This routine is somewhat like the Model I/III routine VDCHAR at 0033. Assuming scrolling has been enabled, then scrolling will occur if necessary.

The LCD routine is quite versatile. While no one would be surprised at its response to printable ASCII values (decimal 32 and above), the routine also handles certain values less than 32. These values are shown in table 13.3.

Table 13.3. Nonprintable values which may not be sent to the LCD routine

Value	ASCII Meaning	Call to Send	ROM Address	Response
07	Bell	4229	7662	Beeping sound
08	Backspace		4461	Moves cursor to left
09	Horizontal tab		4480	Moves to next tab column- 8,16, etc.
0A	Line feed	4225	4494	Line feed-column remains the same
0B	Vertical tab	422D	44A8	Home cursor
OC	Form feed	4231	4548	Clear screen & home
OD	Carriage return		44AA	Cursor to left edge-row does not change
1B	Escape		43B2	Interpret next character

The nonprinting values are decoded according to a ROM table at 438A-43A1. The escape sequences, in turn, are decoded in a ROM table at 43B8-43F9. The addresses of the routines to accomplish the various escape sequences can be determined from the ROM table. There are twenty-one permissible LCD escape sequences which are listed in table 13.4.

Table 13.4. LCD Escape Sequences.

Name	Call	Hex	Chr	ROM Address	Function
		41	A	4469	Up one line unless already at edge
		42	B	446E	Down one line unless already at edge
		43	C	4453	Right one space unless already at edge
		44	D	445C	Left one space unless already at edge
		45	E	4548	Same as printing 0C-clears screen
		48	H	44A8	Same as printing 0B-moves cursor to 1,1
		4A	J	454E	Erase from cursor to end of line
ERAEOL	425D	4B	K	4537	Erase from cursor to end of line
INSLIN	4258	4C	L	44EA	Insert a blank line on LCD at cursor
DELLIN	4253	4D	M	44C4	Delete a line on LCD at current line
CURSON	4249	50	P	44AF	Turn on cursor
CUROFF	424E	51	Q	44BA	Turn off cursor
SETSYS	4235	54	T	4439	Set system line (lock LCD line 8)
RSTSYS	423A	55	U	4437	Reset system line (unlock LCD line 8)
LOCK	423F	56	V*	443F	Lock LCD display (no scrolling)
UNLOCK	4244	57	W	4440	Unlock LCD display (allow scrolling)
	4262	58	X	444A	Repaint screen
		59	Y	43AF	Cursor position (see text)
		6A	J	4548	Same as printing 0C-clears-screen
		6C	1	4535	Erase entire line containing cursor
ENTREV	4269	70	p	4431	Set reverse character mode
EXTREV	426E	71	q	4432	Turn off reverse character mode

continued on following page

In other words, if a character value listed in table 13.3 is "sent to the screen" by the LCD routine, the action shown in the table will be taken. If the character "sent to the screen" is an ASCII escape character (decimal 27) then the character that follows will be interpreted as an escape sequence as shown in table 13.4 and not as a printable character.

* Radio Shack incorrectly lists this as ESC Y.

Several of these routines are used directly by BASIC. The BASIC command CLS is executed by means of a call to the CLS routine at 4231. (CLS is equivalent to the Model I/III routine VDCLS at 01C9.) Also, note that the BASIC command BEEP is executed by means of a call to the routine at 4229 listed in table 13.3.

How to Send Special Characters

There are several ways to program each of the functions listed in these tables. The first is simply to load the character (or characters, in the case of an escape sequence) into the A register, and execute one or more RST 4 instructions. This requires several opcodes to execute however.

Alternatively, you can call the address listed as "CALL address" in the table. If you disassemble that code, you will find that in each case the value is loaded to the accumulator, and the RST 4 is invoked. Because these call addresses have been published by Radio Shack, they are likely to survive any ROM upgrades.

Another means of programming the functions, in the case of the escape sequences, is to use the ESCA subroutine at 4270. Before calling the routine, place the value of the desired escape sequence from table 13.4 in the accumulator.

Finally, it is possible in each case to directly call the routine shown in the "ROM address" column. The advantage is faster execution time, while the disadvantage is that the address may change with a ROM upgrade.

Sending a carriage-return-line-feed combination to the screen can be accomplished with a call to CRLF at 4222 as shown below:

```
4222  3E  0D    MVI  A,0D   ;LD A, 0D
4224  E7        RSI  4      ; send to LCD
4225  3E  0A    MVI  A,0A   ;LD A, 0A
4227  E7        RST  4
4228  C9        RET
```

This routine will save four bytes each time you use it.

Sending Characters to the Printer

The LCD routine is versatile in other ways. It relies on a flag stored at F675 which, if zero, indicates that output should be directed to the LCD, as the name suggests. If the value at F675 is nonzero, the value in the accumulator will be sent to the line printer instead. This may be seen, for instance, in the code for LLIST at 113B and the code for LIST at 1140:

```
113B  3E  01      MVI A,01    ;LD A, 01
113D  32  75  F6  STA F675    ;LD (F675), A
1140  C1          POP B       ;beginning of LIST routine
```

To send output to the printer, simply set the printer flag at F675 before calling RST 4.

How to Call 4B44

If you dissassemble all of ROM, you will see that LCD is never invoked by a CALL 4B44. Instead the ROM designers placed a JP 4B44 at ROM address 0020, so that an RST 4 (sometimes called an RST 20) opcode may be used, saving two bytes of ROM each time it is called.

Obviously, not all uses of the LCD routine may be accomplished by an RST 4. For example, a conditional call cannot be accomplished in less than three bytes. This may be seen in ROM at 4B3F and at 54BC.

Other Published LCD ROM Routines

Two routines allow the machine language equivalent of the BASIC commands PSET and PRESET (which are abbreviations of "pixel set" and "pixel reset"). These are PLOT at 744C and UNPLOT at 744D, respectively. In each case the pixel to be changed is addressed through the DE register-pair. D contains the X coordinate between 0 and 239. E contains the Y coordinate between 0 and 63.

Cursor Position Routines

The routine, POSIT allows a machine language program to handle the cursor directly. POSIT, at 427C, moves the cursor to the position given in the H (column 1-40) and L (row 1-8) registers. This routine is almost identical to and is accomplished by the ESC-Y sequence.

The ESC-Y sequence is four characters long. It is composed of an escape (decimal 27), a Y (decimal 89), the desired row plus 31 decimal (sum varies from 32 to 39), and finally the desired column plus 31 decimal (sum varies from 32 to 71). Building up this escape sequence takes many bytes of instructions. The call to POSIT at 427C is always more economical. To see this, look at the code at 427C-4289:

```
427C  3E  59      MVI A,59        ;LD A,59 "Y"
427E  CD  70  42  CALL ESCA
4281  7D          MOV A,L         ;desired row
4282  D6  1F      ADI 1F          ;make it printable
4284  E7          RST4            ;print it
4285  7C          MOV A,H         ;desired column
4286  C6  1F      ADI 1F
4288  E7          RST 4
4289  C9          RET
```

The routine is interesting for several reasons. First of all, it is the only four character escape sequence for the LCD. Second, it illustrates that characters in the escape sequence must be greater than 32 decimal, (i.e. printable) so that the RST 4 routine won't mishandle them. Third, it shows what lengths Microsoft went to to make sure that the ROM operating system is cleanly structured. Virtually all routines affecting the screen are, at bottom, communicated to the screen through the RST 4 routine. This allows the programmer in charge of RST 4 to be sure that he has exclusive control over the inner workings of the screen.

POSIT is handy for moving the cursor about, as well as for returning it to its former position when printing. To do this, get the current cursor position with LHLD F639, store it with PUSH HL, print at the screen as desired, and then execute POP HL and CALL 427C.

Unpublished ROM Routines for the LCD

Several routines are available which relate to the LCD, other than those published by Radio Shack. These routines may change if the ROM is altered creating problems.

A routine at 001E simply sends a space to the screen. A call to this location would take up three bytes, which is no savings over simply loading a 20 hex to the accumulator (two bytes) and calling RST4. This routine would only save bytes if you were at the end of a subroutine and needed to print a space, then return. A jump (not a call) to 001E could do both at once.

Another nice routine is located at 11A2. Assume HL points to a string of values (known to be printable) terminated with a 00 hex. Calling 11A2 will send that string to the screen. An identical routine is located at 5A58.

Finally, a routine at 1BE0 prints up to 256 characters to the screen, filtering out unprintable characters. The values are pointed to by HL, and the number of values to print is stored in the B register. (To print 256 values, load 0 in the B register.) This routine is handy for memory dumps. One drawback is that carriage returns and line feeds are suppressed. Recall that with this routine, as with any routine using RST 4, the output may be routed to the printer simply by changing the value at F675.

The routine at 27B1, in a rather circuitous way, sends to the screen the character string pointed to by HL and ending with either an 0 or a quotation mark.

The routine at 5791 sends to the screen the character string which is pointed to by HL and ends with either a 0 or a quotation mark. The whole output is preceded by a carriage return if the cursor is not already located at the left edge of the screen.

14

The Bar Code Reader

The Model 100's 9-pin connector labeled BCR with hardware designation CN2, can be used to attach a bar code reader wand. Pins 5 and 7 are grounded, pin 9 is a 5 volt source, and pin 2 provides data to the Model 100 from whatever is plugged into the BCR connector. The pins are shown in the illustration on page 209 of the Model 100 user's manual.

The signal at pin 2 has two hardware designations— it is called RXDB on page 209 of the user's manual and RXD on page 4-11 of the service manual. Regardless of the designation, it goes two places in the Model 100 — to one of the interrupt pins of the CPU and to bit 3 of input port BB.

The bar code reader interface circuitry is shown in figure 14.1. The power for the wand comes from pin 9 of CN2 and is designated VDD. The phototransistor signal enters the Model 100 at pin 2, designated RXD. After inversion, the BCR signal goes to bit 3 input port BB, implemented in hardware by port C of the PIO chip. The signal path to the CPU interrupt pin is designated RST 5.5.

The following BASIC program affords an easy means of examining how the input port functions:

```
1 IF 8 AND INP(187) THEN BEEP:GOTO 1 ELSE 1
```

Due to the pull-up resistor R70 at the input terminal, the expression 8 AND INP(187) evaluates to zero when nothing is attached to CN2. As a result the ELSE clause of the IF statement is executed, and no beep is emitted.

With a BCR wand attached (and no custom BCR software loaded), touching the wand to a white surface should produce a beep. This is because the wand contains a light source, powered by pin 9, and a phototransistor, which grounds pin 2 when light is reflected back to it.

The interrupt capability of the BCR interface works through what is called the RST 5.5 input of the 8085. Grounding CN2 pin 2 sends an interrupt signal to the CPU. The interrupt status of the CPU, which has been previously set by the SIM and EI or DI in structions, determines whether an interrupt will actually occur in response to the interrupt signal.

Determining When to Start Reading a Bar Code

The most elegant (and complicated) method of determining when reading of a bar code has commenced is through the hardware interrupt. The EI and DI instructions enable and disable interrupts (*see* chapter 15). Thus, the DI instruction prevents the occurrence of an interrupt due to grounding of CN2 pin 2.

If interrupts are enabled, namely if the EI instruction has been executed more recently than the DI instruction, then the most recent update of the interrupt mask (done through the SIM instruction) determines whether the BCR signal will be able to cause an interrupt.

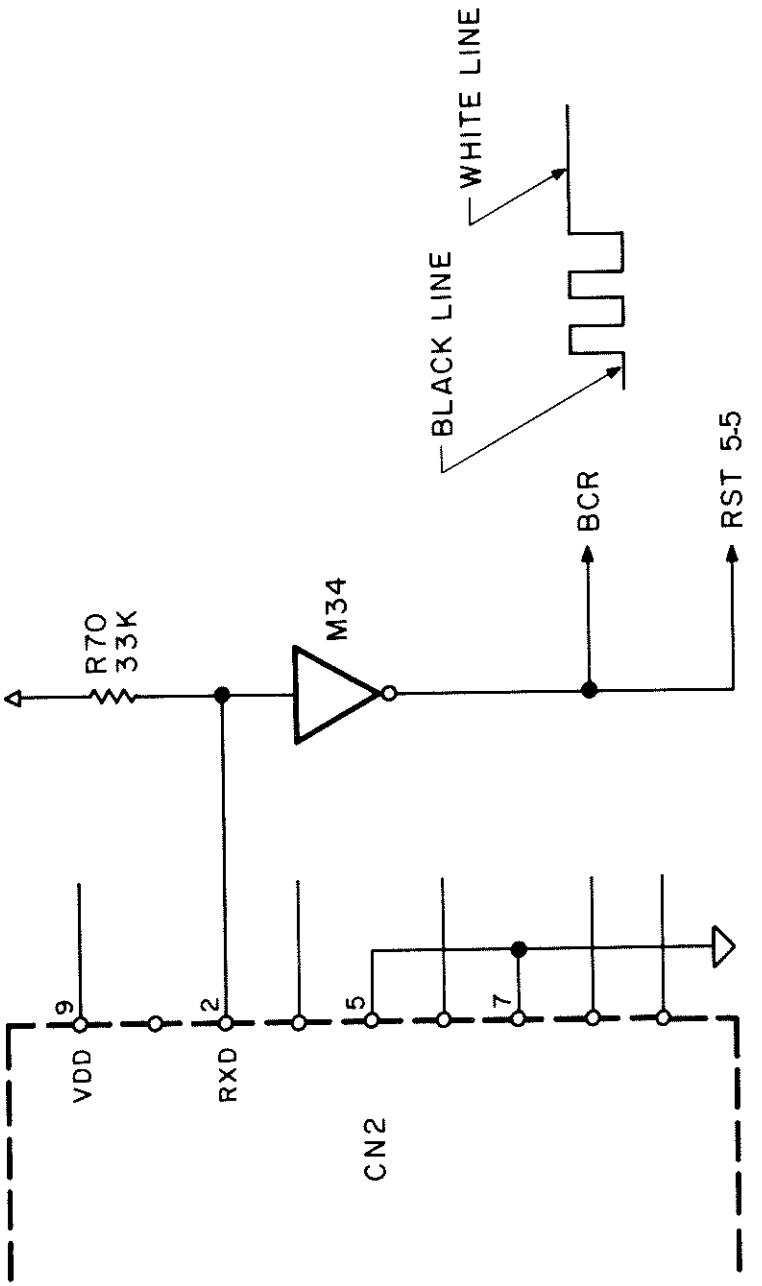

Figure 14.1. Bar code reader interface circuitry

The SIM instruction (*see* chapter 2) fulfills either of two functions depending on the contents of the accumulator. If bit 3 is set, the interrupt mask is updated, based on bits 0, 1, and 2. If bit 6 is set, the cassette output pin is updated, based on bit 7. It is possible to update both, by setting bits 3 and 6. However such a practice is generally not undertaken.

Of the three interrupts controlled by the interrupt mask, it is bit 0 which controls the bar code reader interrupt. Setting that bit masks, or defers, any BCR interrupts. The present Model 100 ROM sets that bit whenever it updates the interrupt mask, so that BCR interrupts are never enabled.

Since the present contents of the interrupt mask are available to the CPU through the RIM instruction (*see* chapter 15), it is possible to read the interrupt mask into the accumulator, set or reset bit 0, set bit 3, and execute the SIM instruction. This action masks or unmasks the BCR interrupt without disturbing the status of the interrupt masking for the other two interrupts (UART data ready and clock/calendar timing pulse).

Handling the Interrupt

As discussed in chapter 15, when the BCR interrupt is enabled and unmasked and when the interrupt occurs due to the wand encountering a white surface, a subroutine call to 2CH occurs. 2CH is located in ROM; this location is a jump to F5F9. The user should put a jump to the interrupt-handling routine at that address. The routine should end with an EI and a RET.

Polling the Bar Code Reader

BCR data can be input without using interrupts using a technique called *polling*. When BCR data is to be input, the CPU repeatedly inspects the incoming signal at bit 3 of input port BB. Usually the signal is a logic zero, so a transition to a logic one indicates scanning has begun. Since the remainder of the bar code is read rather quickly, the CPU must monitor the input port very closely. The CPU ignores any other inputs except the break key.

Reading the Bar Code

Reading the bar code is a complicated matter. Assuming the wand moves along the image at approximately 33 inches per second, a code one inch wide has come and gone in 30 milliseconds of read time. There may be thirty bars in the code, meaning that sixty black/white or white/black transitions must be detected, with an average of about 500 microseconds in between. The time interval between detection of each pair of transitions must be noted for later analysis.

Here is a routine for detecting a pair of transitions, assuming a white region has just been detected:

```
        MVI D,00        ; how long is white band?
LOOP:   IN BB           ; get BCR data
        ANI 08          ; get bit 3
        INR D
        JNZ LOOP        ; if still white, loop
        IN BB           ; it's black, look again
        ANI 08          ; maybe one black value
        INR D
        JNZ LOOP        ; was a fluke
        MOV M,D         ; (HL) contains width of bar
        INX HL
        MVI D,00        ; how long is black band?
LOOP1:  IN BB           ; get BCR data
        ANI 08          ; get bit 3
        INR D
        JZ LOOP1        ; if still black, loop
        IN BB           ; it's white, look again
        ANI 08          ; maybe one white value
        INR D
        JZ LOOP1        ; was a fluke
        MOV M,D         ; (HL) contains width of bar
        INX HL
```

The loops in the above routine each last 31 machine cycles, or about 13 microseconds. The D register averages a final value of 40; less if the band was narrow and more if it was wide.

When the entire code has been read, the intervals must be analyzed to see which bars and gaps were wide and which were narrow. Then, based on the code being used (Universal Product Code, 3 of 9,

Interleaved 2 of 5, Codabar, etc.), the wide/narrow information must be translated into digits or ASCII characters.

If the coding system includes a parity check, it is a good practice to announce bad parity with a distinctive tone, giving the user the opportunity to scan the pattern again.

Using the BCR Connector for Purposes Other Than Reading Bar Codes

Devices other than a wand can be plugged into the BCR connector. Any information source represented by a short, or open, between pins 2 and 5 of CN2 may be read by the CPU as discussed previously.

It is a good practice to use an optocoupler to protect the BCR interface. Use a standard coupler such as Radio Shack 276-1654, wired as shown in figure 14.2. The resistor limits current to twenty milliamperes, protecting the five-volt supply and the light-emitting diode at pins 1 and 2 of the coupler. Closing the switch causes a logic one at bit 3 of input port BB.

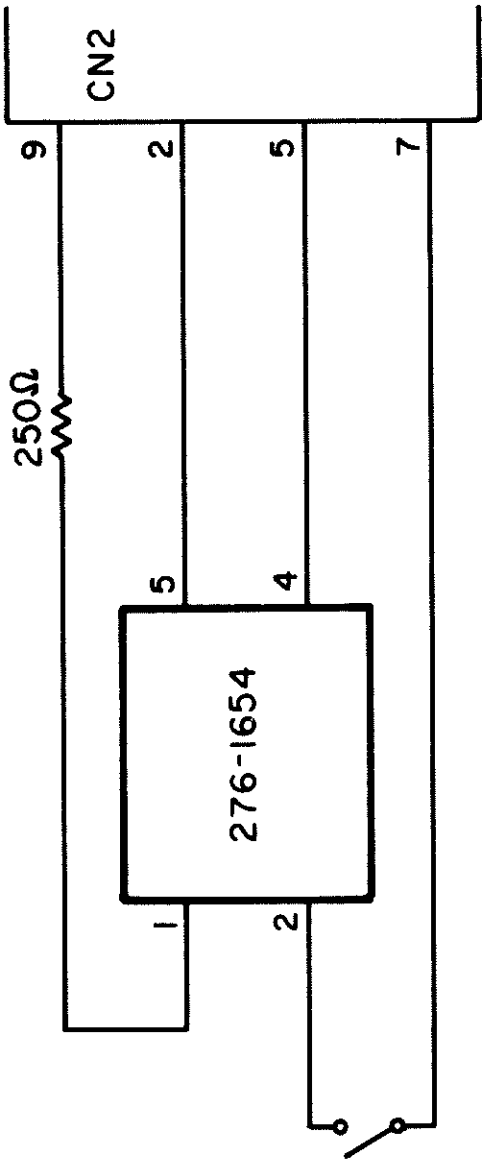

Figure 14.2. Discrete input at BCR connection

15

Interrupts

Rarely does any event proceed from start to finish without being interrupted. For example, if you are reading a book and the telephone rings, you are likely to put a bookmark in the book, answer the phone, and later return to your reading.

Often in programming a computer to accomplish a task, the equivalent of "answering the phone" is a useful addition to that program. For example, when the Model 100 is executing TELCOM, it monitors the keyboard to see if a key had been pressed since its last check. When a key is pressed, the Model 100 must respond accordingly. These responses might include: opening a file for uploading or downloading, scrolling the screen, or transmitting a character to the distant computer by way of the UART.

For these reasons, and for many others that will be apparent, the Model 100 uses *interrupts* for several time-related functions. Interrupt can be defined as a means by which program execution is interrupted by a stimulus which originates outside the CPU. The CPU's usual sequential execution of instructions is brought to a temporary halt, and control is transferred to program instructions stored at an interrupt address associated with that particular interrupt. The CPU address being executed at the time of the interrupt is stored on the stack. This allows CPU instruction execution to resume properly once the interrupt instructions have been executed.

When the conditions prompting the interrupt has been satisfied, a properly written program invokes the RET instruction, returning program control to the address about to be executed prior to the arrival of the interrupt signal.

To accomplish all this, the Model 100 incorporates:

- a CPU designed to be able to respond to several interrupt signals
- a hardware layout connecting several interrupting devices to the CPU
- a ROM operating system which provides routines for the CPU to perform in the event of an interrupt condition.

The 8085 can accept as many as five separate electrical interrupt signals from other devices. In that respect it differs from its closest relatives, the 8080 and the Z80, which each accept only one.

The five interrupt signals to which the 8085 will respond are shown in table 15.1. This table also depicts the address to which the 8085 transfers control and the interrupt source to which the 8085 is connected in the Model 100.

Pin 6 of the 8085, the TRAP line, is wired to the power supply (*see* figure 16.1). A signal designated LPS is generated when the battery or external power source drops below 3.7 volts. *See* chapter 16 for a discussion of the power supply.

This is not the same as the low battery warning light, which illuminates much sooner than the LPS signal, namely when the power drops below 4.1 volts. The CPU cannot detect the low battery warning light.

The internal design of the 8085 is such that the LPS signal causes what is functionally equivalent to a subroutine call to 24H. Disassembly of the ROM reveals a jump to F602 at that location. F602 generally contains a jump to 1431, which in turns results in:

- storage of a few crucial parameters that are used later upon power-up.
- a powering down of the Model 100 through bit 4 of output port 178 or 186, the power-control signal.

You could store a different jump at F602 if a different means of handling low power was desired. Such a handler must end with a jump to 1431.

Pin 7 of the 8085, the RST 7.5 line, is wired to the clock/calendar chip. A signal designated TP is generated approximately once every four milliseconds. The actual pulse rate is 256 hertz. The 8085 examines 3CH to determine the next action. The ROM instruction there is DI (disable interrupts) followed by a jump to 1B32, which is a call to F5FF. That part of RAM usually contains a RET (return), though you could store a jump there to a routine to be prompted by the TP signal. Such a routine must end with a RET instruction.

The return brings control back to the TP-interrupt routine at 1B35, which performs a variety of housekeeping activities and ultimately returns control to the interrupted program.

Note it is possible to reprogram the clock/calendar chip to generate the TP signal at rates other than 256 hertz. *See* chapter 11.

Pin 8 of the 8085, the RST 6.5 line, is wired to the Universal Asynchronous Receiver/Transmitter (UART). A signal designated DR (Data Ready) is generated when the UART has received a character, whether from the RS-232C port or from the phone modem. The RST 6.5 interrupt generates a subroutine call to 34H, which contains a DI opcode and a jump to 6DAC, which in turn calls F5FC. Usually F5FC contains a simple return; the routine at 6DAF then retrieves the received character and places it in a buffer. You could put a subroutine call or jump at F5FC to handle incoming characters differently.

Pin 9 of the 8085, the RST 5.5 line, is wired to the BCR socket.

The internal design of the 8085 results in what is functionally equivalent to a subroutine call to 2CH. Disassembly of the ROM at

that location reveals a DI instruction followed by a jump to F5F9.

F5F9 generally contains a simple return, assuming the bar-code reader driver has not been loaded. The user could put a jump instruction at F5F9 if different handling of a received BCR signal were desired; such a handler must again end with a return.

Pin 10 of the 8085, the INTR line, is wired to the Expansion Bus connector, pin 17. A signal designated INTR (Interrupt) is generated when that pin is pulled high by a device plugged into that connector.

The response of the 8085 to this interrupt is identical to that of the 8080 and similar to that of the Z80. A Restart subroutine call occurs to the location jammed onto the address bus by the interrupting device. A detailed discussion of interrupt jamming is beyond the scope of this book. For further information, *see* Larsen, Titus and Titus, *8080/8085 Software Design Book 2,* chapters 2 and 3 and Leventhal, *8085 Assembly Language Programming* chapter 12.

Interrupt Priorities

The 8085 is designed with a priority circuit that decides which of two simultaneous interrupts will determine the interrupt routine to be followed. The priorities are detailed in table 15.2.

Masking and Disabling of Interrupts

Handling of an interrupt, even if it simply results in a RET instruction, takes time. Since this interval can throw off time-sensitive functions, such as cassette I/O. It is necessary to be able to disable, or mask, most interrupts. This is accomplished through the EI, DI, RIM, and SIM instructions.

As is indicated in table 15.2, the LPS interrupt cannot be masked, although the other four can be. You can either disable all four through the DI instruction or enable interrupts through the EI instruction.

In the 8080 and Z80, this reflects the full range of interrupt mask choices. In the 8085, however, three of the four interrupts (TP, DR, and BCR) may be individually masked, through the use of the SIM instruction, *see* table 15.3.

Masking accomplished by the SIM instruction is cumulative with that accomplished by the DI instruction. Enabling of, say, the DR interrupt requires both an EI and the appropriate SIM.

As a result, certain combinations of enabled and disabled interrupts are not possible. For example, no set of instructions can bring

about an enabled DR interrupt and at the same time a disabled INTR interrupt.

Let's look at how the ROM routines enable and disable the various interrupts. Disassembly of the subroutine at 765C reveals:

```
765C    F3      DI
765D    3E 1D   MVI A,1D       (LD A,1D)
765F    30      SIM
7660    FB      EI
```

At 765D, the accumulator is loaded with 1D, which is 00011101 in binary. Bits 4, 3, 2, and 0 are on. The next instruction, SIM, loads the contents of the accumulator into the interrupt mask. It is important that bit 6 be off, so that this execution of the SIM instruction will leave the cassette output (SOD) unchanged. Because bit 5 is on, the TP interrupt, if pending, is reset.

Because bit 3 is on, the mask values in bits 2, 1, and 0 are updated. The result is that the DR interrupt is enabled, but the TP and BCR interrupts are disabled.

Strictly speaking, it is not the SIM instruction that enables the DR interrupt; SIM merely sets the stage. It is the EI in the following line that enables the DR and the INTR interrupts.

Other ROM routines enable different combinations of interrupts. For example, the routines at 1B3B and 457D enable the DR interrupt but not the TP or BCR interrupts. They do not reset the TP request. The routine at 6CE4 resets the TP request and enables the DR and TP interrupts but not the BCR interrupt. The routines at 4584, 6D69, 71F6, 726D, 73EA, and 743E enable the DR and TP interrupts but not the BCR interrupt.

When an interrupt is handled by the 8085, all other interrupts are disabled, just as if a DI instruction had been executed. As a result, no other interrupt, whether higher or lower in priority, is handled. You should reenable interrupts somewhere in the interrupt-handling routine. This is usually done just before the final RET instruction. The actual reenabling occurs after the completion of execution of the instruction immediately following the EI. If the 8085 had been designed so that reenabling occurred with the EI instruction itself, the

presence of any pending interrupts would take up too much space in the stack area.

Radio Shack Model I and III programmers know that disabling of interrupts using CMD "T" interfered with system timekeeping. No such problem exists within the Model 100, as time is kept by a clock/-calendar chip whose function is unaffected by CPU status and which continues to function after the Model 100 is turned off.

Masking of interrupts is used to great advantage in Model 100 BASIC. The commands ON COM GOSUB and ON MDM GOSUB are driven directly by the DR interrupt and are masked by COM STOP and MDM STOP.

Similarly, the commands ON KEY GOSUB and ON TIME$ GOSUB are driven by the TP interrupt and masked by KEY STOP and TIME$ STOP.

Uses for the RIM Instruction

The RIM instruction is used for cassette input (*see* chapter 12). This is, in fact, the only purpose to which it is put by the Model 100 ROM. However, RIM can also be used to read the mask status of certain interrupts.

Suppose you desired to change only one bit of the interrupt mask — for instance to enable, the DR interrupt while leaving unchanged the masked or unmasked status of the TP and BCR interrupts. This could be accomplished by:

```
RIM
ANI 05      (AND A,05)      ; trim bits 1 and 3-7
ORI 08      (OR  A,08)      ; turn on bit 3
SIM
```

This routine's RIM instruction reads the present mask status of the TP and BCR interrupts (bits 0 and 2 of the interrupt mask). It then turns off bits 1 and 6 (through the AND instruction) to enable the DR interrupt. To leave the SOD signal undisturbed, it turns on bit 3 to allow updating of the interrupt mask and updates the mask with the SIM instruction.

Suppose you wished to poll the UART DR line, rather than service it through the interrupt routine at F5FC. To 8080 and Z80 programmers this would appear to be impossible in the Model 100 as a hardware matter, since the UART DR line does not go to any I/O port circuitry (such as the PIO) but instead goes only to CPU pin 8, which is an interrupt pin.

The 8085 RIM instruction, however, allows such polling. As shown in table 15.4, after a RIM, bits 4, 5, and 6 of the accumulator indicate whether BCR, DR, and TP interrupts, respectively, are pending.

Table 15.1. Types of Model 100 interrupts

CPU Pin	8085 Designation	M 100 Designation	Jump Address	Source
6	TRAP	LPS	24H	Power supply
7	RST 7.5	TP	3CH	Clock/calendar chip
8	RST 6.5	DR	34H	UART
9	RST 5.5	BCR	2CH	Bar-code reader
10	INTR	INTR	varies	Expansion bus

Table 15.2. Interrupt priority and masking

Priority	Signal	How Masked
1	Low power signal	Nonmaskable
2	Timing pulse	DI or SIM bit 2
3	Data ready	DI or SIM bit 1
4	Bar code reader	DI or SIM bit 0
5	Expansion bus	DI

Table 15.3. Set interrupt mask (SIM) configuration

Bit	8085 Function	Model 100 Use
0	RST 5.5 mask	Ignore BCR interrupts
1	RST 6.5 mask	Ignore UART DR interrupts
2	RST 7.5 mask	Ignore 256-Hz interrupts
3	mask set enable	Mask set enable
4	reset 7.5 interrupt	Reset 256-Hz interrupt
5	not used	Not used
6	SOD set enable	Allow updating of cassette output flip-flop
7	SOD	Data for cassette output

Table 15.4. Read interrupt mask (RIM) configuration

Bit	8085 Function	Model 100 Use
0	RST 5.5 enabled	BCR interrupt enabled
1	RST 6.5 enabled	UART DR interrupt enabled
2	RST 7.5 enabled	256-Hz interrupt enabled
3	IE status	IE status
4	RST 5.5 pending	BCR interrupt pending
5	RST 6.5 pending	UART data ready
6	RST 7.5 pending	256-Hz pulse pending
7	SID	Cassette data in

16

The Power Supply

The Model 100 is designed to draw its power from four AA cells or from an AC adapter (catalog number 26-3804). Main power is controlled by the ON/OFF switch SW-5 on the right side of the unit.

With no peripherals attached and no sound being emitted from the beeper, the Model 100 draws about 350 milliwatts. Radio Shack maintains that under full load conditions it may draw as much as 1100 milliwatts, although this author has never seen it draw more than about 975 milliwatts.

The actual current drain at a given instant is a function not only of the internal load but also of the voltage supplied. This is because the Model 100 contains a DC-to-DC convertor which creates all needed voltages from whatever DC level is supplied. The convertor requires a

certain amount of source power (source voltage multiplied by source current), so that if the DC source is of a lower voltage, the convertor makes up for it by drawing more current. It is able to do this with input voltages ranging from 7 volts (from the AC adapter) to as little as 3.7 volts (from batteries that have run down almost to the point of prompting a CPU-induced power-down).

The DC-to-DC Convertor

The most complicated part of the Model 100 power supply is the DC-DC convertor, shown in the upper right corner of figure 16.1.

Transformer OT2, on the right, is used to provide the various voltages. The designers of the Model 100 took advantage of the fact that transformers are more efficient at higher frequencies; transistors T21 and T22 provide an alternating current of 100 kilohertz to the primary winding. This is substantially higher than the 60 hertz alternating current used in the transformer of the AC adapter.

The currents from the center-tapped secondary winding are rectified, filtered, and regulated to produce the various voltages required in the computer. Minus 5 volts, designated VEE, is provided for the RS-232C driver, for the various operational amplifiers in the cassette and modem circuits, and for the liquid crystal display. Plus 5 volts, designated VB and backed up by a nickel-cadmium (nicad) cell, is presented for the RAM memory and clock/calendar chip. Plus 5 volts, designated VDD, is provided for everything else in the computer.

Memory Power

Memory protection is provided by a nicad cell rated 3.6 volts at 50 milliampere-hours. The cell, shown in figure 16.2, appears in the schematic as 3-51FT and bears reference number P-36 in the service manual. Since the rated protection time for a 32K machine is eight days, this means that the 32K of RAM together with the clock/calendar chip draw something less than 0.25 microamperes.

If a nicad is repeatedly discharged only halfway and then recharged, it will lose the second half of its capacity. Thus, most nicads should always be discharged fully before recharging. Nicads are known as *discharge memory* devices due to their propensity to lose the unused portion of their capacity when they are not discharged completely before recharging.

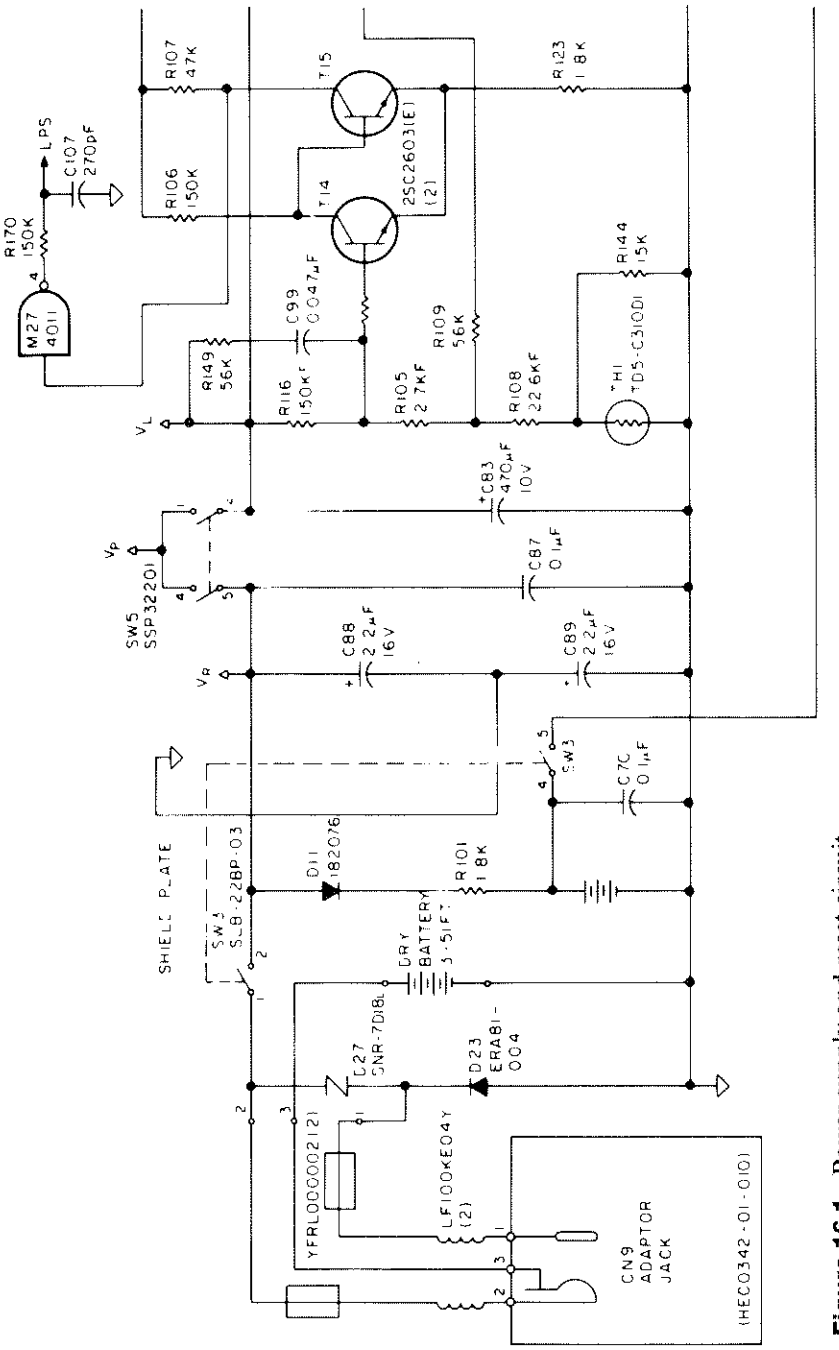

Figure 16.1. Power supply and reset circuit

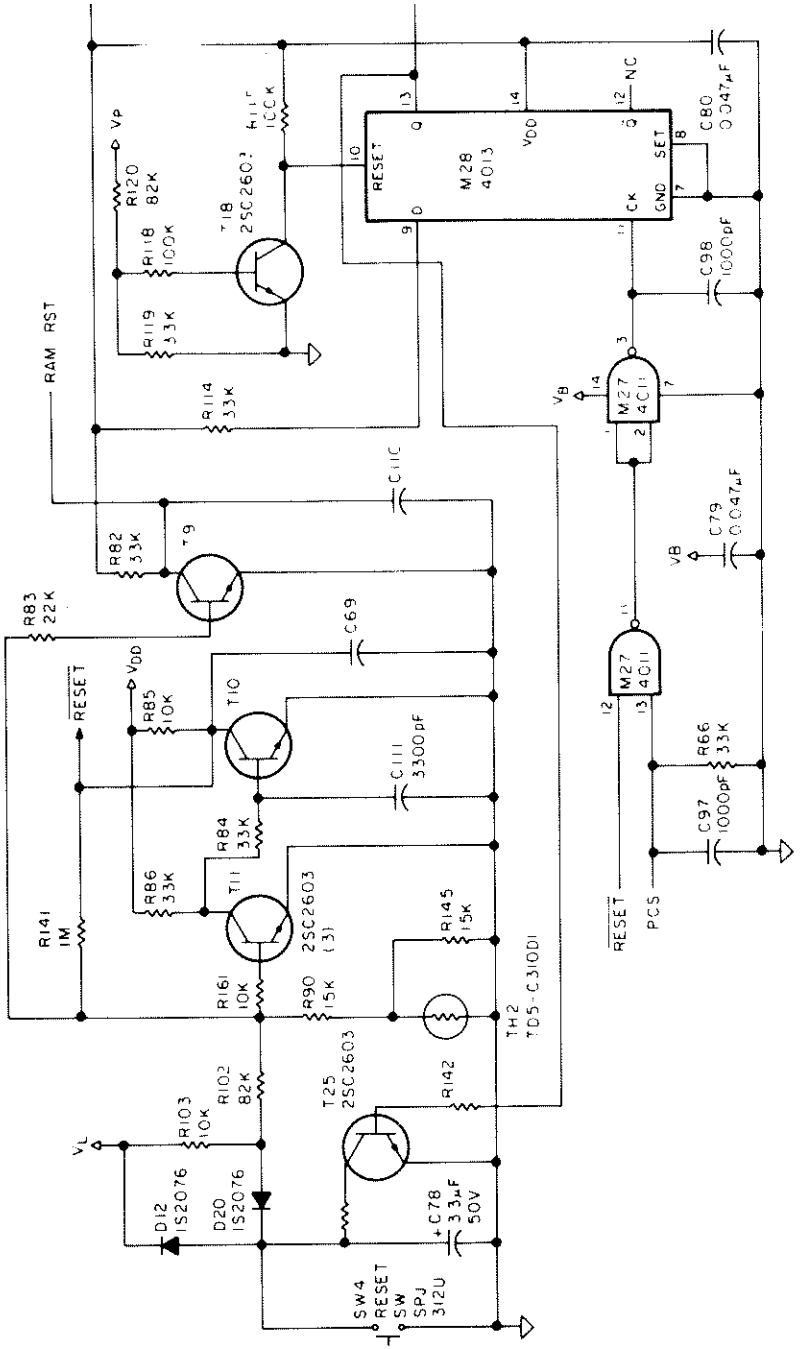

Figure 16.1.(cont.) Power supply and reset circuit

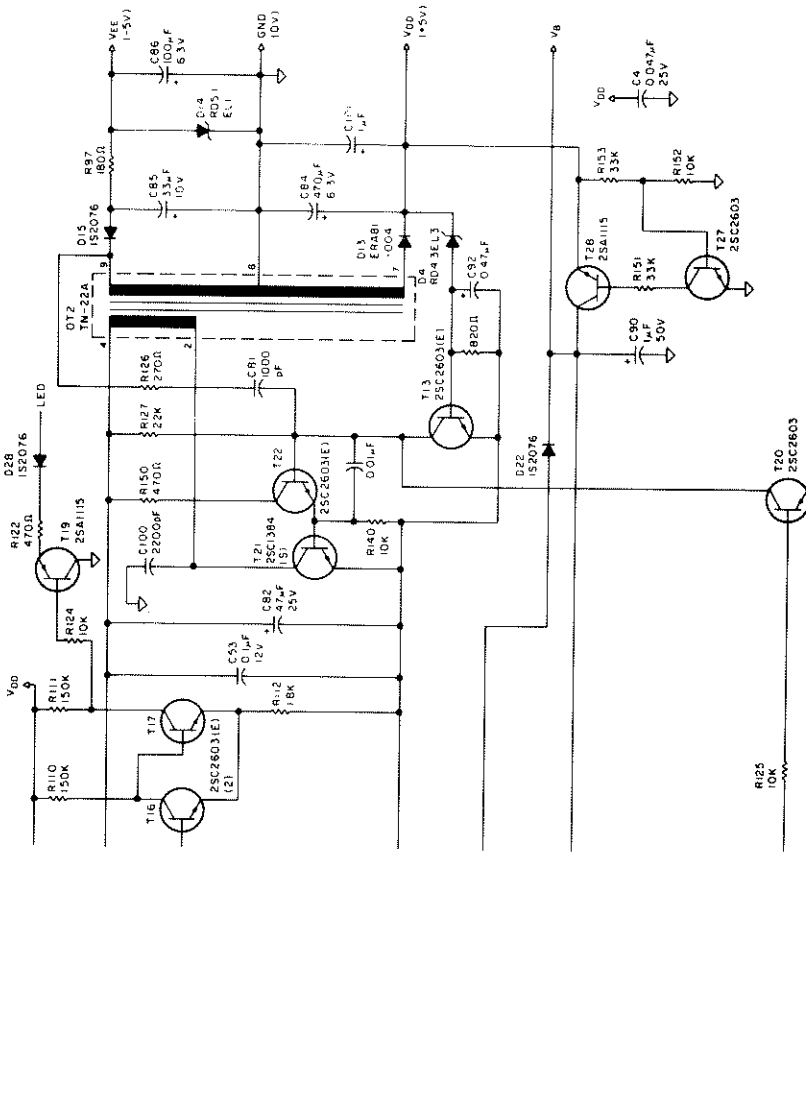

Figure 16.1.(cont.) Power supply and reset circuit

In the Model 100, however, the nicad is recharging whenever the four AA cells are in place and the AC adapter is connected. The only time the nicad discharges is during the interval when the "low battery" light comes on and when fresh batteries are inserted. Therefore, always keep a spare set of AA batteries on hand to be installed as soon as the LOW BATTERY light comes on.

Before servicing the computer and during installation of expansion RAM modules, you must remove all sources of power from the circuit board. Because the nicad is soldered in place and cannot easily be removed, a switch SW3 is provided to disconnect the nicad. This double-pole single-throw slide switch labeled "memory power" is located on the bottom of the computer. Turning it off for more than a few seconds results in loss of all data in RAM and loss of proper clock/calendar timekeeping.

It is hard to imagine any reason to turn off memory protection other than for servicing the computer. If the computer is going to be stored for a considerable length of time, it is a good practice to remove the AA cells as a precaution against leakage. In any event you should back up all important files on tape, since you cannot rely on the nicad lasting more than a few days.

An extended storage period provides the ideal opportunity to discharge the nicad fully so as to restore its full capacity. To do so, leave the memory power switch on.

Low-Power Signals

The power supply includes circuitry to warn both you and the CPU of impending battery loss. A voltage divider, located at the upper center of figure 16.1 and composed of TH1, R144, R108, R105, and R116, yields voltages calculated to trigger the "low battery" light-emitting diode through transistors T16 and T17. The voltage divider also triggers LPS, the low power signal, through T14 and T15.

The connection between the "low battery" transistors and the LED is rather circuitous. The transistors drive T19, which is wired to CN8 on the main printed circuit board. A two-wire cable is plugged into CN8 whose other end is soldered to location CN3 on the LCD panel, which is in turn wired to the LED itself, designated P-24.

The divider is set up so that as the supply voltage falls, the LED illuminates first (at about 4.1 volts). Later LPS is activated (when the supply has fallen to about 3.7 volts). There is no connection from the LED circuit to the CPU. The LPS signal is the first clue the CPU has that the supply voltage is falling.

Since the switching thresholds of the transistors are affected by temperature, thermistor TH1 is provided to improve consistency of circuit response over a range of temperatures.

Whenever power drops far enough to activate the LPS transistors (and this includes simply turning off the main power switch SW-5), LPS goes to two places. It appears at bit 7 of input port D8 as a logic zero (changed from the usual logic one) and also activates the TRAP interrupt line of the 8085 CPU.

The TRAP interrupt, which cannot be masked or disabled, disables all other interrupts and causes a subroutine call to 24H. This results in an orderly termination of calculations in progress. The computer then turns itself off by turning on bit 4 of output port BA. The resulting signal, called PCS (power control signal) toggles flip-flop M28, which appears at the bottom of figure 16.1. An output of the flip-flop inhibits the feedback which usually sustains the convertor oscillator, cutting off energy to the VEE and VDD lines. As a result, all activity stops except clock/calendar timekeeping and RAM data preservation. The ROM code for the power down is located at 1431 through 1458.

Reset Circuitry

When power is turned on, circuitry at the lower left of figure 16.1 provides a RESET-NOT signal to initialize the CPU, the PCS circuitry, the LCD, and flip-flop M14. The CPU provides RESET* to initialize the modem, the UART, the PIO, flip-flop M36, and any device plugged into the Expansion Bus. The reset circuitry also provides a RAM RST signal, which disables the RAM chips to protect their contents during power-down. This signal is also sent to the Expansion Bus to protect any RAM that is installed there. The single-pole single-throw RESET button SW-4 at the rear of the computer provides the same reset signals as does the on/off switch.

Powering Up The CPU

The 8085 CPU begins program execution at memory location 0000. The ROM code at that location jumps to 7D33, where, after a 100-millisecond delay, the PIO is initialized. (*See* chapter 5 for a discussion of the PIO.) If CTRL-BREAK was pressed during the reset or power-on, if the amount of installed RAM has changed, or if the RAM file directory entry for BASIC at F5F0 is missing, a "cold start" is performed complete with a purging of all RAM files.

Assuming the RAM directory has been found to be in order and no new RAM has been installed, the routine checks to see if it can reload the pointers that were in effect when the power-down occurred. The stack pointer, for instance, is retrieved from FABE.

The AC Adapter

The AC adapter, catalog number 26-3804, is not UL approved. However, it is similar in design and construction to many UL-approved adapters, so there is no safety risk. Inside the adapter is a transformer, rectifier, and capacitor, as shown in figure 16.2.

The transformer has a 120-volt primary winding and a nominal 5.6 volt secondary winding rated at 400 milliamperes. The alternating current output is connected to a one-piece full-wave rectifier, which produces direct current, but with a substantial AC ripple. The DC current is smoothed by a 2200-microfarad, 10-volt electrolytic capacitor and provided to the Model 100 by a two-meter cable. The white-striped lead of the cable is the negative wire, which connects to the inner conductor of the round plug.

The plug is a 5.5-millimeter barrel plug, equivalent to Radio Shack catalog number 274-1551. The plug connects with the DC6V jack on the right side of the computer, internally designated CN9. Ferrite beads are installed on the internal wiring to CN9 to aid in suppression of radio frequency energy which might otherwise be transmitted into the house wiring through the AC adapter or into the air by the adapter cord acting as an antenna. Though the adapter is rated 6 volts at 400 milliamperes, the actual no-load output is about 8 volts. This drops to about 7 volts under a typical Model 100 load of 50 milliamperes or to about 6.5 volts under a heavy load of 150 milliamperes.

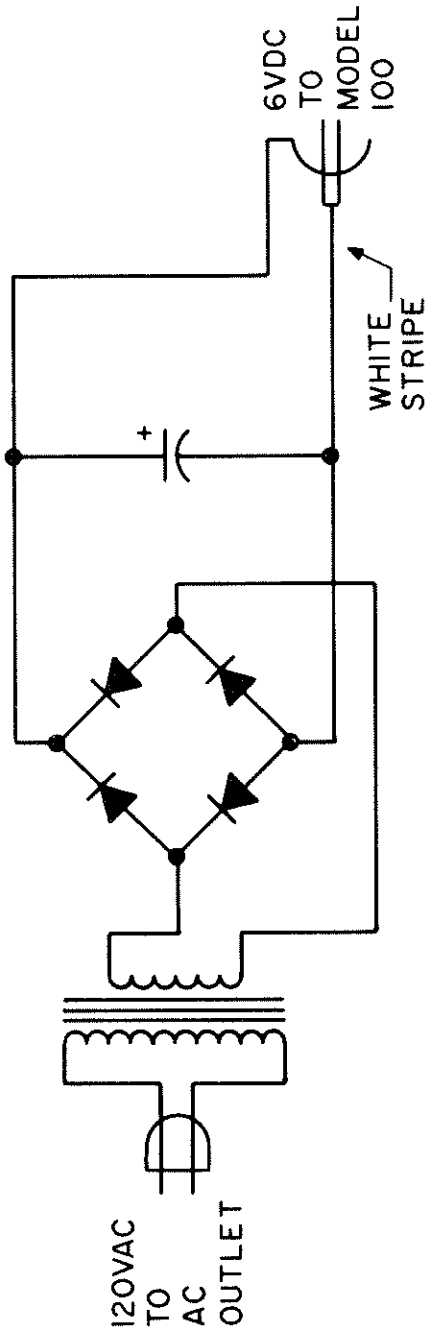

Figure 16.2. AC adapter

Since most AC adapters are composed simply of a transformer and rectifier without a capacitor, they do not perform any filtering. This explains why the Model 100 user's manual warns against using any adapter other than the Radio Shack adapter.

You could use a non-Radio Shack AC adapter which provides around 6 volts or so at 180 milliamperes, as long as capacitors are included which total 2200 microfarads. These may be spliced into the cord running from the adapter to the computer.

Alternative Power Supply

In powering the Model 100, you are not limited simply to AA cells and the AC adapter. Any ripple-free source of DC between 3.8 and 7 volts and capable of providing at least 1.1 watts can be plugged into the DC6V jack using a barrel plug (Radio Shack catalog number 274-1551). Sensible choices include a six-volt lantern battery with screw terminals or three photocell arrays wired in parallel. Just remember that the negative lead must be wired to the center terminal of the barrel plug.

17

Expansions

This chapter discusses a number of topics related to expansion of the Model 100. Hardware modifications or expansions would be required to implement each of these expansions.

The Bar-Code Reader and CRT

The manner in which BASIC handles the opening of devices makes it clear that device specifications WAND and CRT will be available for the Model 100. When an OPEN is performed in BASIC, a file control block is set up with pointers to addresses that BASIC uses in performing output (PUT), input (GET), and file CLOSE, as well as any special handling required by that device. The routine addresses are stored in ROM in various locations and are listed in table 17.1.

Some devices cannot be used for certain directions of data flow. For example, input cannot be performed from the LCD, CRT, or LPT. Note that these table positions are empty.

In the case of the WAND and CRT device names, the ROM entries refer to RAM addresses that are shown here in table 17.1 in parentheses. For example, if a CRT OPEN is attempted, BASIC jumps to the address stored in FB1A.

The original ROM loads these RAM locations with a jump that results in a return to BASIC with a ?FC error. When expansion software is loaded, these locations in RAM are changed to jump addresses for the appropriate handlers.

Table 17.1. Subroutine addresses for BASIC device handling

Device	OPEN	CLOSE	PUT	GET	Special
LCD:	14D8	4D59	14E5		
CRT:	(FB1A)	4D59	(FB1E)		
CAS:	1689	16AD	16C7	16D2	1710
COM:	176D	179E	17A8	17B0	17CA
WAND:	(FB20)	(FB22)	?FC	(FB24)	(FB26)
LPT:	14D8	4D59	175A		55CD
MDM:	176C	17DB	17A8	17B0	17CA
RAM:	1506	158D	15AC	15C4	161B

The addresses in table 17.1 may be printed out by means of the BASIC program listed below.

```
5 OPEN"device"FOROUTPUTAS1
10 HX$="0123456789ABCDEF":AD=20721:FOR D= 0 TO 7:
GOSUB 1000: TA=PEEK(A)+256*PEEK(A+1):
FOR IT=TA TO TA+8 STEP 2:A=IT:GOSUB2000:
NEXT IT:PRINT#1,:NEXT D:CLOSE:END
1000 PRINT#1,CHR$(PEEK(AD));:AD=AD+1:
IFPEEK(AD)<128 THEN 1000 ELSE PRINT#1,":"+CHR$(9);:
A=20755+(2*(255-PEEK(AD))): AD=AD+1:RETURN
2000 PRINT#1,CHR$(9);:A=A+1:GOSUB 3000:A=A-1:GOSUB3000:RETURN
3000 PRINT#1,MID$(HX$,1+((PEEK(A)AND240)/16),1);
MID$(HX$,1+(PEEK(A)AND15),1);:RETURN
```


Unused Pins

Two pins in the Model 100 are available for hobby usage. The first is an input port signal at M23, pin 2. This pin, presently wired only to a pullup resistor, controls bit 6 of input port 208. If a switch were connected from this pin to ground, software could determine the position of the switch by ANDing the value at port 208 with 64. If the result is zero, then the switch is closed.

The other unused pin is M16, pin 14. M16 is the integrated circuit that controls I/O ports 128 to 255. Pin 14 is usually at 5 volts, but drops to 0 whenever I/O ports 144 to 159 are accessed by the CPU. This signal could be fed to other CMOS integrated circuits or, with suitable buffering, could be used to control devices outside of the Model 100.

DISK INPUT/OUTPUT

The Model 100 designers have planned for disk or other bulk storage. The BASIC key word table contains DSKI$ and DSKO$; these words are handled at 5073 and 5071, respectively. Those ROM addresses contain references to RAM vectors at FB2E and FB30; which currently point to a ROM routine that yields a ?FC error. When DSKI$ and DSKO$ are implemented, these values change.

ROM ROUTINES FOR BULK DATA TRANSFER

The routine at 7304 through 7326 makes reference to I/O ports 70 through 73, which are not presently inplemented, but could easily be wired up at the expansion connector. This routine resembles a disk input routine, as it loads three numbers to output ports (drive number, track, and sector) and retrieves a specified number of bytes from an input port.

The routine at 767D through 770A appears to be a bulk output routine. It refers to ports 80, 81, 82, and 83, which are not presently included in hardware but which could easily be added at the expansion connector.

RECORD I/O

It appears that some sort of record I/O is being planned for the Model 100. Two variables, LOF and LOC, are provided for. In most

BASIC versions, LOF carries the number of logical records in a file, while LOC carries the present logical record number within an open file, a portion of which has already been input.

The RAM vector for LOF is located at FB28, while the vector for LOC is located at FB2A.

LISTING FILES TO THE PRINTER

The command LFILES is provided for, which you would expect to cause a listing of files at the printer. The RAM vector is at FB2C.

FUNCTION KEYS IN TELCOM TERM MODE

The F6 and F7 function keys in Term mode presently do nothing. This is because the RAM vectors at FB0C and FB0E simply point to RETurns. If you want to put one to use, just place something else in the RAM vector location. As a demonstration, in BASIC type:

```
POKE 64268,41:POKE 64269,66
```

Whenever the F6 key is pressed, the computer jumps to the routine at the location 41+256*66, which is the BEEP routine.

Many other RAM vectors exist. They are located from FADA to FB39 and are referred to by RST 7's. The byte following the RST 7 is used as an index into the table starting at FADA. Thus, RST 7 followed by 4E (which is what happens at the LOF routine at 506B) causes a reference to FADA+4E, or FB28. The address at FB28 and FB29 is jumped to by the RST 7 routine.

During a cold start, ROM loads the lower portion (up to FB12) of the FADA table with 7FF3, which is a RET. ROM loads the rest of the table (up to FB38) with 08DB, which was chosen because it returns to BASIC with an error 5, ?FC.

Understanding the Option ROM Socket

Selection of the option ROM is accomplished by turning on bit 0 of output port E8, as shown in Figure 17.1. To do so properly, obtain the contents of E8, which are stored by ROM at FF45. OR with 01, and OUT the data to the port.

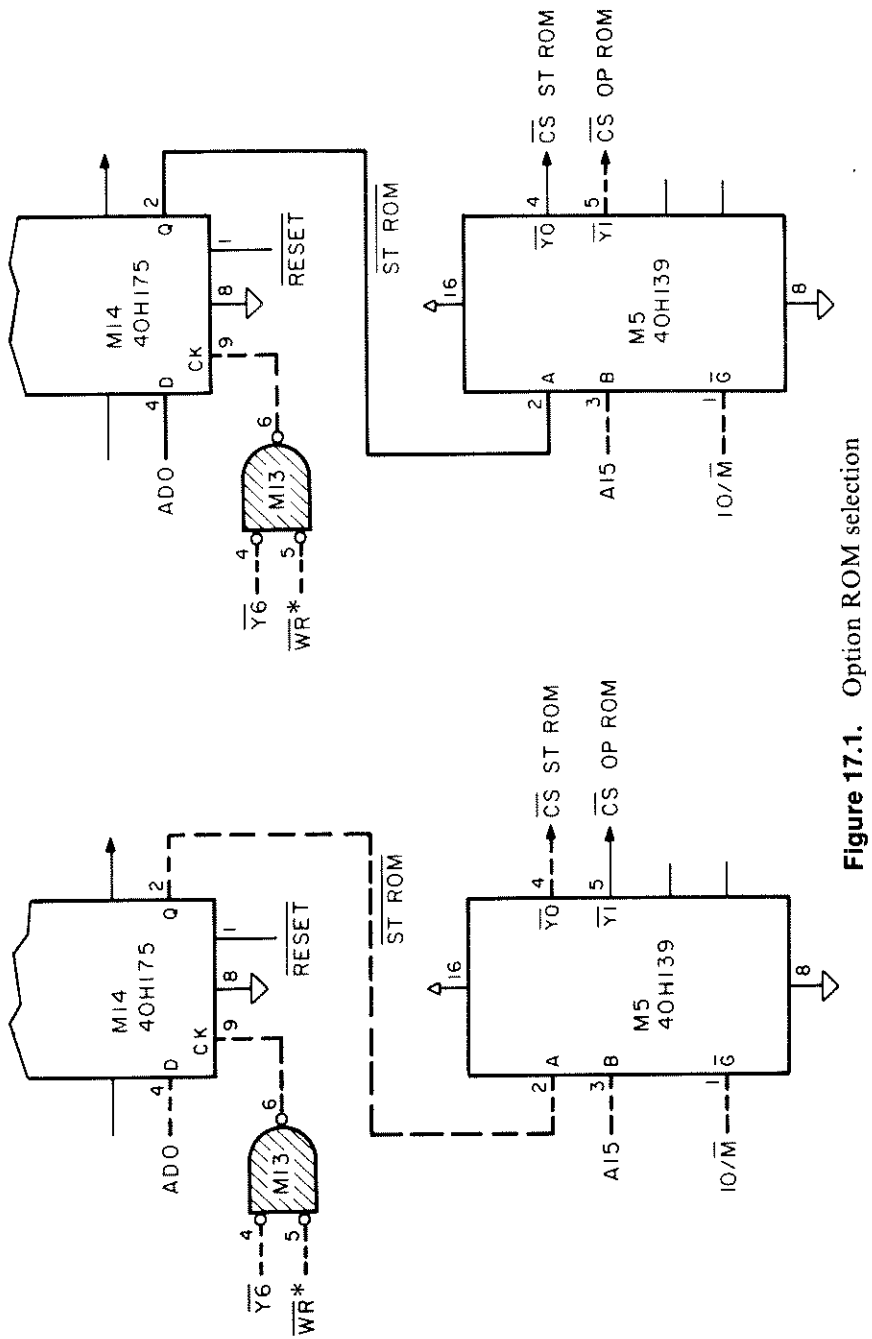

Figure 17.1. Option ROM selection

When a cold start takes place, the start routine checks the option ROM socket to determine the presence of a device (this occurs at 036F).

If location 40H in the option ROM contains 54 and location 41H contains 43, the ROM is assumed to be installed and functioning. A flag at F62A indicates whether the option ROM passed the test. The contents of ROM at 42H to 47H are treated as a filename. They are loaded into the RAM directory at F9BD through F9C2, immediately following SCHEDL. The filetype is specified to be F0 (valid entry, ASCII, machine language, ROM).

EXPANSION BUS SOCKET

The expansion bus is made available in a socket on the underside of the unit. The socket is a standard forty-pin integrated circuit socket, so an IC header can be used to make connections. The service manual states that an *optional I/O control unit* and *RAM file unit* can be connected to the expansion socket. As discussed previously, there is room for growth in the RAM vectors for DSKI$ and DSKO$ and in the ROM routines to nonexistent I/O ports between 70 and 8F.

The signals at the expansion socket are listed in table 17.2 and are discussed in turn.

Table 17.2. Expansion socket signals

Pin	Name	Input or Output
1	VDD	output
2	GND	output
3	AD0	input/output
4	AD2	input/output
5	AD4	input/output
6	AD6	input/output
7	A8	output
8	A10	output
9	A12	output
10	A14	output
11	GND	output
12	RD*	output
13	IO/M*	output
14	ALE*	output
15	CLK	output

Continued on following page

16	A*	output
17	INTR	input
18	GND	output
19	RAM RST	output
20	NC	
21	NC	
22	NC	
23	GND	
24	INTA	output
25	RESET*	output
26	Y0*	output
27	S1	output
28	S0	output
29	WR*	output
30	GND	output
31	A15	output
32	A13	output
33	A11	output
34	A9	output
35	AD7	input/output
36	AD5	input/output
37	AD3	input/output
38	AD1	input/output
39	GND	output
40	VDD	output

All but a few of the signals at the expansion connector are derived directly from the CPU, with only the usual buffering.

MEMORY ACCESS AT THE EXPANSION CONNECTOR

Since the Model 100 already contains devices which respond to every possible memory address, from 0000 to FFFF, the addition of a memory device via the expansion connector faces difficulties. As a result, unless you wish to cut traces on the printed circuit board, any addition of external memory could most easily be accomplished in port space rather than address space.

Nonetheless, memory addition is possible though difficult. Perhaps the easiest way would be to take advantage of the fact that the ROM from 0000 to 7FFF is already bank-switched. One could add circuitry putting 32K of RAM in 0000 to 7FFF whenever pin 27 (chip select not) of the option ROM socket was low. It would be necessary to copy into RAM those ROM routines that were needed for continued program execution.

It would appear the Model 100 designers have some sort of RAM expansion in mind since two reset signals are provided at the connector — the usual RESET* signal (pin 25) and also the RAM RST signal (pin 19). The service manual states the RAM RST signal is to be used with external CMOS RAM.

Address Decoding

All computers use signals like RD, WR, IO/M, address, and data for memory access. Memory address decoding in the Model 100 differs in one important respect, however, from that in most computers. Because the 8085 uses the same eight wires for data transfer and for address lines 0 through 7, any memory device added at the expansion connector requires address decoding circuitry which takes into account the ALE (address latch enable) signal present at pin 14. This signal determines whether the contents of the eight lines are to be interpreted by the other devices as address or data.

Other input/output status and control signals have been brought out to the connector. S0 and S1 are provided so that advance warning may be given to slow peripherals if a read or write is imminent. The signal designated A* is a combined RD* and WR* signal.

Adding Ports to the Model 100

Port space is a much more promising area of expansion than address space. For one thing, the port space is not yet filled; this is shown in figure 5.3. Also, because the port addresses are available on address lines 8 to 15, there is no need to worry about the ALE signal.

Finally, you can take advantage of circuitry in the Model 100 to decode port addresses. The port-select signal for ports 80 through 8F is already present at the connector — it is Y0* at pin 26. In addition, the port-select signal for ports 90 through 9F is inside the Model 100 waiting to be used. It can be found at pin 14 of M16.

Parallel Port Input

A typical general-purpose parallel input circuit is shown in figure 17.2. Using the configuration shown in the figure, the positions of the eight switches are available to the CPU at any input port in the range of 80 to 8F.

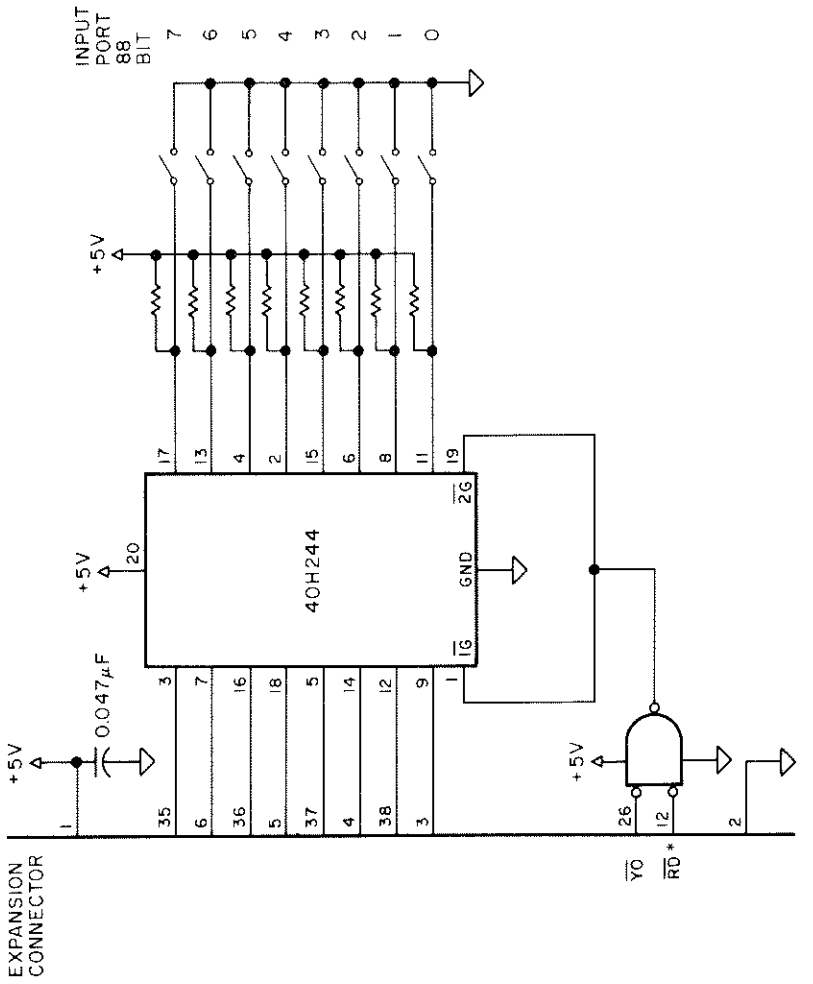

Figure 17.2. Parallel port input

Parallel Port Output

Parallel output is most easily accomplished using flip-flops, as shown in figure 17.3. An output to any port in the range of 80 to 8F will load the byte in the accumulator into the flip-flops and will turn on the LEDs. A relay is shown which is turned on if the corresponding bit (bit 0) is a one.

Interrupts at the Connector

The signals INTR and INTA, used for interrupt-driven programming, are available at the connector. If a device at the connector asserts INTR by pulling it up to +5 volts, the CPU responds to the interrupt. The response can be directed by jamming an interrupt vector address onto the address/data lines. This is not a technique for the idle experimenter. For more information, refer to the references cited in chapter 15 on interrupts.

THE TELEPHONE RING PULSE INPUT

One intriguing circuit is that connected to pin 8 of the phone jack CN4, labeled ring pulse. This pin, if grounded, turns off bit 5 of input port D8. The easiest way to ground it is to short it to pin 2 of that connector. A simple hardware device, like that shown in Figure 17.4, can be connected to the direct-connect phone cable (catalog number 26-1410) and to the Model 100.

THEORY OF OPERATION

When the phone line is quiet, direct current (DC) is present on the line without added ring-detect circuitry from the line by blocking the DC.

When the central office sends the ring signal, the capacitor couples the alternating current to diodes D1 and D2 which charges capacitor C1 to about 200 volts. This voltage, limited by resistor R2, allows a flow of about 20 milliamperes through the LED of optoisolator IC1, which turns on its phototransistor. Thus, during the ring pulse, Model 100 phone jack pin 8 is grounded through IC1 to phone jack pin 2.

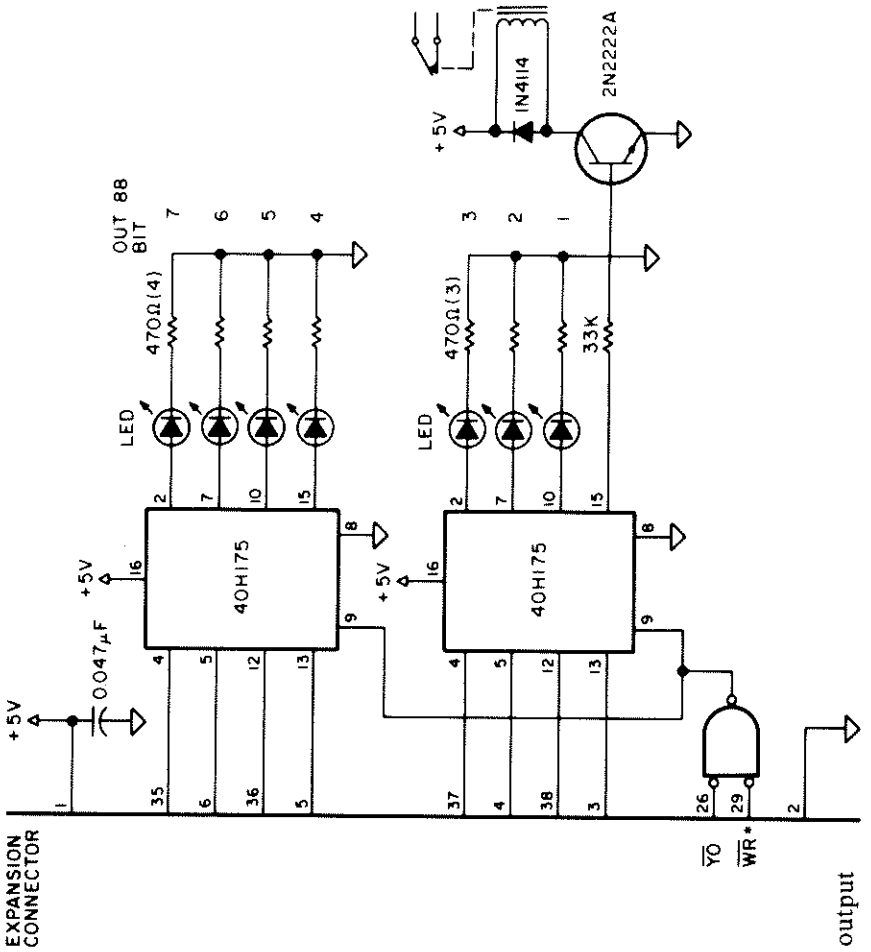

Figure 17.3. Parallel port output

The central office ring pulses occur about every six seconds and last about a second. When each pulse has ceased, capacitor C1 discharges quickly through the LED, turning off the LED and allowing the voltage at phone jack pin 8 to float back to its high level.

A parts list for a ring/detect circuit is provided in table 17.3. Any construction technique can be used. Lead lengths and placement are not critical, except that the wiring of pins 5 and 6 of the optoisolator should be carefully segregated from everything else. This author wired the circuit directly to the 8-pin plug of the modem cord. It would also be possible to equip the circuit with male and female 8-pin connectors to connect the computer and the modem cord. Testing the circuit is easy using the following simple BASIC program:

```
1 IF (INP(208) and 32)=0 THEN BEEP
2 GOTO 1
```

When the phone rings, the Model 100 beeps. It is an easy matter to write software in BASIC or machine language to take advantage of the RP signal. First, the CPU selects the modem mode through bit 3 of output port BA. Then it checks that the switches are set to the DIR and ANS positions through bits 4 and 5 of input port BB. After setting the baud rate and word length through a BASIC OPEN MDM: command or through the machine-language routines given in chapter 8, the CPU monitors input port D8 and answers the incoming call. This is done by way of bit 7 of output port BA when bit 5 changes to zero.

The program then interacts with the caller. When the caller hangs up, the incoming carrier would be lost. This would be signaled to the CPU at bit 0 of input port D8, and the CPU disconnects the call.

Figure 17.3. Schematic of a ring/detect circuit

Design	Catalog No.	Price	Description
C1,C2	272-1053	.59	0.1 uf 250V capacitor
D1,D2	276-1103	2/.69	1A 400V diode
IC1	276-1654	5/1.98	optoisolator (any of the five types will do)
R1	271-027	.19	2.2K resistor
R2	271-034	.19	10K resistor

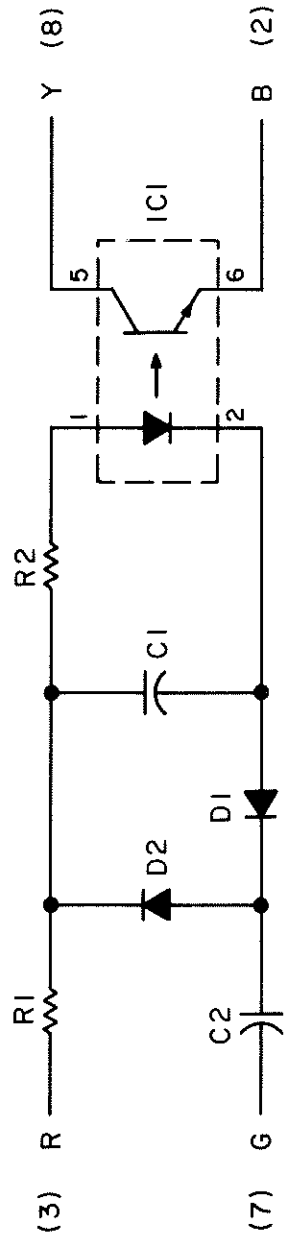

Figure 17.4. Ring/detect circuit

18

The Remainder of ROM

Although a number of published ROM subroutines have been discussed in earlier chapters, several routines did not fit into those categories. These are discussed here. In addition, other aspects of the ROM operating system are described in this chapter.

Published ROM Initialization Routines

MENU, called at 5797, is not really a subroutine, but rather an entry point for a fundamental ROM process from which there is no return. It goes to the main menu. Jumping to it is preferable to calling it.

The routine IOINIT, start address 6CE0, initializes values at the I/O ports and loads a disk bootstrap loader if a disk is attached. After

the loader is placed at E000 to E0FF, the routine jumps to E000. Assuming no disk is installed, the routine returns to the calling program.

The routine INITIO, called at 6CD6, zeros memory from FF40 to FFFD. This initializes a number of flags, such as XON/XOFF status, SOUND, baud rate, and the like. Afterwards the routine performs as IOINIT does. Neither of these initialization routines destroys user files.

PUBLISHED RAM FILE-HANDLING ROUTINES

User files reside in RAM starting at the lowest installed RAM address, with BA files at the bottom, followed by DO files and CO files. The directory is located in high memory from F962 to FA74. There is room to store sixteen filenames, with eleven bytes of information each. The format of the filename information is given in table 18.1.

Table 18.1. RAM Directory Format

Byte	Description
0	Directory flag*
1-2	File start address
3-10	Eight-byte file name

Table 18.2. RAM directory flag information

Bit	Description
7	0 if a killed file
6	1 if a DO file
5	1 if a CO file
4	1 if located in ROM
3	1 for invisible file
2-1	Reserved
0	Internal use only

Notice that no entry exists for filesize. To determine filesize, either of two techniques may be used. By comparing the various start addresses, one can determine the next file's starting address. The two start addresses may simply be subtracted.

* The directory flag contains the information described in table 18.2.

However, the filenames in the directory are not in order by start address, so one must search the entire directory to find the next file up. The ending address of the top file is stored at FBB2+.

Another way to determine file size is by examining the files themselves, since the file itself always contains enough information to determine its size. With BA and DO files, you must scan the file, beginning at the start address, to locate the end-of-file marker. For DO files, the EOF character is 26H. For BA files, it is a three character sequence, 00H.

For CO files, there is no end-of-file character as such, but the length of the file is determined by adding the value of the third byte to the product of 256 times the fourth byte plus six.

Suzuki and Hayash

Sooner or later you will stumble across the names Suzuki and Hayashi in the directory. Hayashi is the PASTE buffer. Suzuki is the location of the BASIC program, if any, that has not yet been SAVE'd and therefore does not have a name. (It is the file you sometimes see listed as BASIC*.) Sometimes you select BASIC from the main menu and find that some lines are still present from the last time you were in BASIC; these lines were hiding in Suzuki.

Suzuki and Hayashi take up memory, of course, like any other file. It can be emptied by entering BASIC and typing NEW.

DO Files

A DO file's first character is that which would be returned if you accessed it with TEXT or with BASIC's OPEN and INPUT statements. The file ends with the 26H described above.

BA File Format

A BA file in RAM is set up as a sequence of lines, each with the following form:

- 2 bytes — Hex address of line number to follow
- 2 bytes — Current line number (hex)
- numerous bytes — Program line in tokenized form
- 1 byte — Null (00H)

The file ends with two more nulls (00H), making the last three characters nulls.

CO File Format

The format of a CO file is as follows:

- 2 bytes — Address to load file to
- 2 bytes — Number of bytes to load (six fewer than the size of the RAM file, because of these six address bytes)
- 2 bytes — Transfer (start) address
- many bytes — Contents of file

ACCESSING FILES

BASIC's PEEK statement can be used to examine file contents without the danger of altering the file. The use of POKEs with files, however, is to be discouraged unless you know precisely what you are doing. If you must POKE, keep the following in mind.

- POKE only within the files, not elsewhere in RAM. In particular, do not POKE to addresses at or above 62960 (F5F0).
- If you create an end-of-file (EOF) marker within a DO file, you must delete any characters from there up to the beginning of the next file. Use MASDEL, described below.
- If, within a BA file, you tamper with any of the file format addresses described above in memory, such as the BASIC address of the following line number, the program will no longer function properly.

Published ROM Calls for Manipulating RAM Files

Most of the following routines can be used only to manipulate DO files in RAM.

The routine MAKTXT, called at 220F, creates a DO file in the RAM directory with the name specified in the buffer at FC93 through FC98. If the file already exists, the routine returns with the carry flag set.

Upon exiting the routine, the HL register points to the TOP address of the new file (the address at which new characters would be added), and DE points to the address of the directory file flag, located somewhere in the directory starting at F962.

The routine CHKDC, called at 5AA9, examines the directory to determine if a specified DO filename is present in the directory as a valid file. Prior to the call, DE should point to the first in a series of memory addresses (RAM or ROM) containing a filename in ASCII followed by a null.

Upon return the Z flag is set if no such file was found. If the file was found, HL points to the start, or lowest, address of the file.

The routine GTXTTB, called at 5AE3, finds the TOP address of the file assuming its location in the directory is known. If you provide the address of the directory entry in HL, it returns the TOP address of the file.

The routine KILASC, called at 1FBE, kills a DO file. Prior to the call, DE must be set to the starting address of the file, and HL must be set to the address of the directory entry. Everything above it in user memory (CO files and other DO files) is moved down to fill in the space.

The routine INSCHR, called at 6B61, inserts one character in a DO file. Prior to the call, HL points to the address at which to insert the character, and A contains the character to be inserted. Everything above in user area (all CO files and some DO files) is moved up one position.

If there is no more room in RAM, the routine returns with the carry flag set.

The routine MAKHOL, called at 6B6D, inserts a specified number of spaces in a file. Everything is moved up in memory to accommodate the spaces. These spaces may later be changed as desired. Prior to the call, BC must contain the number of spaces to be inserted, and HL must point to the address at which to insert the spaces. HL and BC are preserved, which is handy. If there was insufficient room in RAM, the routine returns with the carry flag set.

The routine MASDEL, called at 6B9F, deletes a specified number of characters from a file. Prior to the call, BC must contain the number

of characters to be deleted, and HL should point to the address at which to begin deleting. HL and BC are preserved.

Block Moves

The Model 100 ROM contains a number of block move routines. In each case, a source address is loaded to a register pair, a destination address is loaded to another register pair, and the number of bytes to be transfered is loaded to another register (for a transfer of up to 256 bytes) or to another register pair (for a transfer of up to 65536 bytes). At this point, the routine can either load and increment or load and decrement, as shown in table 18.3.

Table 18.3. Block move subroutines

Source	Destination	# of Bytes Bytes	Increment or Decrement	Call Address
BC	DE	L	Increment	290F
DE	HL	A	Increment	5A62
DE	HL	B	Increment	3469
DE	HL	B	Decrement	3472
HL	DE	B	Increment	2542
HL	DE	BC	Decrement	6BE6
HL	DE	BC	Increment	6BDB
HL	DE	C	Decrement	2EE6
HL	DE	till 0 is loaded	Increment	65C3

One routine, at 65C3, is different in that it loads until such time as a null character is encountered in the source location.

Lowercase Conversion

The routine at 0FE8 converts lowercase to uppercase. Prior to the call, HL must point to a character. After the call, the accumulator contains that character, converted to uppercase.

CONVERTING NUMERICAL HEX TO ASCII

The routine at 1999 converts the numerical hex byte pointed to by DE to ASCII, with the result in HL. Then DE and HL are incremented. (All the routine does is OR the contents with 30H.) No error checking is undertaken. The value at DE must be in the range of 0 to 9.

CONVERTING TWO NUMERICAL HEX BYTES TO ASCII

The routine at 1996 causes the previous routine to be executed twice. This is useful for converting seconds from the clock/calendar.

REGISTER-PAIR COMPARISON

The subroutine called by RST 3 (RST 18) compares DE to HL. If they are equal, the zero flag will be set.

UTILITY FOR COMMAND DECODING

The subroutine at 6CA9 is a useful general-purpose command decoder. Prior to the call, DE points to a command table containing four-letter commands and jump addresses. HL points to a command received as user input. The command is converted to upper case and is compared one-by-one with the commands in the table. If there is a match, the routine jumps to the address in the table associated with that command. If there is no match, the routine returns with the Z flag set.

The table is composed of one or more command entries, followed by a null. Each command entry is composed of four letters and a two-byte jump address.

RAM VARIABLE MAP

A variety of variables are located in high-end RAM. They are listed in table 18.4. Many of these may be changed by a user program as needed.

Table 18.4. RAM variables

Address	Description	Initialized Where
F5F0	Start of system	035A
F5F2	SP upon power-down	035C
F5F4+	HIMEM value	035E
F5F6	Power-on hook	0360
F5F9	BCR interrupt hook	0363
F5FC	UART DR hook	0366
F5FF	TP interrupt hook	0369
F602	LPS interrupt hook	036C

Continued on following page

F605	Power-on hook	036F
F60F	Option ROM hook	0379
F62A	Option ROM installed	0394
F62B	20 or 10 pps	
F639	Cursor row	03A3
F63A	LCD cursor column	03A4
F63B	LCD active lines	03A5
F63C	LCD active columns	03A6
F63D	Label line flags	03A7
F648	Reverse video if 1	03B2
F64E	X-pixel set	03B8
F64F	Y-pixel set	03B9
F657	Power-down value	03C1
F65B-F660	Initial Stat setting	03C5
F674	LPOS printer column	03DE
F675	Output 0=LCD 1=LP	03DF
F678	Top of available RAM	03E2
F685	Keyboard buffer	
F788	POS- cursor position	
F789	Key labels	
F88C+	Location of PASTE buffer	
F923	Seconds— units	
F924	Seconds— tens	
F925	Minutes— units	
F926	Minutes— tens	
F927	Hours— units	
F928	Hours— tens	
F929	Day of month-units	
F92A	Day of month-tens	
F92B	Day of week	
F92C	Month	
F92D	Year-units	
F92E	Year-tens	
F962-F9B9	Directory 6BF1+	
F9BA-F9C4	Option ROM or filname	
F9C5-FA74	Directory	
FAAC	Character recently sent	
FAAE	Port A8 contents	
FAC0	Lowest installed RAM	
FADA-FB14	RST 38 hooks-all RET	
FB16-FB39	RST 38 hooks-all error 5	
FBB0+	Top of DO files	
FBB2+	Top of CO files	
FC18	FLoating Point Accumulator 1	
FC69	Floating Point Accumulator 2	
FC82	Maxfiles?	
FC93-FC98	Filename	

Continued on following page

FE00-FF3F	LCD memory	
FF40	Distant computer sent XOFF	
FF41	This Model 100 sent XOFF	
FF42	Is XOFF enabled?	
FF44	SOUND OFF=1 ON=0	
FF45	Port E8 contents	
FF8B	Address of baud rate	

Appendix A. Nonprintable Characters and Model 100 Assignments

The following table indicates the meanings that other devices may give to nonprintable ASCII characters. It also shows what meaning, if any, the Model 100 assigns to these characters.

Decimal	ASCII Meaning	Model 100 Meaning
0	Null	
1	SOH-Start of Heading	
2	STX-Start of Text	
3	ETX-End of Text	
4	EOT-End of Transmission	
5	ENQ-Enquiry	
6	ACK-Acknowledge	
7	BEL-Bell	produces BEEP
8	BS-Backspace	moves cursor one space to left
9	HT-horizontal tab	moves cursor to next 8th column
10	LF-line feed	moves cursor one line down (no horizontal movement)
11	VT-vertical tab	cursor to home
12	FF-form feed	clears screen
13	CR-carriage return	cursor to left edge
14	SO-shift out	
15	SI-shift in	
16	DLE-data link escape	
17	DC1-device control 1	if XON/XOFF enabled, continue transmission
18	DC2-device control 2	
19	DC3-device control 3	if XON/XOFF enabled, pause in transmission
20	DC4-device control 4	
21	NAK-negative acknowledge	
22	SYN-synchronous idle	
23	ETB-end of transmission block	
24	CAN-cancel	
25	EM-end of medium	
26	SUB-substitute	
27	ESC-escape	(*see* table 13.4)

28	FS-file separator	
29	GS-group separator	
30	RS-record separator	
31	US-unit separator	
127	DEL-delete	same as backspace

Appendix B.
ROM Map

This map will be of help to those who are disassembling ROM routines.

Address	Function
0000	hardware reset address
0008	restart 08
0010	restart 10
0018	restart 18-compares HL and DE
0020	restart 20-character to LCD or LPT
0024	low-power signal routine
0028	restart 28-check variable type
002C	bar code reader routine
0030	restart 30-sign of floating point number
0034	UART data ready routine
0038	restart 38-ram hooks
003C	clock/calendar pulse routine
0040 to 007F	address table for BASIC functions SGN to MID$, keywords of which appear at 01F0 to 025F
0080 to 018E	BASIC command keywords END to NEW, with address table starting at 0262
018F to 01D5	BASIC function keywords TAB to STEP
01D6 to 01EF	BASIC operator keywords "+" to "<", with address table starting at 02F8
01F0 to 025F	BASIC function keywords SGN to MID$, with address table starting at 0040
0262 to 02E1	address table for BASIC command keywords END to NEW, located at 0080 to 018E
02EE to 031B	address table for BASIC operators, used at 10DA
031C to 0359	BASIC two-character error messages
035A to 03E9	initialization RAM image loaded to F5F0-F76F
035C	initialization value for SP upon power-down
035E	initialization value for HIMEM
0360	initialization value for power-on hook
0363	initialization value for bar code reader interrupt hook
0366	initialization value for UART data ready interrupt hook
0369	initialization value for clock/calendar interrupt hook

Address	Function
036C	initialization value for low power signal interrupt hook
036F	power-on routine— option ROM testing
0394	initialization value for option ROM presence flag
03A3 to 03A6	initialization values for screen position and size parameters
03A7	initialization value for label line flag
03B2	initialization value for reverse video flag
03C1	initialization value for power-down countdown
03C5	initialization value for Stat
03DE	initialization value for LPOS
03DF	initialization value for output flag
04DD	prints error message based on A
0726	BASIC command FOR
076B	BASIC command TO
0783	BASIC command STEP
0840	BASIC command dispatcher based on token in A
0858	sets pointer to BASIC text
0872	BASIC command DEF
0881	BASIC command DEFDBL
0886	BASIC command DEFINT
0896	BASIC command DEFSNG
089F	BASIC command DEFSTR
08DB	FC error
08EB	convert ASCII decimal to hex in DE
090F	BASIC command RUN
091E	BASIC command GOSUB
0936	BASIC command GOTO
0966	BASIC command RETURN
099E	BASIC command DATA
09A0	BASIC command ELSE, REM
09C3	BASIC command LET
0A2F	BASIC command ON
0A34	BASIC command ON ERROR
0AB0	BASIC command RESUME
0B0F	BASIC command ERROR
0B1A	BASIC command IF
0B4E	BASIC command LPRINT
0B56	BASIC command PRINT
0C01	BASIC command TAB(
0C45	BASIC command LINE
0C50	BASIC command LINE INPUT
0C74	"Redo from start"
0C99	BASIC command INPUT#

Address	Function
0CA3	BASIC command INPUT
0CD9	BASIC command READ
0D71	BASIC command "Extra ignored"
0F47	BASIC function ERR
0F56	BASIC function ERL
0F7E	BASIC function VARPTR
0FE8	converts LC to UC
1054	BASIC operator NOT
108C	BASIC operator OR
1097	BASIC operator AND
10A2	BASIC operator XOR
10AD	BASIC operator EQV
10B5	BASIC operator IMP
10C8	BASIC function LPOS
10CE	BASIC function POS
1100	BASIC function INP
110C	BASIC command OUT
112E	convert ASCII to integer
1138	BASIC command LLIST
1140	BASIC command LIST
11A2	print from buffer till zero byte reached
11AA	put data in buffer until zero byte reached
1284	BASIC function PEEK
128B	BASIC command POKE
12CB	wait for character from keyboard
12F0	paste routine
13A5	toggle label line
13DB	check keybaord queue for characters
1412	break routine
1419	POWER routine
1431	power down sequence
1451	POWER OFF
1459	BASIC command POWER CONT
1469	set power down value
1470	print character without expanding ASCII 09
148A	read cassette header and synch byte
14A8	motor on
14AA	motor off
14B0	read a character from cassette
14C1	send character to cassette and update checksum
14D2	LCD device control block
14D8	LCD file open
14E5	LCD file put
14F2	CRT device control block
14FC	RAM device control block

Address	Function
1506	RAM file open
158D	RAM file close
167F	CAS device control block
1689	CAS file open
16AD	CAS file close
16C7	CAS file put
16D2	CAS file get
1754	LPT device control block
175A	LPT file put
1762	COM device control block
176C	MDM file open
176D	COM file open
179E	COM file close
17A8	COM and MDM file put
17B0	COM and MDM file get
17D1	MDM device control block
17DB	MDM file close
17E6	sets serial interface per Stat
1877	WAND device control block
1889	BASIC function EOF
1904	BASIC function TIME$
190F	get time string to HL
1924	BASIC function DATE$
192F	get date string to HL
1955	BASIC function DAY$
1962	get day string to HL
1978	"SunMonTueWedThuFriSat"
1999	convert hex to digit
19A0	loads clock/calendar into RAM
19AB	BASIC command TIME$=
19BD	BASIC command DATE$=
19F1	BASIC command DAY$=
19FA	BASIC command MAXRAM
1A78	BASIC command IPL
1A9E	BASIC command MDM, COM
1AB2	BASIC command KEY(
1B0F	BASIC command ON TIME$
1B32	clock/calendar pulse handler
1BB8	BASIC keyword KEY
1BBD	BASIC command KEY LIST
1BE0	prints memory to screen filtering
1C57	BASIC command PSET
1C66	BASIC command PRESET
1D90	BASIC function CSRLIN
1D9B	BASIC keyword MAX
1DB2	BASIC command MAXFILES

Address	Function
1DB9	BASIC keyword HIMEM
1DC3	BASIC command WIDTH
1DC5	BASIC command SOUND
1DE5	BASIC command SOUND OFF
1DE6	BASIC command SOUND ON
1DEC	BASIC command MOTOR
1DFA	BASIC command CALL
1E22	BASIC command SCREEN
1E5E	BASIC command LCOPY
1E5E	screen dump routine
1F3A	BASIC command FILES
1F91	BASIC command KILL
1FBE	BASIC command KILL for DO file
2037	BASIC command NAME
20FE	BASIC command NEW
220F	opens RAM DO file
2280	BASIC command CSAVE
22B9	block move to tape
22CC	CSAVEM
22DD	BASIC command CSAVEM
2377	BASIC command CLOAD
2413	block move from tape
2456	CLOAD?
2491	BASIC command LOADM
2491	BASIC command RUNM
24A7	BASIC command CLOADM
2542	block move of B bytes from HL to DE increasing
2573	CLOADM?
25D5	"Top: "
25DB	"End: "
25E1	"Exe: "
260B	open for output CAS:.BA
260E	open for output CAS:.DO
2611	open for output CAS:.CO
2650	open for input CAS:.BA
2653	open for input CAS:.DO
2656	open for input CAS:.CO
26FE	"Found:"
2705	"Skip:"
273A	BASIC function STR$
27B1	prints a line to screen
28CC	string concatenation
290C	block move bytes from BC to DE increasing
2943	BASIC function LEN
294F	BASIC function ASC
295F	BASIC function CHR$
296D	BASIC function STRING$

Address	Function
298E	BASIC function SPACE$
29AB	BASIC function LEFT$
29D6	BASIC function RIGHT$
29E6	BASIC function MID$
2A07	BASIC function VAL
2A37	BASIC function INSTR
2B4C	BASIC function FRE
2B69	BASIC operation double precision subtraction
2B78	BASIC operation double precision addition
2CFF	BASIC operation double precision multiplication
2DC7	BASIC operation double precision division
2EE6	block move
2EEF	BASIC function COS
2F09	BASIC function SIN
2F58	BASIC function TAN
2F71	BASIC function ATN
2FCF	BASIC function LOG
305A	BASIC function SQR
30A4	BASIC function EXP
313E	BASIC function RND
325C to 33DB	floating point constants
33DC	restart 30 routine
33F2	BASIC function ABS
3407	SGN
3469	block move B bytes from DE to HL increasing
3472	block move B bytes from DE to HL decreasing
3498	BASIC operator single precision comparison
34C2	BASIC operator integer comparison
34FA	BASIC operator double precision comparison
3501	BASIC function CINT
352A	BASIC function CSNG
35BA	BASIC function CDBL
3645	BASIC function FIX
3654	BASIC function INT
36F8	BASIC operator integer subtraction
3704	BASIC operator integer addition
3725	BASIC operator integer multiplication
377E	BASIC operator integer division
37DF	BASIC operator MOD
37F4	BASIC operator single precision addition
37FD	BASIC operator single precision subtraction
3803	BASIC operator single precision multiplication
380E	BASIC operator single precision division
39D4	convert hex to integer and print
3D7F	BASIC operator single precision exponentiation

Address	Function
3D8E	BASIC operator double precision exponentiation
3DF7	BASIC operator integer exponentiation
3FA0	BASIC command TIME$ ON
3FB2	BASIC command TIME$ OFF
3FB9	BASIC command TIME$ STOP
407F	BASIC command RESTORE
409A	BASIC command STOP
409F	BASIC command END
40DA	BASIC command CONT
40F9	BASIC command CLEAR
4174	BASIC command NEXT
4222	CR and LF to LCD
4225	LF to LCD
4229	BEEP to LCD
422D	LCD cursor home
4231	LCD clear screen
4235	LCD set label line
423A	LCD unlock label line
423F	LCD disallow scrolling
4244	LCD allow scrolling
4249	LCD cursor on
424E	LCD cursor off
4253	LCD delete line at cursor
4258	LCD insert blank line at cursor
425D	LCD erase to end of line
4262	LCD escape X sequence
4269	LCD enter reverse character mode
426E	LCD exit reverse character mode
4270	LCD send escape sequence
427C	LCD set cursor position
428A	LCD erase function key display
42A5	LCD set and display function keys
42A8	LCD display function keys
438A to 43A1	lookup table for special ASCII LCD characters
43B8 to 43F9	lookup table for LCD escape sequences
4408	LCD TAB routine
4431	LCD ESC p routine
4433	LCD ESC q routine
4437	LCD ESC U routine
4438	LCD ESC T routine
443F	LCD ESC V routine
4441	LCD ESC W routine
444A	LCD ESC X routine
4453	LCD ESC C routine
445C	LCD ESC D routine
4461	LCD backspace routine

Address	Function
4469	LCD ESC A routine
446E	LCD ESC B routine
4480	LCD TAB routine
4494	LCD line feed routine
44A8	LCD vertical tab routine
44A8	LCD ESC H routine
44AF	LCD ESC P routine
44BA	LCD ESC Q routine
44C4	LCD ESC M routine
44EA	LCD ESC L routine
4535	LCD ESC I routine
4537	LCD ESC k routine
4548	LCD ESC J routine
4548	LCD ESC E routine
463E	prompt with ? and get line
4644	get line from keyboard
4684	BASIC control-C handler
4696	BASIC ENTER handler
46A0	BASIC backspace handler
46A0	BASIC leftarrow handler
46A0	BASIC CTRL-H handler
46C3	BASIC CTRL-U handler
46C3	BASIC CTRL-X handler
46CA	BASIC TAB handler
478B	BASIC command DIM
4991	BASIC command USING
4B44	character to LCD
4B55	put to printer expanding ASCII 09
4BA0	carriage return to printer
4BEA	BASIC function INKEY$
4C0F	filename string scan
4CCB	BASIC command OPEN
4D59	COM file close routine
4D70	BASIC command LOAD
4D71	BASIC command MERGE
4DCF	BASIC command SAVE
4E28	BASIC command CLOSE
4E8E	BASIC function INPUT$
4F0A	clear B bytes of memory at HL
4F0B	load A into B bytes of mnemory at HL
404E	bad file name
5051	already open
5054	direct statement in file
5057	file not found

Address	Function
505A	file not open
505D	bad file number
5060	undefined input error IE
5063	input past end
5066	undefined error FL
506B	BASIC function LOF
506D	BASIC function LOC
506F	BASIC command LFILES
5071	BASIC function DSKO$
5073	BASIC function DSKI$
50F1 to 5112	BASIC device control block name match table
5113 to 5124	address table for device control blocks
5146	TELCOM start location
51C0	TELCOM Stat function key
522F	TELCOM Call function key
524D	TELCOM Find function key
52BB	disconnect phone line
52D0	connect phone line
532D	autodialer routine
5455	TELCOM Term function key
5523	TELCOM Prev function key
553E	TELCOM Full/Half function key
5550	TELCOM Echo function key
559D	TELCOM Up function key
567E	TELCOM Down function key
571E	TELCOM Bye function key
5791	CR/LF, then display string at HL up to null
5797	go to main menu
57DF	display copyright notice
582E	display number of free bytes
5834	menu select handler
5970	list files on screen
59C9	move cursor across filenames
5A58	send string to screen
5A62	block move
5A79	clear function keys
5A7C	set function keys
5A9E	display function keys
5AA9	search for filename in directory
5AE3	get top address of file
5B3E	used to clear all function keys
5B46	normal BASIC function keys
5B68	ADDRSS routine
5B6F	SCHEDL routine

Address	Function
5BF5	ADDRSS/SCHEDL Find function key
5BF7	ADDRSS/SCHEDL Lfind function key
5D46	is character at HL a space?
5D6A	home cursor
5DEE	TEXT routine
5E51	BASIC command EDIT
6016 to 6055	TEXT CTRL-character address table
607C	TEXT CTRL-P (escape) handler
608A	TEXT TAB handler
60BE	TEXT carriage return handler
60DE	TEXT rightarrow handler
60E2	TEXT downarrow handler
610B	TEXT backspace handler
6118	TEXT delete handler
6151	TEXT leftarrow handler
6155	TEXT uparrow handler
617A	TEXT SHIFT-rightarrow handler
618C	TEXT SHIFT-leftarrow handler
61C2	TEXT SHIFT-uparrow handler
61D7	TEXT SHIFT-downarrow handler
61FD	TEXT CTRL-rightarrow handler
620B	TEXT CTRL-leftarrow handler
6210	TEXT CTRL-uparrow handler
621C	TEXT CTRL-downarrow handler
6242	TEXT Sel function key handler
628F	TEXT CTRL-C and SHIFT-BREAK handler
6431	TEXT Copy function key handler
6445	TEXT Cut function key handler
6551	TEXT Find function key handler
65C3	block move HL to DE until null
667C	TEXT tab handler
6691	TEXT SHIFT-PRINT handler
6713	TEXT Save function key handler
6774	TEXT Load function key handler
6AC3	HALT in TEXT- why?
6B61	insert character in RAM file
6B6D	insert spaces in RAM file
6B9F	delete characters from RAM file
6BDB	block move BC bytes from HL to DE increasing
6BE6	block move BC bytes from HL to DE decreasing
6BF1 to 6C48	initialization value for RAM directory
6C49	BASIC routine starts here

Address	Function
6CD6	cold start reset
6CE0	warm start reset
6CED	set 256 Hertz interrupt
6CFC	initialize disk drive
6D3F	put to LPT
6D6D	inspect incoming serial queue
6D7E	get incoming serial data
6DAC	handles UART data ready
6DBE	parity, overrun, framing error
6E0B	put CTRL-Q to UART
6E1E	put CTRL-S to UART
6E32	put byte to UART
6E75	set UART baud rate
6E94 to 6EA5	baud rate table
6EA6	set baud rate and modem
6ECB	deactivate UART
6EEF	carrier detect
6F31	enable CTL-S, CTL-Q protocol
6F32	disable CTL-S, CTL-Q protocol
6F46	cassette write header and synch byte
6F5B	put one byte to cassette
6F85	cassette read header and synch byte
6FDB	cassette bit-input routine
702A	cassette byte-input routine
7055 to 7241	keyboard scanning code
71E4	add a character to keyboard buffer
7242	get keyboard contents
7270	check keyboard for characters or SHIFT-BREAK
7283	check keyboard for SHIFT-BREAK or CTRL-C
729F	check keyboard for SHIFT-BREAK
72C5	BASIC command SOUND
7304 to 7326	disk input routine
7329	clock/calendar data loader
7391	clock/calendar timing pulse handler
7440	cursor set routine
744C	pixel set routine
744D	pixel reset routine
7551 to 7653	LCD location table
7662	BEEP routine
7676	toggle beeper
767D to 76DB	disk bootstrap loader
76DE	disk output routine
7711	LCD character generation table

Address	Function
7BF1 to 7D32	keyboard decoding table
7D33	RESET routine
7D43	erase all files if CTRL-BREAK pushed
7D6C	test for option ROM
7DE7	erase all files
7E24	load option ROM if present
7EAC	prints number of free bytes
7EC6	initialize RAM vectors
7FD6	restart 38 handler

Appendix C.
8080, 8085, & Z80 Opcodes

Decimal	Hex	8080 Opcode	Z80 Opcode
0	00	NOP	NOP
1	01 FF FF	LXI B,FFFF	LD BC,FFFF
2	02	STAX B	LD (BC),A
3	03	INX B	INC BC
4	04	INR B	INC B
5	05	DCR B	DEC B
6	06 FF	MVI B,FF	LD B,FF
7	07	RLC	RLCA
8	08	-Data-	-Data-
9	09	DAD B	ADD HL,BC
10	0A	LDAX B	LD A,(BC)
11	0B	DCX B	DEC BC
12	0C	INR C	INC C
13	0D	DCR C	DEC C
14	0E FF	MVI C,FF	LD C,FF
15	0F	RRC	RRCA
16	10	-Data-	-Data-
17	11 FF FF	LXI D,FFFF	LD DE,FFFF
18	12	STAX D	LD (DE),A
19	13	INX D	INC DE
20	14	INR D	INC D
21	15	DCR D	DEC D
22	16 FF	MVI D,FF	LD D,FF
23	17	RAL	RLA
24	18	-Data-	-Data-
25	19	DAD D	ADD HL,DE
26	1A	LDAX D	LD A,(DE)
27	1B	DCX D	DEC DE
28	1C	INR E	INC E
29	1D	DCR E	DEC E
30	1E FF	MVI E,FF	LD E,FF
31	1F	RAR	RRA

Table C.1. Numerical list of Opcodes

Decimal	Hex	8080 Opcode	Z80 Opcode
32	20	RIM	-----
33	21 FF FF	LXI H,FFFF	LD HL,FFFF
34	22 FF FF	SHLD FFFF	LD (FFFF),HL
35	23	INX H	INC HL
36	24	INR H	INC H
37	25	DCR H	DEC H
38	26 FF	MVI H,FF	LD H,FF
39	27	DAA	DAA
40	28	-Data-	-Data-
41	29	DAD HL	ADD HL,HL
42	2A FF FF	LHLD FFFF	LD HL,(FFFF)
43	2B	DCX H	DEC HL
44	2C	INR L	INC L
45	2D	DCR L	DEC L
46	2E FF	MVI L,FF	LD L,FF
47	2F	CMA	CPL
48	30	SIM	-----
49	31 FF FF	LXI SP,FFFF	LD SP,FFFF
50	32 FF FF	STA FFFF	LD (FFFF),A
51	33	INX SP	INC SP
52	34	INR M	INC (HL)
53	35	DCR M	DEC (HL)
54	36 FF	MVI M,FF	LD (HL),FF
55	37	STC	SCF
56	38	-Data-	-Data-
57	39	DAD SP	ADD HL,SP
58	3A FF FF	LDA FFFF	LD A,(FFFF)
59	3B	DCX SP	DEC SP
60	3C	INR A	INC A
61	3D	DCR A	DEC A
62	3E FF	MVI A,FF	LD A,FF
63	3F	CMC	CCF
64	40	MOV B,B	LD B,B
65	41	MOV B,C	LD B,C

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
66	42	MOV B,D	LD B,D
67	43	MOV B,E	LD B,E
68	44	MOV B,H	LD B,H
69	45	MOV B,L	LD B,L
70	46	MOV B,M	LD B,(HL)
71	47	MOV B,A	LD B,A
72	48	MOV C,B	LD C,B
73	49	MOV C,C	LD C,C
74	4A	MOV C,D	LD C,D
75	4B	MOV C,E	LD C,E
76	4C	MOV C,H	LD C,H
77	4D	MOV C,L	LD C,L
78	4E	MOV C,M	LD C,(HL)
79	4F	MOV C,A	LD C,A
80	50	MOV D,B	LD D,B
81	51	MOV D,C	LD D,C
82	52	MOV D,D	LD D,D
83	53	MOV D,E	LD D,E
84	54	MOV D,H	LD D,H
85	55	MOV D,L	LD D,L
86	56	MOV D,M	LD D,(HL)
87	57	MOV D,A	LD D,A
88	58	MOV E,B	LD E,B
89	59	MOV E,C	LD E,C
90	5A	MOV E,D	LD E,D
91	5B	MOV E,E	LD E,E
92	5C	MOV E,H	LD E,H
93	5D	MOV E,L	LD E,L
94	5E	MOV E,M	LD E,(HL)
95	5F	MOV E,A	LD E,A
96	60	MOV H,B	LD H,B
97	61	MOV H,C	LD H,C

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
98	62	MOV H,D	LD H,D
99	63	MOV H,E	LD H,E
100	64	MOV H,H	LD H,H
101	65	MOV H,L	LD H,L
102	66	MOV H,M	LD H,(HL)
103	67	MOV H,A	LD H,A
104	68	MOV L,B	LD L,B
105	69	MOV L,C	LD L,C
106	6A	MOV L,D	LD L,D
107	6B	MOV L,E	LD L,E
108	6C	MOV L,H	LD L,H
109	6D	MOV L,L	LD L,L
110	6E	MOV L,M	LD L,(HL)
111	6F	MOV L,A	LD L,A
112	70	MOV M,B	LD (HL),B
113	71	MOV M,C	LD (HL),C
114	72	MOV M,D	LD (HL),D
115	73	MOV M,E	LD (HL),E
116	74	MOV M,H	LD (HL),H
117	75	MOV M,L	LD (HL),L
118	76	HLT	HALT
119	77	MOV M,A	LD (HL),A
120	78	MOV A,B	LD A,B
121	79	MOV A,C	LD A,C
122	7A	MOV A,D	LD A,D
123	7B	MOV A,E	LD A,E
124	7C	MOV A,H	LD A,H
125	7D	MOV A,L	LD A,L
126	7E	MOV A,M	LD A,(HL)
127	7F	MOV A,A	LD A,A
128	80	ADD B	ADD A,B

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
129	81	ADD C	ADD A,C
130	82	ADD D	ADD A,D
131	83	ADD E	ADD A,E
132	84	ADD H	ADD A,H
133	85	ADD L	ADD A,L
134	86	ADD M	ADD A,(HL)
135	87	ADD A	ADD A,A
136	88	ADC B	ADC A,B
137	89	ADC C	ADC A,C
138	8A	ADC D	ADC A,D
139	8B	ADC E	ADC A,E
140	8C	ADC H	ADC A,H
141	8D	ADC L	ADC A,L
142	8E	ADC M	ADC A,(HL)
143	8F	ADC A	ADC A,A
144	90	SUB B	SUB A,B
145	91	SUB C	SUB A,C
146	92	SUB D	SUB A,D
147	93	SUB E	SUB A,E
148	94	SUB H	SUB A,H
149	95	SUB L	SUB A,L
150	96	SUB M	SUB (HL)
151	97	SUB A	SUB A,A
152	98	SBB B	SBC A,B
153	99	SBB C	SBC A,C
154	9A	SBB D	SBC A,D
155	9B	SBB E	SBC A,E
156	9C	SBB H	SBC A,H
157	9D	SBB L	SBC A,L
158	9E	SBB M	SBC A,(HL)
159	9F	SBB A	SBC A,A

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
160	A0	ANA B	AND B
161	A1	ANA C	AND C
162	A2	ANA D	AND D
163	A3	ANA E	AND E
164	A4	ANA H	AND H
165	A5	ANA L	AND L
166	A6	ANA M	AND (HL)
167	A7	ANA A	AND A
168	A8	XRA B	XOR B
169	A9	XRA C	XOR C
170	AA	XRA D	XOR D
171	AB	XRA E	XOR E
172	AC	XRA H	XOR H
178	AD	XRA L	XOR L
174	AE	XRA M	XOR (HL)
175	AF	XRA A	XOR A
176	B0	ORA B	OR B
177	B1	ORA C	OR C
178	B2	ORA D	OR D
179	B3	ORA E	OR E
180	B4	ORA H	OR H
181	B5	ORA L	OR L
182	B6	ORA M	OR (HL)
183	B7	ORA A	OR A
184	B8	CMP B	CP B
185	B9	CMP C	CP C
186	BA	CMP D	CP D
187	BB	CMP E	CP E
188	BC	CMP H	CP H
189	BD	CMP L	CP L
190	BE	CMP M	CP (HL)

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
191	BF	CMP A	CP A
192	C0	RNZ	RET NZ
193	C1	POP B	POP BC
194	C2 FF FF	JNZ FFFF	JP NZ,FFFF
195	C3 FF FF	JMP FFFF	JP FFFF
196	C4 FF FF	CNZ FFFF	CALL NZ,FFFF
197	C5	PUSH B	PUSH BC
198	C6 FF	ADI FF	ADD A,FF
199	C7	RST 0	RST 00
200	C8	RZ	RET Z
201	C9	RET	RET
202	CA FF FF	JZ FFFF	JP Z,FFFF
203	CB	-Data-	-Data-
204	CC FF FF	CZ FFFF	CALL Z,FFFF
205	CD FF FF	CALL FFFF	CALL FFFF
206	CE FF	ACI FF	ADC A,FF
207	CF	RST 1	RST 08
208	D0	RNC	RET NC
209	D1	POP D	POP DE
210	D2 FF FF	JNC FFFF	JP NC,FFFF
211	D3 FF	OUT FF	OUT (FF),A
212	D4 FF FF	CNC FFFF	CALL NC,FFFF
213	D5	PUSH D	PUSH DE
214	D6 FF	SUI FF	SUB FF
215	D7	RST 2	RST 10
216	D8	RC	RET C
217	D9	-Data-	-Data-
218	DA FF FF	JC FFFF	JP C,FFFF
219	DB FF	IN FF	IN A,(FF)
220	DC FF FF	CC FFFF	CALL C,FFFF
221	DD	-Data-	-Data-

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
222	DE FF	SBI FF	SBC FF
223	DF	RST 3	RST 18
224	E0	RPO	RET PO
225	E1	POP H	POP HL
226	E2 FF FF	JPO FFFF	JP PO,FFFF
227	E3	XTHL	EX (SP),HL
228	E4 FF FF	CPO FFFF	CALL PO,FFFF
229	E5	PUSH H	PUSH HL
230	E6 FF	ANI FF	AND FF
231	E7	RST 4	RST 20
232	E8	RPE	RET PE
233	E9	PCHL	JP (HL)
234	EA FF FF	JPE FFFF	JP PE,FFFF
235	EB	XCHG	EX DE,HL
236	EC FF FF	CPE FFFF	CALL PE,FFFF
237	ED	-Data-	-Data-
238	EE FF	XRI FF	XOR FF
239	EF	RST 5	RST 28
240	F0	RP	RET P
241	F1	POP PSW	POP AF
242	F2 FF FF	JP FFFF	JP P,FFFF
243	F3	DI	DI
244	F4 FF FF	CP FFFF	CALL P,FFFF
245	F5	PUSH PSW	PUSH AF
246	F6 FF	ORI FF	OR FF
247	F7	RST 6	RST 30
248	F8	RM	RET M
249	F9	SPHL	LD SP,HL
250	FA FF FF	JM FFFF	JP M,FFFF
251	FB	EI	EI
252	FC FF FF	CM FFFF	CALL M,FFFF

Table C.1. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
253	FD	-Data-	-Data-
254	FE FF	CPI FF	CP FF
255	FF	RST 7	RST 38

Table C.1. (cont.)

Decimal	Hex	8085 Opcode	8080 Opcode
206	CE FF	ACI FF	ADC A,FF
143	8F	ADC A	ADC A,A
136	88	ADC B	ADC A,B
137	89	ADC C	ADC A,C
138	8A	ADC D	ADC A,D
139	8B	ADC E	ADC A,E
140	8C	ADC H	ADC A,H
141	8D	ADC L	ADC A,L
142	8E	ADC M	ADC A,(HL)
135	87	ADD A	ADD A,A
128	80	ADD B	ADD A,B
129	81	ADD C	ADD A,C
130	82	ADD D	ADD A,D
131	83	ADD E	ADD A,E
132	84	ADD H	ADD A,H
133	85	ADD L	ADD A,L
134	86	ADD M	ADD A,(HL)
198	C6 FF	ADI FF	ADD A,FF
167	A7	ANA A	AND A
160	A0	ANA B	AND B
161	A1	ANA C	AND C
162	A2	ANA D	AND D
163	A3	ANA E	AND E
164	A4	ANA H	AND H
165	A5	ANA L	AND L
166	A6	ANA M	AND (HL)
230	E6 FF	ANI FF	AND FF
205	CD FF FF	CALL FFFF	CALL FFFF
220	DC FF FF	CC FFFF	CALL C,FFFF
252	FC FF FF	CM FFFF	CALL M,FFFF

Table C.2. Alphabetical list of 8085 opcodes

Decimal	Hex	8085 Opcode	Z80 Opcode
47	2F	CMA	CPL
63	3F	CMC	CCF
191	BF	CMP A	CP A
184	B8	CMP B	CP B
185	B9	CMP C	CP C
186	BA	CMP D	CP D
187	BB	CMP E	CP E
188	BC	CMP H	CP H
189	BD	CMP L	CP L
190	BE	CMP M	CP (HL)
212	D4 FF FF	CNC FFFF	CALL NC,FFFF
196	C4 FF FF	CNZ FFFF	CALL NZ,FFFF
244	F4 FF FF	CP FFFF	CALL P,FFFF
236	EC FF FF	CPE FFFF	CALL PE,FFFF
254	FE FF	CPI FF	CP FF
228	E4 FF FF	CPO FFFF	CALL PO,FFFF
204	CC FF FF	CZ FFFF	CALL Z,FFFF
39	27	DAA	DAA
9	09	DAD B	ADD HL,BC
25	19	DAD D	ADD HL,DE
41	29	DAD HL	ADD HL,HL
57	39	DAD SP	ADD HL,SP
61	3D	DCR A	DEC A
5	05	DCR B	DEC B
13	0D	DCR C	DEC C
21	15	DCR D	DEC D
29	1D	DCR E	DEC E
37	25	DCR H	DEC H
45	2D	DCR L	DEC L
53	35	DCR M	DEC (HL)

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
11	0B	DCX B	DEC BC
27	1B	DCX D	DEC DE
43	2B	DCX H	DEC HL
59	3B	DCX SP	DEC SP
243	F3	DI	DI
251	FB	EI	EI
118	76	HLT	HALT
219	DB FF	IN FF	IN A,(FF)
60	3C	INR A	INC A
4	04	INR B	INC B
12	0C	INR C	INC C
20	14	INR D	INC D
28	1C	INR E	INC E
36	24	INR H	INC H
44	2C	INR L	INC L
52	34	INR M	INC (HL)
3	03	INX B	INC BC
19	13	INX D	INC DE
35	23	INX H	INC HL
51	33	INX SP	INC SP
218	DA FF FF	JC FFFF	JP C,FFFF
250	FA FF FF	JM FFFF	JP M,FFFF
195	C3 FF FF	JMP FFFF	JP FFFF
210	D2 FF FF	JNC FFFF	JP NC,FFFF
194	C2 FF FF	JNZ FFFF	JP NZ,FFFF
242	F2 FF FF	JP FFFF	JP P,FFFF
234	EA FF FF	JPE FFFF	JP PE,FFFF
226	E2 FF FF	JPO FFFF	JP PO,FFFF
202	CA FF FF	JZ FFFF	JP Z,FFFF
58	3A FF FF	LDA FFFF	LD A,(FFFF)
10	0A	LDAX B	LD A,(BC)

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
26	1A	LDAX D	LD A,(DE)
42	2A FF FF	LHLD FFFF	LD HL,(FFFF)
1	01 FF FF	LXI B,FFFF	LD BC,FFFF
17	11 FF FF	LXI D,FFFF	LD DE,FFFF
33	21 FF FF	LXI H,FFFF	LD HL,FFFF
49	31 FF FF	LXI SP,FFFF	LD SP,FFFF
127	7F	MOV A,A	LD A,A
120	78	MOV A,B	LD A,B
121	79	MOV A,C	LD A,C
122	7A	MOV A,D	LD A,D
123	7B	MOV A,E	LD A,E
124	7C	MOV A,H	LD A,H
125	7D	MOV A,L	LD A,L
126	7E	MOV A,M	LD A,(HL)
71	47	MOV B,A	LD B,A
64	40	MOV B,B	LD B,B
65	41	MOV B,C	LD B,C
66	42	MOV B,D	LD B,D
67	43	MOV B,E	LD B,E
68	44	MOV B,H	LD B,H
69	45	MOV B,L	LD B,L
70	46	MOV B,M	LD B,(HL)
79	4F	MOV C,A	LD C,A
72	48	MOV C,B	LD C,B
73	49	MOV C,C	LD C,C
74	4A	MOV C,D	LD C,D
75	4B	MOV C,E	LD C,E
76	4C	MOV C,H	LD C,H
77	4D	MOV C,L	LD C,L
78	4E	MOV C,M	LD C,(HL)
87	57	MOV D,A	LD D,A

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
80	50	MOV D,B	LD D,B
81	51	MOV D,C	LD D,C
82	52	MOV D,D	LD D,D
83	53	MOV D,E	LD D,E
84	54	MOV D,H	LD D,H
85	55	MOV D,L	LD D,L
86	56	MOV D,M	LD D,(HL)
95	5F	MOV E,A	LD E,A
88	58	MOV E,B	LD E,B
89	59	MOV E,C	LD E,C
90	5A	MOV E,D	LD E,D
91	5B	MOV E,E	LD E,E
92	5C	MOV E,H	LD E,H
93	5D	MOV E,L	LD E,L
94	5E	MOV E,M	LD E,(HL)
103	67	MOV H,A	LD H,A
96	60	MOV H,B	LD H,B
97	61	MOV H,C	LD H,C
98	62	MOV H,D	LD H,D
99	63	MOV H,E	LD H,E
100	64	MOV H,H	LD H,H
101	65	MOV H,L	LD H,L
102	66	MOV H,M	LD H,(HL)
111	6F	MOV L,A	LD L,A
104	68	MOV L,B	LD L,B
105	69	MOV L,C	LD L,C
106	6A	MOV L,D	LD L,D
107	6B	MOV L,E	LD L,E
108	6C	MOV L,H	LD L,H
109	6D	MOV L,L	LD L,L
110	6E	MOV L,M	LD L,(HL)

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
119	77	MOV M,A	LD (HL),A
112	70	MOV M,B	LD (HL),B
113	71	MOV M,C	LD (HL),C
114	72	MOV M,D	LD (HL),D
115	73	MOV M,E	LD (HL),E
116	74	MOV M,H	LD (HL),H
117	75	MOV M,L	LD (HL),L
62	3E FF	MVI A,FF	LD A,FF
6	06 FF	MVI B,FF	LD B,FF
14	0E FF	MVI C,FF	LD C,FF
22	16 FF	MVI D,FF	LD D,FF
30	1E FF	MVI E,FF	LD E,FF
38	26 FF	MVI H,FF	LD H,FF
46	2E FF	MVI L,FF	LD L,FF
54	36 FF	MVI M,FF	LD (HL),FF
0	00	NOP	NOP
183	B7	ORA A	OR A
176	B0	ORA B	OR B
177	B1	ORA C	OR C
178	B2	ORA D	OR D
179	B3	ORA E	OR E
180	B4	ORA H	OR H
181	B5	ORA L	OR L
182	B6	ORA M	OR (HL)
246	F6 FF	ORI FF	OR FF
211	D3 FF	OUT FF	OUT (FF),A
233	E9	PCHL	JP (HL)
193	C1	POP B	POP BC
209	D1	POP D	POP DE
225	E1	POP H	POP HL
241	F1	POP PSW	POP AF

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
197	C5	PUSH B	PUSH BC
213	D5	PUSH D	PUSH DE
229	E5	PUSH H	PUSH HL
245	F5	PUSH PSW	PUSH AF
23	17	RAL	RLA
31	1F	RAR	RRA
216	D8	RC	RET C
201	C9	RET	RET
32	20	RIM	-----
7	07	RLC	RLCA
248	F8	RM	RET M
208	D0	RNC	RET NC
192	C0	RNZ	RET NZ
240	F0	RP	RET P
232	E8	RPE	RET PE
224	E0	RPO	RET PO
15	0F	RRC	RRCA
199	C7	RST 0	RST 00
207	CF	RST 1	RST 08
215	D7	RST 2	RST 10
223	DF	RST 3	RST 18
231	E7	RST 4	RST 20
239	EF	RST 5	RST 28
247	F7	RST 6	RST 30
255	FF	RST 7	RST 38
200	C8	RZ	RET Z
159	9F	SBB A	SBC A,A
152	98	SBB B	SBC A,B
153	99	SBB C	SBC A,C
154	9A	SBB D	SBC A,D

Table C.2. (cont.)

Decimal	Hex	8085 Opcode	Z80 Opcode
155	9B	SBB E	SBC A,E
156	9C	SBB H	SBC A,H
157	9D	SBB L	SBC A,L
158	9E	SBB M	SBC A,(HL)
222	DE FF	SBI FF	SBC FF
34	22 FF FF	SHLD FFFF	LD (FFFF),HL
48	30	SIM	-----
249	F9	SPHL	LD SP,HL
50	32 FF FF	STA FFFF	LD (FFFF),A
2	02	STAX B	LD (BC),A
18	12	STAX D	LD (DE),A
55	37	STC	SCF
151	97	SUB A	SUB A,A
144	90	SUB B	SUB A,B
145	91	SUB C	SUB A,C
146	92	SUB D	SUB A,D
147	93	SUB E	SUB A,E
148	94	SUB H	SUB A,H
149	95	SUB L	SUB A,L
150	96	SUB M	SUB (HL)
214	D6 FF	SUI FF	SUB FF
235	EB	XCHG	EX DE,HL
175	AF	XRA A	XOR A
168	A8	XRA B	XOR B
169	A9	XRA C	XOR C
170	AA	XRA D	XOR D
171	AB	XRA E	XOR E
172	AC	XRA H	XOR H
173	AD	XRA L	XOR L
174	AE	XRA M	XOR (HL)
238	EE FF	XRI FF	XOR FF
227	E3	XTHL	EX (SP),HL

Table C.2. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
142	8E	ADC M	ADC A,(HL)
143	8F	ADC A	ADC A,A
136	88	ADC B	ADC A,B
137	89	ADC C	ADC A,C
138	8A	ADC D	ADC A,D
139	8B	ADC E	ADC A,E
206	CE FF	ACI FF	ADC A,FF
140	8C	ADC H	ADC A,H
141	8D	ADC L	ADC A,L
134	86	ADD M	ADD A,(HL)
135	87	ADD A	ADD A,A
128	80	ADD B	ADD A,B
129	81	ADD C	ADD A,C
130	82	ADD D	ADD A,D
131	83	ADD E	ADD A,E
198	C6 FF	ADI FF	ADD A,FF
132	84	ADD H	ADD A,H
133	85	ADD L	ADD A,L
9	09	DAD B	ADD HL,BC
25	19	DAD D	ADD HL,DE
41	29	DAD HL	ADD HL,HL
57	39	DAD SP	ADD HL,SP
166	A6	ANA M	AND (HL)
167	A7	ANA A	AND A
160	A0	ANA B	AND B
161	A1	ANA C	AND C
162	A2	ANA D	AND D
163	A3	ANA E	AND E
230	E6 FF	ANI FF	AND FF
164	A4	ANA H	AND H
165	A5	ANA L	AND L

Table C.3. Alphabetical list of Z80 opcodes

Decimal	Hex	8080 Opcode	Z80 Opcode
220	DC FF FF	CC FFFF	CALL C,FFFF
205	CD FF FF	CALL FFFF	CALL FFFF
252	FC FF FF	CM FFFF	CALL M,FFFF
212	D4 FF FF	CNC FFFF	CALL NC,FFFF
196	C4 FF FF	CNZ FFFF	CALL NZ,FFFF
244	F4 FF FF	CP FFFF	CALL P,FFFF
236	EC FF FF	CPE FFFF	CALL PE,FFFF
228	E4 FF FF	CPO FFFF	CALL PO,FFFF
204	CC FF FF	CZ FFFF	CALL Z,FFFF
63	3F	CMC	CCF
190	BE	CMP M	CP (HL)
191	BF	CMP A	CP A
184	B8	CMP B	CP B
185	B9	CMP C	CP C
186	BA	CMP D	CP D
187	BB	CMP E	CP E
254	FE FF	CPI FF	CP FF
188	BC	CMP H	CP H
189	BD	CMP L	CP L
47	2F	CMA	CPL
39	27	DAA	DAA
53	35	DCR M	DEC (HL)
61	3D	DCR A	DEC A
5	05	DCR B	DEC B
11	0B	DCX B	DEC BC
13	0D	DCR C	DEC C
21	15	DCR D	DEC D
27	1B	DCX D	DEC DE
29	1D	DCR E	DEC E
37	25	DCR H	DEC H
43	2B	DCX H	DEC HL
45	2D	DCR L	DEC L
59	3B	DCX SP	DEC SP
243	F3	DI	DI

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
251	FB	EI	EI
227	E3	XTHL	EX (SP),HL
235	EB	XCHG	EX DE,HL
118	76	HLT	HALT
219	DB FF	IN FF	IN A,(FF)
52	34	INR M	INC (HL)
60	3C	INR A	INC A
4	04	INR B	INC B
3	03	INX B	INC BC
12	0C	INR C	INC C
20	14	INR D	INC D
19	13	INX D	INC DE
28	1C	INR E	INC E
36	24	INR H	INC H
35	23	INX H	INC HL
44	2C	INR L	INC L
51	33	INX SP	INC SP
233	E9	PCHL	JP (HL)
218	DA FF FF	JC FFFF	JP C,FFFF
195	C3 FF FF	JMP FFFF	JP FFFF
250	FA FF FF	JM FFFF	JP M,FFFF
210	D2 FF FF	JNC FFFF	JP NC,FFFF
194	C2 FF FF	JNZ FFFF	JP NZ,FFFF
242	F2 FF FF	JP FFFF	JP P,FFFF
234	EA FF FF	JPE FFFF	JP PE,FFFF
226	E2 FF FF	JPO FFFF	JP PO,FFFF
202	CA FF FF	JZ FFFF	JP Z,FFFF
2	02	STAX B	LD (BC),A
18	12	STAX D	LD (DE),A
50	32 FF FF	STA FFFF	LD (FFFF),A
34	22 FF FF	SHLD FFFF	LD (FFFF),HL
119	77	MOV M,A	LD (HL),A
112	70	MOV M,B	LD (HL),B
113	71	MOV M,C	LD (HL),C
114	72	MOV M,D	LD (HL),D

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
115	73	MOV M,E	LD (HL),E
54	36 FF	MVI M,FF	LD (HL),FF
116	74	MOV M,H	LD (HL),H
117	75	MOV M,L	LD (HL),L
10	0A	LDAX B	LD A,(BC)
26	1A	LDAX D	LD A,(DE)
58	3A FF FF	LDA FFFF	LD A,(FFFF)
126	7E	MOV A,M	LD A,(HL)
127	7F	MOV A,A	LD A,A
120	78	MOV A,B	LD A,B
121	79	MOV A,C	LD A,C
122	7A	MOV A,D	LD A,D
123	7B	MOV A,E	LD A,E
62	3E FF	MVI A,FF	LD A,FF
124	7C	MOV A,H	LD A,H
125	7D	MOV A,L	LD A,L
70	46	MOV B,M	LD B,(HL)
71-	47	MOV B,A	LD B,A
64	40	MOV B,B	LD B,B
65	41	MOV B,C	LD B,C
66	42	MOV B,D	LD B,D
67	43	MOV B,E	LD B,E
6	06 FF	MVI B,FF	LD B,FF
68	44	MOV B,H	LD B,H
69	45	MOV B,L	LD B,L
1	01 FF FF	LXI B,FFFF	LD BC,FFFF
78	4E	MOV C,M	LD C,(HL)
79	4F	MOV C,A	LD C,A
72	48	MOV C,B	LD C,B
73	49	MOV C,C	LD C,C
74	4A	MOV C,D	LD C,D
75	4B	MOV C,E	LD C,E
14	0E FF	MVI C,FF	LD C,FF
76	4C	MOV C,H	LD C,H

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
77	4D	MOV C,L	LD C,L
86	56	MOV D,M	LD D,(HL)
87	57	MOV D,A	LD D,A
80	50	MOV D,B	LD D,B
81	51	MOV D,C	LD D,C
82	52	MOV D,D	LD D,D
83	53	MOV D,E	LD D,E
22	16 FF	MVI D,FF	LD D,FF
84	54	MOV D,H	LD D,H
85	55	MOV D,L	LD D,L
17	11 FF FF	LXI D,FFFF	LD DE,FFFF
94	5E	MOV E,M	LD E,(HL)
95	5F	MOV E,A	LD E,A
88	58	MOV E,B	LD E,B
89	59	MOV E,C	LD E,C
90	5A	MOV E,D	LD E,D
91	5B	MOV E,E	LD E,E
30	1E FF	MVI E,FF	LD E,FF
92	5C	MOV E,H	LD E,H
93	5D	MOV E,L	LD E,L
102	66	MOV H,M	LD H,(HL)
103	67	MOV H,A	LD H,A
96	60	MOV H,B	LD H,B
97	61	MOV H,C	LD H,C
98	62	MOV H,D	LD H,D
99	63	MOV H,E	LD H,E
38	26 FF	MVI H,FF	LD H,FF
100	64	MOV H,H	LD H,H
101	65	MOV H,L	LD H,L
42	2A FF FF	LHLD FFFF	LD HL,(FFFF)
33	21 FF FF	LXI H,FFFF	LD HL,FFFF
110	6E	MOV L,M	LD L,(HL)
111	6F	MOV L,A	LD L,A
104	68	MOV L,B	LD L,B

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
105	69	MOV L,C	LD L,C
106	6A	MOV L,D	LD L,D
107	6B	MOV L,E	LD L,E
46	2E FF	MVI L,FF	LD L,FF
108	6C	MOV L,H	LD L,H
109	6D	MOV L,L	LD L,L
49	31 FF FF	LXI SP,FFFF	LD SP,FFFF
249	F9	SPHL	LD SP,HL
0	00	NOP	NOP
182	B6	ORA M	OR (HL)
183	B7	ORA A	OR A
176	B0	ORA B	OR B
177	B1	ORA C	OR C
178	B2	ORA D	OR D
179	B3	ORA E	OR E
246	F6 FF	ORI FF	OR FF
180	B4	ORA H	OR H
181	B5	ORA L	OR L
211	D3 FF	OUT FF	OUT (FF),A
241	F1	POP PSW	POP AF
193	C1	POP B	POP BC
209	D1	POP D	POP DE
225	E1	POP H	POP HL
245	F5	PUSH PSW	PUSH AF
197	C5	PUSH B	PUSH BC
213	D5	PUSH D	PUSH DE
229	E5	PUSH H	PUSH HL
201	C9	RET	RET
216	D8	RC	RET C
248	F8	RM	RET M
208	D0	RNC	RET NC
192	C0	RNZ	RET NZ
240	F0	RP	RET P
232	E8	RPE	RET PE
224	E0	RPO	RET PO

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
200	C8	RZ	RET Z
23	17	RAL	RLA
7	07	RLC	RLCA
31	1F	RAR	RRA
15	0F	RRC	RRCA
199	C7	RST 0	RST 00
207	CF	RST 1	RST 08
215	D7	RST 2	RST 10
223	DF	RST 3	RST 18
231	E7	RST 4	RST 20
239	EF	RST 5	RST 28
247	F7	RST 6	RST 30
255	FF	RST 7	RST 38
158	9E	SBB M	SBC A,(HL)
159	9F	SBB A	SBC A,A
152	98	SBB B	SBC A,B
153	99	SBB C	SBC A,C
154	9A	SBB D	SBC A,D
155	9B	SBB E	SBC A,E
156	9C	SBB H	SBC A,H
157	9D	SBB L	SBC A,L
222	DE FF	SBI FF	SBC FF
55	37	STC	SCF
150	96	SUB M	SUB (HL)
151	97	SUB A	SUB A,A
144	90	SUB B	SUB A,B
145	91	SUB C	SUB A,C
146	92	SUB D	SUB A,D
147	93	SUB E	SUB A,E
148	94	SUB H	SUB A,H
149	95	SUB L	SUB A,L
214	D6 FF	SUI FF	SUB FF
174	AE	XRA M	XOR (HL)
175	AF	XRA A	XOR A

Table C.3. (cont.)

Decimal	Hex	8080 Opcode	Z80 Opcode
168	A8	XRA B	XOR B
169	A9	XRA C	XOR C
170	AA	XRA D	XOR D
171	AB	XRA E	XOR E
238	EE FF	XRI FF	XOR FF
172	AC	XRA H	XOR H
173	AD	XRA L	XOR L

Table C.3. (cont.)

Appendix D
Bibliography

MCS-80/85 Family User's Manual, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. Order no. 205775. $7.50.

8080/8085 Assembly Language Programming Manual, Intel Corpoartion, 3065 Bowers Avenue, Santa Clara, CA 95051. Order no. 980940. $13.50.

8080/8085 Software Design, Book 1, by Christopher A. Titus, David G. Larsen, and Jonathon A. Titus, Howard W. Sams & Co., Inc. 4300 West 62nd St. Indianapolis, Indiana 46268, 1978. ISBN 0-672-21541-1. $12.95.

8080/8085 Software Design, Book 2, by Christopher A. Titus, David G. Larsen, and Jonathon A. Titus, Howard W. Sams & Co., Inc. 4300 West 62nd St. Indianapolis, Indiana 46268, 1978. ISBN 0-672-21615-9. $12.95.

8080A-8085 Assembly Language Programming by Lance A. Leventhal, Osborne/McGraw-Hill, Berkeley, California, 1978, ISBN 0-931988-10-1. $18.95

8080/8085 Assembly Language Subroutines by Lance A. Leventhal, Winthrop Saville, Osborne/McGraw-Hill, 2600 Tenth Street, Berkeley, California 94710, 1983, ISBN 0-931988-58-6. $17.95.

Memory Index

Address	Page No.
7FF3	45, 260
E000-E0FF	272
F5F0-F602	254, 277
F5F9	234, 241
F5FC	129, 241, 244
F5FF	192, 241

Alphabetical Index

D

E

F

G

H

I