4

Hidden Powers of the Liquid Crystal Display

Concepts
How liquid crystal displays work
How to program the LCD

The Liquid Crystal Display (LCD) is the main output device for the Model 100. As such, it provides a good starting point for understanding the operation of the Model 100. The LCD also represents a new approach in display technology, an approach that has much promise because it requires less power and space than the older video technology. It is one of the major reasons why the Model 100 is truly portable.

We will start our exploration of the LCD with a general description of liquid crystal displays and then see in detail how the built-in display of the Model 100 works. We will see how to program the display screen, both directly and by calling various levels of subroutines in the computer's ROM.

How Liquid Crystal Displays Work

In contrast to the more traditional video CRT (cathode ray tube), a liquid crystal display does not generate its own light. Instead, it selectively blocks light that comes from the outside.

If you look closely at the LCD of your Model 100, you will see that it consists of a two-dimensional array of tiny squares. These are the picture elements (pixels) of the display (see Figure 4-1). The horizontal pixel positions are numbered from 0 to 239 from left to right, and the vertical positions are labeled from 0 to 63 from top to bottom.

Each tiny square is a sandwich in which the "bread" consists of polarized filter material and the "filling" is made of liquid crystal. Glass plates separate the liquid crystal from the polarizing filters in this sandwich (see Figure 4-2).

Each pixel can be individually lightened or darkened by applying a voltage to it that affects its transparency. A layer of reflective material behind the entire display helps bounce the light through those pixels of the display that are transparent.

To understand how the pixels can be made more or less transparent, you must understand a little about the theory of light. Light is electromagnetic radiation; that is, it consists of combinations of electrical and magnetic waves.

Ordinary light consists of a hodgepodge of individual light waves. Each individual light wave is a precisely balanced pair of electrical and magnetic waves. These component waves both travel in the same direction, the direction of motion of the wave; they vibrate at the same frequency, the frequency of the wave; but their directions of vibration are perpendicular to each other and to the direction of the motion of the wave (see Figure 4-3). The two directions of vibration determine an oriented plane called the "plane of vibration". This gives an orientation to each individual light wave.

A polarizing filter polarizes light by allowing only those light waves to pass through it that are oriented in a certain way. That is, it blocks light

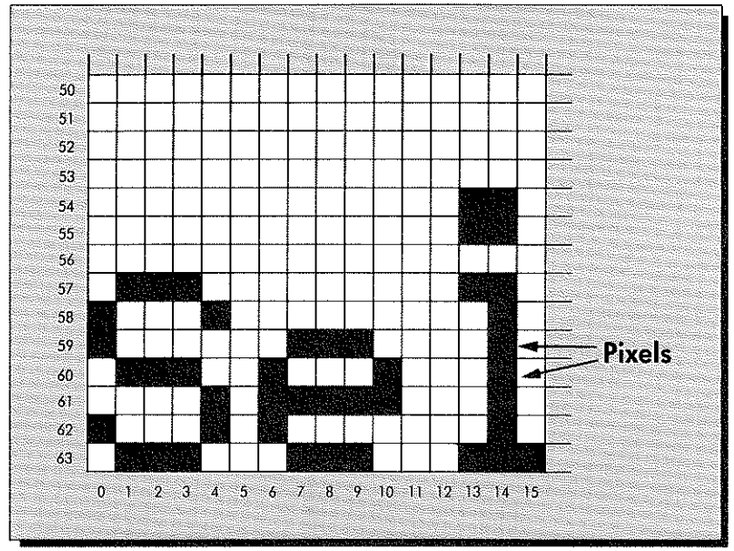

Figure 4-1. Pixels of LCD display

waves whose electrical (or magnetic) components do not line up in a certain direction. Light that has passed through such a filter is said to be "polarized", because the orientations of its individual light waves are closely aligned with each other.

When two polarizing filters are placed together face to face, they will let through light if their polarizations are aligned but will block most of the light if their polarizations are twisted with respect to each other.

Liquid crystal twists the orientation of the light that passes through it. The amount of the twist depends upon the voltage applied to the crystal. When the crystal is sandwiched between two polarizing filters, this twisting, and hence the voltage applied to the liquid crystal, is translated into the degree of transparency of the sandwich.

Light comes into an LCD display from the outside, goes through the sandwich, is reflected from the mirrorlike surface behind the display, and comes back through the display to your eyes. The amount of light that makes its way through this arrangement depends on your viewing angle as well as the voltage that is applied to the liquid crystal. The adjustment wheel on the right side of your Model 100 allows you to select the best voltage for optimum visibility from your particular viewing angle.

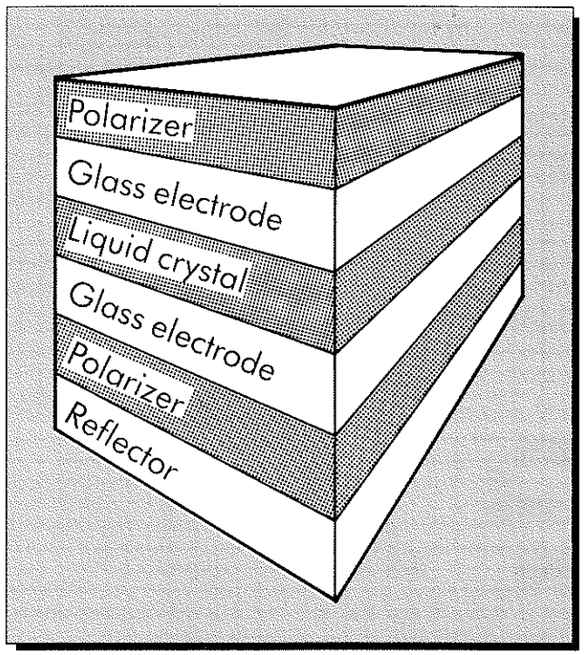

Figure 4-2. Pixel sandwich

Each pixel of the display can be individually controlled with its own voltage. On the Model 100, a display element is transparent when little voltage is applied and becomes opaque as more voltage is applied. On some systems direct current is used, but on the Model 100 alternating current is used to extend the life of the display.

The Model 100's display screen consists of a 240 by 64 array of pixels. Ten chips called "LCD horizontal drivers" directly control ten different regions of the display. Each horizontal driver has 50 lines that can control 50 horizontal positions of the display. The drivers come in pairs, one to control the upper 32 rows and one to control the lower 32 rows of each horizontal section of the display (see Figure 4-4). You can see from this figure that four pairs of horizontal LCD drivers control the first 200 horizontal positions (in 50-position sections) and one pair controls the last 40 positions. These last two horizontal drivers have only 40 of their 50 outputs connected to the display.

Each horizontal LCD driver stores a total of 1600 bits, one for each of the pixels in a 50 by 32 section of the display (except of course for the last pair of drivers, which don't map to a complete 50 by 32 section). The 1600

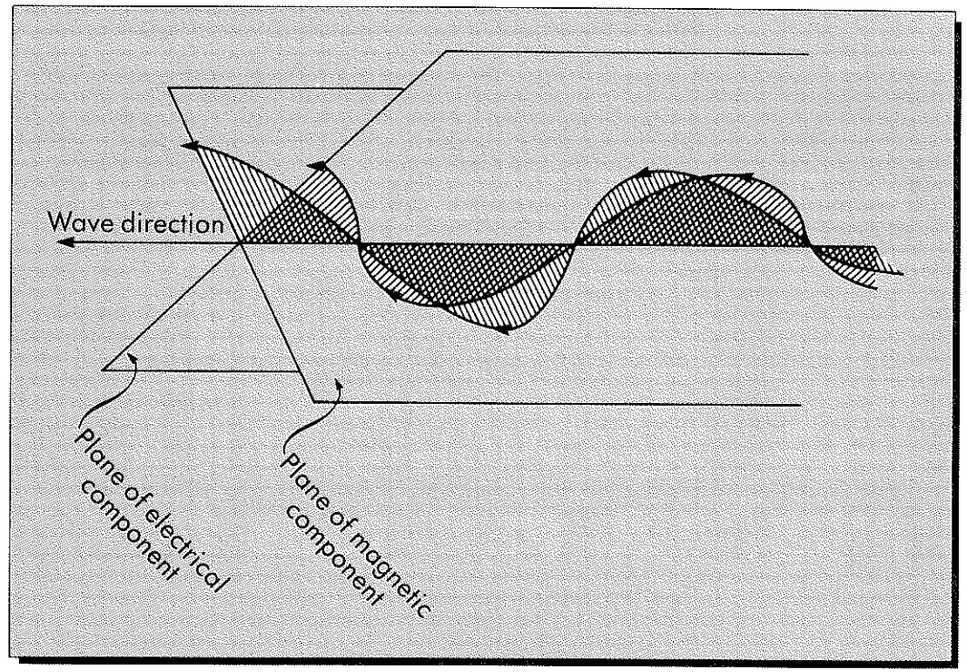

Figure 4-3. Components of a light wave

bits are stored in four banks of 50 bytes. Each bank corresponds to a 50 by 8 strip of the display. Bank 0 corresponds to the top eight rows of pixels, bank 1 corresponds to the next eight rows, bank 2 corresponds to the next eight rows, and the last eight rows correspond to bank 3. Within each bank, each byte corresponds to a 1 by 8 column of pixels (see Figure 4-5).

The horizontal LCD drivers continually refresh the display on their particular sections of the screen. Each output line to the display refreshes a 1 by 32 column of pixels. The information is sent through these lines serially, first the top row, then the second row, and so on, over and over again, creating a top-to-bottom scanning pattern.

A pair of vertical LCD drivers controls the rows of the display, enabling and disabling them in synchronization with the above-mentioned scanning pattern. One vertical driver controls the upper 32 rows of pixels, and the other controls the lower 32 rows. The two vertical drivers scan at the same rate and time through their own parts of the display. When the horizontal drivers produce the information for their first row, the vertical drivers enable only their first row, and so on (see figure 4-6).

The scanning rate is one row about every 446 microseconds. The entire 32 rows are scanned about every 14.3 milliseconds, or 70 times a second. This is slightly faster than a CRT display is normally scanned.

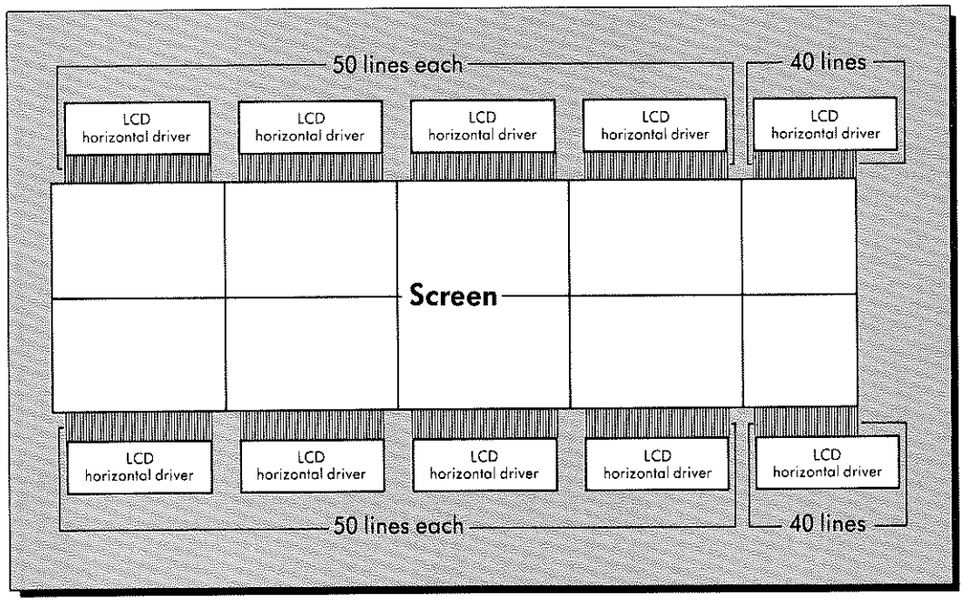

Figure 4-4. Horizontal LCD drivers

In contrast to the horizontal drivers, the vertical drivers do not store any information; they merely maintain a constant scanning pattern.

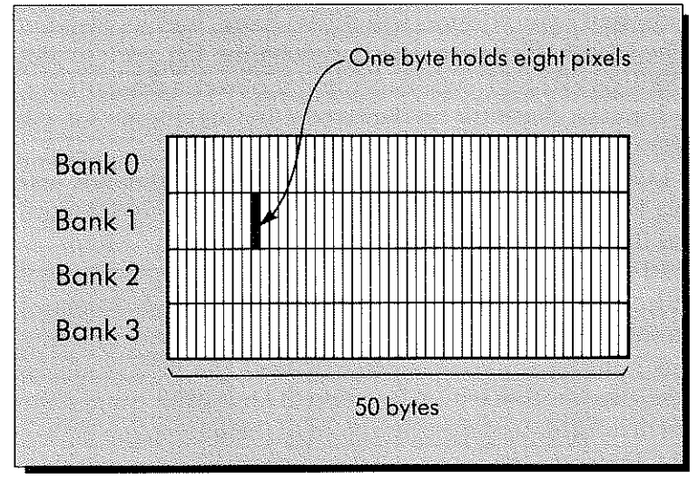

Figure 4-5. Banks and bytes in a horizontal LCD driver

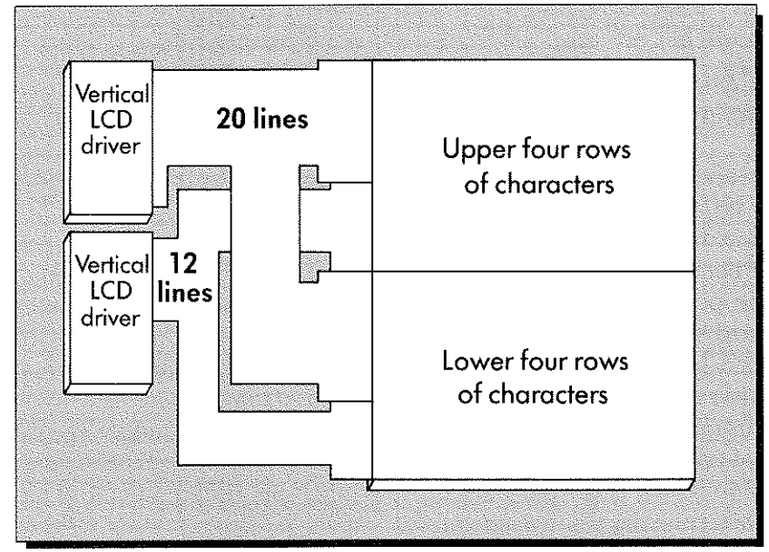

Figure 4-6. Vertical LCD drivers

How to Program the LCD

You can program the LCD screen by sending bytes to the horizontal LCD driver chips. These bytes can also be read back at a later time.

You can separately address each individual pixel in the entire display. To do this, you must determine the pixel's LCD driver, its "bank" within the LCD driver, its horizontal byte position within the bank, and its position within that byte. The Model 100 figures this out each time you ask it to plot a point with the PSET or PRESET command, and it makes a similar computation each time it puts a character on the screen. In this chapter we will see exactly how this works.

To control the LCD, you use port FEh = 254d to send commands and read status and port FFh = 255d to send and receive data bytes (see Figure 4-7). We will discuss this in more detail later.

Ports B9h = 185d and BAh = 186d specify which of the ten horizontal drivers is being addressed. For port B9h = 185d, bits 0 through 4 control the selection of the five LCD drivers across the top of the display in left to right order, and bits 5 through 7 control three of the LCD drivers on the lower left part of the display. Bits 0 and 1 of port BAh = 186d control the remaining two LCD drivers for the lower right part of the screen (see Figure 4-7). To turn an LCD driver "on", so that it can receive a command or transfer data, put a one in the corresponding bit; and to turn it "off", put a zero in that bit. For example, if you put 00001010 binary into port B9h = 185d and the binary pattern 10 into bits 0 and 1 of port BAh = 186d, then the LCD drivers will be "on" and "off" in the following pattern:

off on off on off

off off off off on

Usually, only one LCD driver is programmed at a time; thus there is usually only one "1" bit, with the rest equal to "0".

Several words of warning are needed about using ports B9h = 185d and BAh = 186d because they are also used for a number of other functions, including the power, keyboard, clock, buzzer, and communications lines. Since keyboard scanning, cursor blinking, and clock reading are going on constantly as a background task, you must turn this background task off before selecting the LCD drivers. This is somewhat dangerous, since if you don't turn this task back on, your keyboard will no longer work, and you will lose control of your machine. Fortunately, when you program in BASIC, the keyboard-cursor-clock is turned back on for the INPUT statement and as BASIC finishes running your program.

You should be careful not to disturb bits 2 through 7 of port BAh = 186d. In fact, if you put a zero into bit 4 of this port, you will turn off the power to the machine! To properly program this port, you must read the port first, AND its contents with the mask 11111100 binary, OR the contents with 000000aa binary, where aa is the desired pattern for bits 0 and 1, and then put the result back into the port.

LCD commands, which are sent through port FEh = 254d, allow you to turn the display on and off and specify how data bytes are to be loaded in and out of the LCD driver chip. Each command byte consists of two parts or fields: a two-bit bank selector field that is stored in bits 6 and 7, and a six-bit field that is stored in bits 0 through 5.

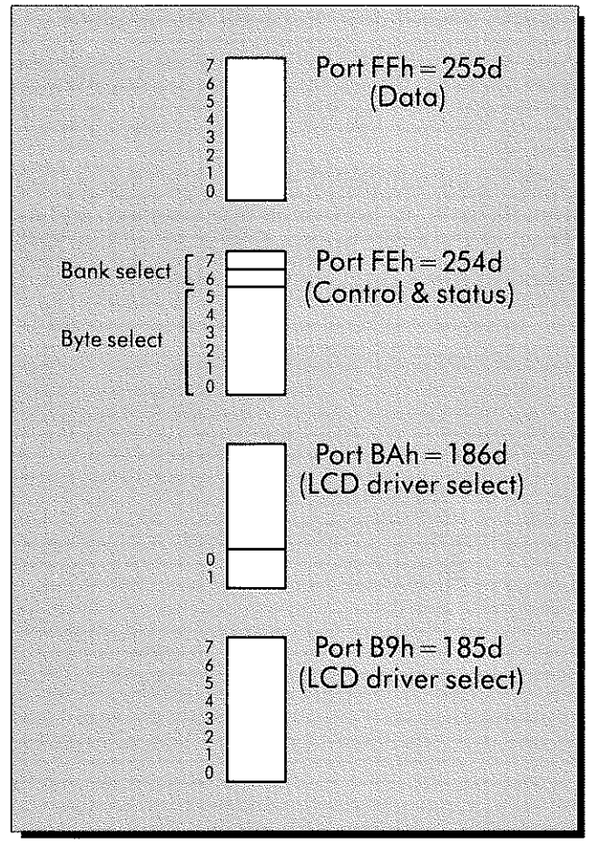

Figure 4-7. Control, status, and data ports for the LCD

The six-bit field of the command byte contains a number between 0 and 63. If this number is in the range from 0 to 49, it indicates a horizontal byte position in the bank specified by the two-bit bank select field. This byte position is called the current byte position and is used to indicate where the next byte will be loaded in or out of the driver. For example, if you send the command byte the binary value 01000011, then the bank select field is 01, and the horizontal position field is 000011 binary or 3 decimal. Thus the next data byte will be at position 3 of bank 1 (see Figure 4-8).

Values greater than 49 in the six-bit field program the LCD driver in other ways. For example, a value of 56 in the six-bit field turns off the display, making the corresponding part of the screen blank; a value of 57 turns the display back on, and values of 58 and 59 affect the order in which bytes are to be loaded in or out of the display.

After a data byte is loaded in or out of the driver (through port FFh = 255d), the current position is advanced. Normally the position advances to the right, but you can reverse the direction by sending a command byte with a value of 58 in its six-bit field. In this mode bytes are loaded into the LCD in right-to-left order. To return to the normal left-to-right loading order, send a command byte of 59.

As we noted above, to address an individual pixel of the display, you must determine its LCD driver, its bank, its horizontal byte position, and its position within that byte. Here is a BASIC program that illustrates how these considerations can be used to plot a pixel anywhere on the screen.

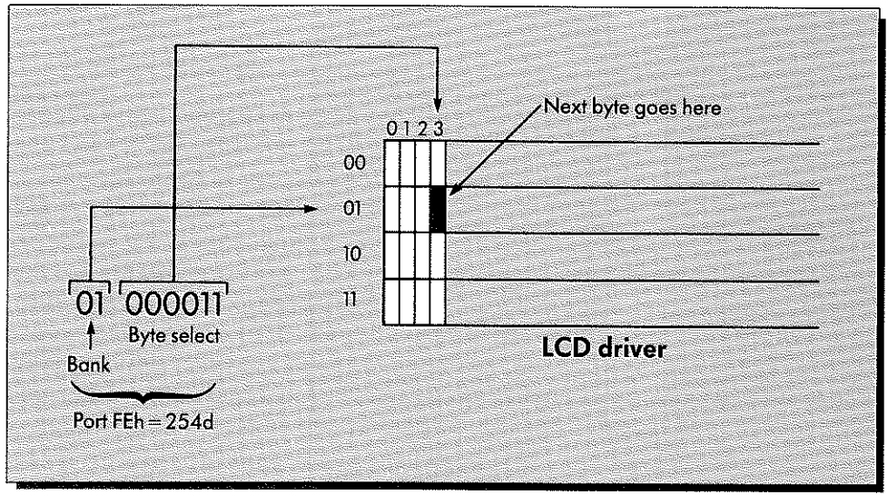

Figure 4-8. Loading an LCD driver


```
100 ' LCD DIRECT PROGRAMMING
110 CLS
120 INPUT "LCD ENABLE BITS (0-1023)";E
130 INPUT "LCD BANK (0-3)";B
140 INPUT "LCD BYTE POSITION (0-63)";H
150 INPUT "LCD BYTE VALUE (0-255)";V
160 INPUT "NUMBER OF BYTES (>0)";N
170 CALL 30300
180 OUT 185, E AND 255
190 P1 = INP(186) AND 254
200 OUT 186, P1 OR (3 AND E/256)
210 OUT 254, 64*(B AND 3) + (63 AND H)
220 FOR I = 1 TO N
230  OUT 255, 255 AND V
240 NEXT I
```

Line 110 clears the screen so that you can better see the program's result. It also forces the input statement to the top of the screen so that the display won't scroll and make your result disappear before you can examine it. Lines 120-160 input five parameters, specifying the acceptable ranges for their values (see Figure 4-9). E is the ten-bit enable/disable pattern that is sent to ports B9h = 185d and BAh = 186d. This selects a combination of LCD drivers. B selects one of the four banks in the selected LCD drivers. H selects the horizontal byte position within the bank. V specifies the bit pattern for the byte. N specifies the number of bytes that will be sent to the LCD drivers. Line 170 turns off the clock-cursor-keyboard background task. (This routine is located at 765Ch = 30,300d, as we shall see later.) Line 180 sets the lower eight enable/disable bits, and lines 190-200 set the upper two enable/disable bits for the LCD drivers. Line 210 computes the bank and horizontal position command byte and sends it out port FEh = 254d. Lines 220-240 form a FOR...NEXT loop to send the data byte out port FFh = 255d the specified number of times. Try typing this program in and running it.

In the next section we will see how the Model 100 controls these quantities to plot points, lines, boxes, and characters.

ROM Routines for the LCD

The Model 100 ROM contains routines to plot and erase points, to draw lines and boxes, and to print characters on the screen. It also contains code to make the cursor blink. We'll discuss these routines in detail so you can learn how to take advantage of them. You will find them useful for creating special effects such as scrolling subsections of the screen and making real-time displays of complex data. Having complete control of the screen is especially useful if you are designing games or educational programs.

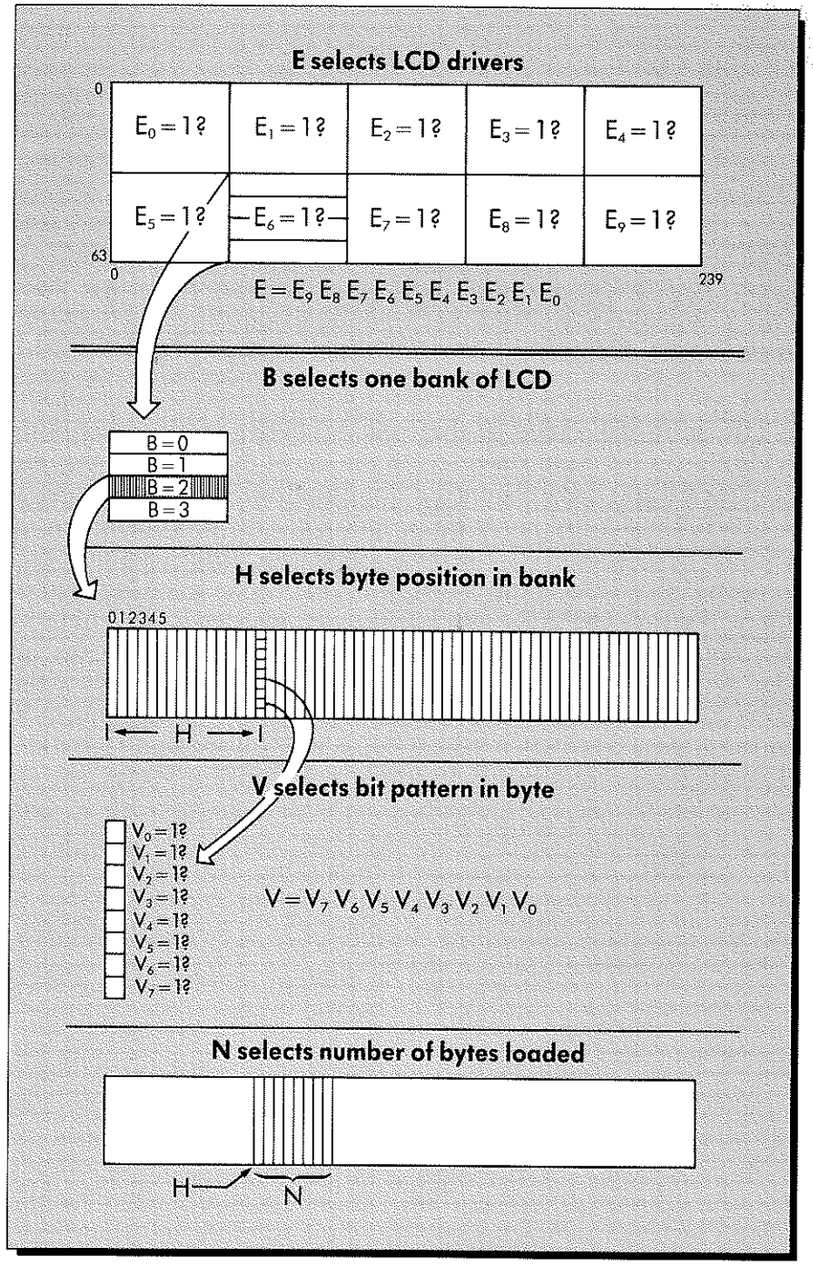

Figure 4-9. How E, B, H, V, and N program the LCD

Point Plotting

Let's start with the point-plotting routines. They give you control of each individual dot on the screen. In BASIC, the PSET and PRESET commands are used to plot points. PSET is used to turn on pixels, and PRESET is used to turn them off.

PSET and PRESET

BASIC commands are implemented as routines in the ROM. The routine for PSET starts at 1C57h = 7255d (see box), and the routine for PRESET starts at 1C66h = 7270d (see box). These routines first call a routine starting at 1D2Eh = 7470d, which gets the (x,y) coordinates of the point from the BASIC command line (see box).

Routine: PSET Command

Purpose: To plot a point on the LCD screen

Entry Point: 1C57h = 7255d

Input: Upon entry, the HL register pair points to the end of the PSET command line, which contains the coordinates of the point in tokenized form. (See the *TRS-80® Model 100 Portable Computer* manual for the syntax of the PSET command.)

Output: To the screen

BASIC Example:

```
CALL 7255,0,63105
```

where the input buffer at F681h = 63,105d contains a tokenized BASIC PSET command line starting with the coordinates of the point. Call the tokenizer routine at 646h = 1606d before using this example.

Special Comments: The input buffer at F681h = 63,105d is also used by the INPUT command.

Routine: PRESET Command

Purpose: To erase a point on the LCD screen

Entry Point: 1C66h = 7270d

Input: Upon entry, the HL register pair points to the end of the PRESET command line, which contains the coordinates of the point in tokenized form. (See the *TRS-80® Model 100 Portable Computer* manual for the syntax of the PRESET command.)

Output: To the screen

BASIC Example:

```
CALL 7270,0,63105
```

where the input buffer at F681h = 63,105d contains a tokenized BASIC PRESET command line starting with the coordinates of the point. Call the tokenizer routine at 646h = 1606d before using this example.

Special Comments: The input buffer at F681h = 63,105d is also used by the INPUT command.

Routine: Get (x,y) Coordinate

Purpose: To get (x,y) coordinate from BASIC command line

Entry Point: 1D2Eh = 7470d

Input: Upon entry, the HL register pair points to a tokenized string containing the (x,y) coordinates.

Output: When the routine returns, the D register contains the value of the x-coordinate and the E register contains the value of the y-coordinate.

BASIC Example: Not directly applicable

Special Comments: None

PLOT/UNPLOT

For PSET the plotting routine is at 744Ch = 29,772d and is called PLOT (see box), and for PRESET it is at 744Dh = 29,773d and is called UNPLOT (see box). These are really two entries into the same plotting routine. In the first case, a nonzero value is placed in the A register at the beginning of the routine. In the second case, the A register is cleared at the beginning of the plotting routine. In either case, the x-coordinate (horizontal position) is in the D register, and the y-coordinate (vertical position) is in the E register.

Routine: PLOT

Purpose: To plot a point on the LCD display screen

Entry Point: 744Ch = 29,772d

Input: Upon entry, the D register contains the x-coordinate and the E register contains the y-coordinate.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

Routine: UNPLOT

Purpose: To erase a point on the LCD display screen

Entry Point: 744Dh = 29,773d

Input: Upon entry, the D register contains the x-coordinate and the E register contains the y-coordinate.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

The first action taken by the PLOT/UNPLOT routine is to call a routine starting at 765Ch = 30,300d that turns off the clock-cursor-keyboard background task (see box). Specifically, this routine turns off interrupt number 7.5, which is normally generated by the clock chip every four milliseconds to run the clock-cursor-keyboard background task. (This interrupt will be discussed in more detail in Chapters 5 and 6.)

Routine: Turn Off and Reset Interrupt 7.5

Purpose: To turn off and rearm interrupt 7.5

Entry Point: 765Ch = 30,300d

Input: None

Output: When the routine returns, interrupt 7.5 is disabled and rearmed for the next time it is enabled.

BASIC Example:

```
CALL 30300
```

Special Comments: The interrupt can be reenabled in a number of ways, such as by means of a PRINT command or the termination of a BASIC program.

The PLOT/UNPLOT routine divides the x-coordinate by 50. The quotient determines which pair of horizontal LCD drivers controls the pixel, and the remainder determines the byte position within the driver.

Enabling the LCD Drivers

The y-coordinate is processed to determine whether the pixel is in the upper or lower half of the screen. This determines which set of five horizontal LCD drivers should be addressed. The HL register points to the first half of a table in memory for the upper half of the screen and the second half of the table for the lower half of the screen (see Figure 4-10). This table contains the bit patterns for enabling the LCD drivers through ports B9h = 185d and BAh = 186d. The quotient determined by the PLOT/UNPLOT routine is added to the HL register to point to the bit pattern for the desired LCD driver. Then the routine at 753Bh = 30,011d (see box) is called to send these enable/disable bits to select the correct LCD driver.

Routine: Enable LCD Drivers

Purpose: To enable LCD drivers

Entry Point: 753Bh = 30,011d

Input: Upon entry, the HL register pair points to an entry in special tables in memory that contain bit patterns to set the 8155 PIO chip that controls the LCD drivers. There are two such tables. One table begins at 7551h = 30,033d, has three bytes per entry, and is indexed by the column position for character positions. The other (see Figure 4-10) begins at 7643h = 30275d, has two bytes per entry, and is indexed by the particular LCD driver.

Output: The specified LCD driver is enabled, and the others are disabled.

BASIC Example:

```
CALL 30011,0,30275+2*L
```

where L is a number between 0 and 9 and indicates the particular LCD driver.

Special Comments: None

The bits that determine the bank number are shifted into the upper two bits, their correct position within the command byte. The bank bits and the horizontal byte position are combined and stored in the B register, ready to be sent to the command port of the LCD driver.

Because the bit for the pixel is stored within a byte that has bits for seven other pixels, the contents of the byte must be read first. This way the values for the other bits can be preserved when the byte is put back.

The routine to read the byte from the LCD driver is located at 74F5h = 29,941d (see box). It waits for the LCD driver status to indicate that the driver is ready to receive a command. Then it sends the command byte. Finally, it reads the data byte (when status indicates the data byte is ready). The ready status is contained in bit 7 of port FEh = 254d.

Routine: Read LCD Bytes

Purpose: To read a sequence of bytes from an LCD driver

Entry Point: 74F5h = 29,941d

Input: Upon entry, the B register contains a command byte for the selected LCD driver, the HL register pair points to an area of memory to which the bytes are to be transferred, and the E register contains the number of bytes to be transferred. The command byte usually selects the bank and byte position within the byte. See the text for further explanation.

Output: When the routine returns, the bytes from the LCD driver are in memory, starting at the location specified by HL upon entry.

BASIC Example: Not directly applicable

Special Comments: None

Address	Binary data		Address
	76543210	76543210	
7644h	00000000	00000001	7643h
7646h	00000000	00000010	7645h
7648h	00000000	00000100	7647h
764Ah	00000000	00001000	7649h
764Ch	00000000	00010000	764Bh
764Eh	00000000	00100000	764Dh
7650h	00000000	01000000	764Fh
7652h	00000000	10000000	7651h
7654h	00000001	00000000	7653h
7656h	00000010	00000000	7655h
	Sent to port BAh	Sent to port B9h	

Figure 4-10. Table for enabling LCD drivers for point plotting

The y-coordinate is used to determine a mask for the bit position within the byte. This uses some modular arithmetic and the same table that was used for the ports B9h = 185d and BAh = 186d. The mask contains a "1" in the correct bit position and "0" in the other positions. For the PSET command, the mask is ORed with the byte just read from the LCD driver. For the PRESET command, the mask is used to clear the appropriate bit of this byte.

The correctly modified byte is sent back to the LCD driver by calling the routine at 74F6h = 29,941d (see box). This works almost the same as the read routine; indeed, it is the same except for a byte at its entry.

Routine: Write LCD Bytes

Purpose: To write a sequence of bytes to an LCD driver

Entry Point: 74F6h = 29,914d

Input: Upon entry, the B register contains a command byte for the selected LCD driver, the HL register pair points to an area of memory from which the bytes are to be transferred, and the E register contains the number of bytes to be transferred. The command byte usually selects the bank and byte position within the byte. See the text for further explanation.

Output: When the routine returns, the bytes from the specified memory area are transferred to the LCD driver.

BASIC Example: Not directly applicable

Special Comments: None

The last thing that the PLOT/UNPLOT routine does is turn on the 7.5 interrupt so that the clock-cursor-keyboard background task can continue. The routine to do this is located at 743Ch = 29,756d (see box). Remember that this background task must continue to operate in order for the keyboard to function.

Routine: Turn on Interrupt 7.5

Purpose: To enable interrupt 7.5

Entry Point: 743Ch = 29,756d

Input: None

Output: To the interrupt control

BASIC Example:

```
CALL 29756
```

Special Comments: None

Line Drawing

Line drawing and area filling are at the next higher level above point plotting. The routine to draw lines starts at 1C6Dh = 7277d (see box). It can be invoked from the BASIC LINE command. The LINE command has a number of parameters and options. It can draw lines and rectangles (filled or unfilled). The endpoints of the lines and the corners of the rectangles can be specified either as a pair of points in the form

(x1,y1)-(x2,y2)

or as a single point in the form

-(x2,y2)

You can see that in the second case, the first point is not specified. Instead, an unseen graphic cursor called the "CP" (current position) is used. Each time a point, line, or box is drawn with the PSET, PRESET, or LINE commands, the CP is updated to the last point referenced. This is done at 1D46h = 7494d in the routine by getting the (x,y) coordinates from the command line. The CP is stored at location F64Eh = 63,054d.

Routine: LINE — BASIC command (Graphics)

Purpose: To draw a line on the LCD screen

Entry Point: 1C6Dh = 7277d

Input: Upon entry, the HL register pair points to the end of the LINE command line, which contains the coordinates of the point in tokenized form. See the *TRS-80® Model 100 Portable Computer* manual for the syntax of the LINE command.

Output: To the screen

BASIC Example:

```
CALL 7277,0,63105
```

where the input buffer at F681h = 63,105d contains a tokenized BASIC LINE command line starting with the coordinates of the point. Call the tokenizer routine at 646h = 1606d before using this example.

Special Comments: None

The line-drawing routine uses a form of Bresenham's line-drawing algorithm and calls either PLOT or UNPLOT to plot or erase points along the line, depending upon the particular "color" chosen. Bresenham's line drawing algorithm is a well-known method for drawing lines quickly. It consists of an initialization stage and a tight loop that steps through the pixels along the line, performing a series of vertical, horizontal, and diagonal moves. In the Model 100's ROM the initialization part starts at 1CD9h = 7385d, and the loop starts at 1D0Ch = 7436d.

The box-drawing option of the LINE command calls the Brẹsenham algorithm four times, once for each side of the box. The routine for this is at 1CBCh = 7356d (see box). A box-fill option at 1CA5h = 7333d (see box) has a loop that calls the Bresenham algorithm over and over again to fill in all the rows inside a specified rectangle.

Routine: Box — Unfilled

Purpose: To draw an unfilled box on the LCD screen

Entry Point: 1CBCh = 7356d

Input: Upon entry, the coordinates of two opposite corner points of the box are on the stack. For each of the two corner points there is one word on the stack. The upper byte of this word contains the x-coordinate, and the lower byte contains the y-coordinate.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

Routine: Box — Filled

Purpose: To draw a filled box on the LCD screen

Entry Point: 1CA5h = 7333d

Input: Upon entry, the coordinates of two opposite corner points of the box are on the stack. For each of the two corner points there is one word on the stack. The upper byte of this word contains the x-coordinate, and the lower byte contains the y-coordinate.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

Character Plotting (Text)

The routines to plot characters on the screen are more complicated than the point- or line-drawing routines. The text routines have a multitude of levels; that is, a higher-level routine calls a lower-level routine, which calls a still lower-level routine, and so forth. We'll go through these routines level by level.

Level 1

Character plotting at the highest level (level 1) can be called by the RST 4 instruction. This is a software interrupt; that is, it is a CPU instruction that acts just like a hardware interrupt. The RST 4 instruction calls whatever routine is located at address 20h = 32d. In the Model 100's ROM, address 20h = 32d is the start of a jump to 4B44h = 19,268d, which is where the character output routine is actually located.

The level 1 character-plotting routine at 4B44h = 19,268d displays a character on the screen at the current cursor position (see box). Before the routine is called, the ASCII code must be in the A register. This routine is called LCD by Radio Shack, but it can be used to direct output to other devices as well, such as the printer and the optional CRT display screen. Perhaps a better name for this routine would be CONSOLE OUT. You can access this routine directly by typing:

```
CALL 32,65
```

or

```
CALL 19268,65
```

In both cases this will place an uppercase "A" on the screen.

Routine: Character Plotting — Level 1

Purpose: To print a character on the LCD screen

Entry Point: 4B44h = 19,268d

Input: Upon entry, the A register contains the ASCII code of the character to be printed.

Output: To the screen

BASIC Example:

```
CALL 19268,A
```

where A is the ASCII code of the character to be printed.

Special Comments: RST 4 also calls this routine (see Chapter 3).

Location F675h = 63,093d contains what we call the "print flag". The level 1 routine checks to see if the print flag is nonzero. If it is nonzero, output from this routine goes to the printer. If it is 0, the routine branches to 4BAAh = 19,370d, where the level 2 character plotting routine is called.

Level 2

The level 2 character-plotting routine is at 4313h = 17,171d (see box). It has "hooks" to allow user-defined routines to be called (see Figure 4-11). By "hook" we simply mean a way for users to attach their own routines.

Routine: Character Plotting — Level 2

Purpose: To print a character on the LCD screen

Entry Point: 4313h = 17,171d

Input: Upon entry, the ASCII code of the character to be printed is in the A register.

Output: To the screen

BASIC Example:

```
CALL 17171,A
```

where A is the ASCII code of the character to be plotted.

Special Comments: None

These "hooks" are done through the RST 7 interrupt instruction (see box in Chapter 3). This instruction calls location 38h = 56d, which jumps to location 7FD6h = 32,726d, where there is a dispatcher routine. The dispatcher routine calls one of the routines whose addresses are stored in a big table in RAM called the "hook" table. This table starts at address FADAh = 64,218d. Normally, the addresses in the first half of the hook table all point to a routine that consists of just a return instruction. The addresses in the second half of this table point to a routine that outputs the illegal function error message. The byte following the RST 7 instruction is used to index into the hook table. For level 2 character plotting, this index is 8. This points to a return instruction. If you want to use your own routine, put its address into the RAM address FADAh + 8, and it will be called every time you plot a character on the screen.

Level 3

After the level 2 character-plotting routine calls the "hook", it calls the level 3 character-plotting routine.

The level 3 character-plotting routine at 431Fh = 17,183d checks location F638h = 63,032d (see box). If this is nonzero, a RST 7 instruction calls the hook table with index 3Ch = 60d. Normally this is an illegal function, but it can be redefined if you want. If location F638h = 63,032d is zero, the

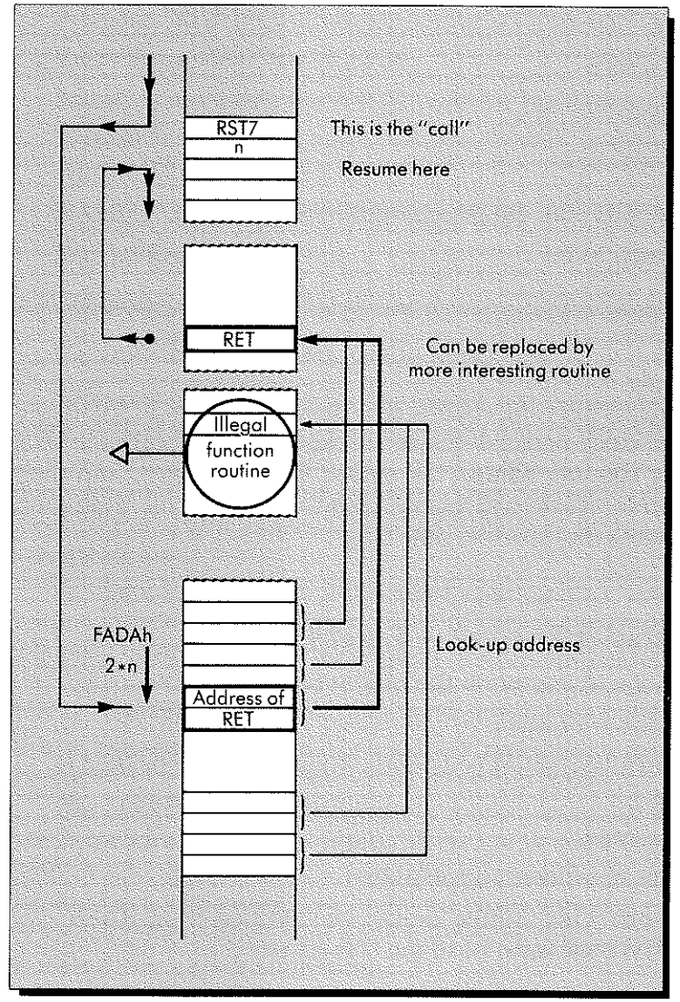

Figure 4-11. How the hook table works

level 4 character-plotting routine is called. Thus location F638h = 63,032d can be used as a device flag for substituting a new device for console output.

Routine: Character Plotting — Level 3

Purpose: To print a character on the LCD screen

Entry Point: 431Fh = 17,183d

Input: Upon entry, the ASCII code of the character to be printed is in the A register.

Output: To the screen

BASIC Example:

```
CALL 17183,A
```

where A is the ASCII code for the character to be printed.

Special Comments: None

Level 4

The level 4 character-plotting routine is located at 4335h = 17,205d (see box). It turns off the clock-cursor-keyboard background task, calls the level 5 character-plotting routine, adjusts the cursor if it is enabled, and then turns the clock-cursor-keyboard background task back on. Location F63Fh = 63,039d stores the cursor enable flag.

Routine: Character Plotting — Level 4

Purpose: To print a character on the LCD screen

Entry Point: 4335h = 17,205d

Input: Upon entry, the ASCII code of the character to be printed is in the C register.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: This routine cannot be CALLed from BASIC because the ASCII code must be in the C register.

Level 5

The level 5 character-plotting routine at 434Ch = 17,228d handles control characters and escape sequences (see box). If the character is not a control character and is not part of an escape sequence, it calls the level 6 character-plotting routine and advances the cursor.

Routine: Character Plotting — Level 5

Purpose: To print a character on the LCD screen

Entry Point: 434Ch = 17,228d

Input: Upon entry, the ASCII code for the character to be printed is in the C register.

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

Control Characters and Escape Sequences

The control characters for the Model 100 include bell (ASCII 7), backspace (ASCII 8), tab (ASCII 9), linefeed (ASCII 10), formfeed (ASCII 12), carriage return (ASCII 13), and delete (ASCII 7Fh = 127d). The level 5 character routine looks for some of these control characters directly and checks for others in a special table at 438Ah = 17,290d (see Appendix K for a complete list).

Escape (ASCII 27) is the control character to initiate an escape sequence. An escape sequence consists of the escape character followed by a sequence of ASCII codes which are to have a special meaning. The Model 100 has a nice set of escape sequences. For example, escape followed by the ASCII code for E is used to indicate that the screen should be erased, and escape followed by the ASCII code for Y and then by two more bytes moves the cursor to a position specified by those two bytes.

The Model 100 looks for escape sequences in a table at 43B8h = 17,336d (see Appendix L). When it detects an escape, it sets a flag (stored at location F646h = 63,046d) so that the next character will be looked up in that table. Some escape sequences have only two characters (for example: escape, E for clearing the screen), and some have more (for example: escape, Y, byte, byte for direct cursor addressing).

Some escape sequences duplicate the actions of the control characters. For example, formfeed (ASCII 12) and the escape "E" sequence both call the routine at 4548h = 17,736d, which clears the screen and puts the cursor into the home position.

A number of escape sequences and control characters can be generated by calling routines (see Appendix M). These are not the routines that actually do the work, such as clearing the screen; instead, they use the RST 4 instruction to send the appropriate control codes and escape sequences. For example, formfeed is generated by calling 4231h = 16,945d, and escape "Q", the escape sequence to turn off the cursor, is generated by calling 424Eh = 16,974d. These routines can be called directly from BASIC programs to perform the indicated functions.

Level 6

The level 6 character-plotting routine is located at 4560h = 17,760d (see box). It puts the ASCII codes for the characters into a special area of memory called the LCD RAM. It also calls the level 7 character-plotting routine to actually plot the character on the screen. The LCD RAM is located from FE00h = 65,024d to FF3Fh = 65,343d. It contains a byte for each character position on the LCD screen. The level 6 character- plotting routine computes the proper address in this RAM and moves the character there so that the current state of the text display on the LCD is always immediately available in this area of main memory.

Routine: Character Plotting — Level 6

Purpose: To print a character on the LCD screen

Entry Point: 4560h = 17,760d

Input: Upon entry, the C register has the ASCII code of the character, and the HL register pair contains the cursor position of the character. The H register contains the column position (1-40), and the L register contains the row position (1-8).

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

The LCD RAM is used when the display is scrolled. Scrolling a screen involves rapidly moving characters from one part of the screen to another. When the screen is scrolled on the Model 100, bytes are read not from the LCD drivers but from the LCD RAM area. The scrolling routine is located at 44D2h = 17,618d (see box). It reads the LCD RAM (via a routine at 4512h = 17,682d — see box) to get the characters and then calls the level 6 character-plotting routine (at 4566h = 17,766d instead of 4560h = 17,760d) to place the scrolled characters back on the screen and into the LCD RAM. At the end of the scrolling routine the next line of the display is erased.

Routine: Scroll

Purpose: To scroll part of the LCD screen

Entry Point: 44D2h = 17,618d

Input: Upon entry, the A register contains the scroll count (1-7), and the L register contains the line number (1-7) of the first line to be scrolled. The H register can contain any value. The number of lines scrolled is one more than the scroll count. The sum of the contents of A and L should not exceed 7.

Output: To the screen and the screen RAM (starting at FE00h = 65,024d).

BASIC Example:

```
CALL 17618,N,L
```

where N is the scroll count and L is the first line to be scrolled.

Special Comments: The scrolling always affects one more line than the scroll count. For example, if you scroll one line, starting with line L, then the contents of line L+1 will be moved to line L and line L+1 will be erased.

Routine: Get Character from LCD RAM

Purpose: To get a character from the LCD display

Entry Point: 4512h = 17,682d

Input: Upon entry, the HL register pair contains the cursor position. H contains the column (1-40), and L contains the row (1-8).

Output: When the routine returns, the ASCII code for the character is in the C register.

BASIC Example: Not directly applicable

Special Comments: None

We have included a BASIC program to illustrate how to use the scrolling routine. This program displays a simple message on each line of the screen and then scrolls lines 5 and 6. To stop the program, hit (BREAK).

```
100  ' SCROLLING EXAMPLE
110  '
120  ' LABEL THE DISPLAY
130    CLS
140    FOR I = 1 TO 8
150     PRINT "LINE";I;
160     IF I<8 THEN PRINT
170    NEXT I
180  '
190  ' SCROLL LOOP
200    PRINT CH$(27);"Y% SCROLL";J;
210    J=J+1
220    CALL 17618,1,5
230    GOTO 190
```

Looking at the program in detail, we see that it consists of two loops. The first loop (lines 120-170) displays the label "LINE i" on each line, where i is the number of the line. Notice that all but the last line are terminated with a PRINT statement (line 160). The last line is handled differently so that it won't be automatically scrolled up and ruin the display.

The second loop scrolls lines 5 and 6 of the display. On line 200, an escape sequence is used to place the cursor on line 6 of the display, then place the message "SCROLL j" there, where j is the value of a variable J. In line 210, J is incremented. In line 220, the scrolling routine is called, with

the A register set to 1 and the HL register pair set to 5. This forces the scrolling to begin on line 5 with a scroll count of 1. Thus two lines, lines 5 and 6, are affected by the scroll. The program then loops around for the next time through the scroll loop.

Level 7

The level 7 character-plotting routine is located at 73EEh = 29,678d. Notice that this address is quite different from the addresses for the other levels of LCD routines. The 7000h area of memory (above 28,672d), where this routine is located, contains routines that are much more "primitive" and device oriented than routines located in other areas.

Routine: Character Plotting — Level 7

Purpose: To print a character on the LCD screen

Entry Point: 73EEh = 29,678d

Input: Upon entry, the C register contains the ASCII code of the character, and the HL register pair contains the character position. H contains the column (1-40), and L contains the row (1-8).

Output: To the screen

BASIC Example: Not directly applicable

Special Comments: None

The level 7 character-plotting routine looks up the bit patterns for the characters in a table starting at 7711h = 30,481d. This table contains five bytes for the dot matrices for characters whose ASCII codes are in the range 20h = 32d through 7Fh = 127d and six bytes for those in the range 80h = 128d through FFh = 255d. The six-byte part of the table starts at 78F1h = 30,961d. The bytes of this table correspond to the columns of the dot matrix for the characters. This is in contrast to the way CRT character generators usually store the character dot matrix, which is row by row.

The level 7 routine turns off the clock-cursor-keyboard background task, stores the stack pointer in FFF8h = 65,528d, and looks up the character in the previously mentioned character table. The routine takes the bytes from this table and stores the dot matrix for the character in a six-byte area starting at FFECh = 65,516d. If the table entry has only five bytes, the sixth position is filled in as blank. The level 7 routine calls upon a byte-plotting routine at 74A2h = 29,858d, which sends the bytes to the LCD drivers and

concludes by turning on the clock-cursor-keyboard background task with the routine at 473Ch = 19.237d.

The byte-plotting routine computes the information to program the LCD driver (see box). Location FFF4h = 65,524d contains the row, and location FFF5h = 65,525d contains the column, of the character position. A table starting at 7551h = 30,033d (see Appendix N) gives information that should be sent out ports B9h = 185d, BAh = 186d, and FEh = 254d for each horizontal character position on the upper and lower halves of the screen. You may recall from our discussion of point plotting that ports B9h = 185d and BAh = 186d are used to select which of the ten LCD drivers should be enabled and that port FEh = 254d is used to select the byte position within the correct LCD driver. The first two bytes of each entry of this table give the enable information that is sent out ports B9h = 185d and BAh = 186d. The third byte of each entry gives the horizontal position within the LCD driver. Bank selection information from the character row position must be combined with it before it is sent out port FEh = 254d.

Routine: Byte Plotting

Purpose: To send six bytes of a character dot matrix to or from LCD drivers

Entry Point: 74A2h = 29,858d

Input: Upon entry, location FFF4h = 65,524d contains the row and location FFF5h = 65,525d contains the column position of the character to be plotted on the screen. The HL register pair points to the area of memory where the bytes of the dot matrix are stored. This is normally a temporary storage buffer located at FFECh = 65,516d. The D register contains the read/write command. If the bytes are to be sent to the LCD driver, D must contain a 1; otherwise, the bytes are to be read from the LCD drivers.

Output: The character is plotted on the screen. Location FFF6h is affected. It contains a pointer to a table used for selecting the correct LCD drivers.

BASIC Example: Not directly applicable

Special Comments: None

The byte-plotting routine for characters shares a good deal of machine code with the byte-loading routines for plotting points. In particular, it shares the section of code that actually sends the bytes to the LCD drivers.

When it is used to plot characters, the byte-plotting routine sends six bytes at a time to the LCD drivers. These bytes are taken from location FFECh = 65,516d, where they were put by the level 7 character-plotting routine, and sent out port FFh = 255d. The LCD driver accepts them serially, incrementing the horizontal byte position each time. Sometimes, however, the character cell overlaps areas of the screen controlled by two different LCD drivers. The routine is very cleverly designed to send the first few bytes to one driver and the last few bytes to the next driver.

Cursor Blinking

Cursor blinking is controlled as one part of the clock-cursor-keyboard background task. The cursor code starts at 7391h = 29,585d. The actual blink routine starts at 73A9h = 29,609d (see box).

Routine: Cursor Blink

Purpose: To blink the cursor

Entry Point: 73A9h = 29,609d

Input: The temporary dot matrix character buffer at FFECh = 65,516d must contain the dot matrix of the character that is to be blinked. Location FFF3h = 65,523d contains a counter to time the blinking.

Output: To the screen

BASIC Example:

```
CALL 29609
```

Special Comments: The background task must be turned off for this BASIC example to work. Use CALL 30300 to turn off the background task.

First, the cursor routine calls the routine at 765Ch = 30,300d. This turns off interrupt 7.5, which initiates the entire clock-cursor-keyboard task. In this case, however, the routine is called to "rearm" the interrupt. This particular interrupt on the 8085 CPU must be rearmed after each use. Next, the cursor routine checks a counter at FFF3h = 65,523d. The interrupt itself happens every 4 milliseconds, but the counter is set to count down from 125, giving a cursor change every 500 milliseconds. The byte-

plotting routine is used to read six bytes from the LCD drivers at the current cursor position on the screen. These bytes are stored starting at location FFECh = 65,516d. They are then reversed and sent back via the byte-plotting routine. A code of 0 in the D register upon entry to the byte-plotting routine means read from the LCD drivers, and a code of 1 in the D register means write to the LCD drivers.

The cursor routine ends in a return that sends it on to the next part of the background task.

Here's a program that blinks the cursor directly. It quickly blinks the cursor 100 times and then exits. You can change its timing to make the cursor behave in any way you want.

```
100 ' BLINK THE CURSOR
110 '
120   CALL 30300
130   FOR I = 1 TO 100
140    CALL 29609
150    FOR J = 1 TO 20:NEXT J
160   NEXT I
```

On line 120 of this program, the background task is disabled because it too blinks the cursor. The rest of the program consists of a FOR loop (lines 130-160) in which the blink routine is called (line 140) and a short delay is made between blinks (line 150). The delay is made with a FOR loop that counts to twenty. To make the cursor blink faster or slower, change the count in this FOR loop.

Summary

In this chapter we have seen how the LCD works and how it is programmed. In particular, we have seen how it plots points, lines, boxes (filled and unfilled), and characters. We have seen that character plotting is a multilevel process in which the top levels are machine-independent and allow for other devices to be attached for console output, while the bottom levels are very much dependent upon the peculiarities of the LCD display. We have also taken a look at the code in the background task that makes the cursor blink.

5

Hidden Powers of the Real-Time Clock

Concepts

- How the real-time clock works
- What happens when you read the time or date
- What happens when you set the time or date
- BASIC time interrupt commands
- The clock-cursor-keyboard background task

The real-time clock provides the Model 100 with a way to tell both the regular "wall clock" time and the calendar date. This allows you to write programs that do things at prescribed times, making your Model 100 into a valuable assistant for reminding you about appointments and other things you have to do. The timing feature can also be used to control equipment, turning it on and off according to whatever rules you program.

The clock also generates a timing pulse that the Model 100 uses to keep a *background task* going. This background task performs a number of vital functions such as blinking the cursor, updating the system time for the ON TIME$ interrupt, maintaining the automatic power shutoff, and scanning the keyboard.

In this chapter we'll explore the secrets of the real-time clock in the Model 100. We'll start with the hardware and then see how it can be used to set and read the computer's time and date, how it helps control the ON TIME$ interrupt, and how it is used in connection with the clock-cursor-keyboard background task.

How the Real-Time Clock Works

The real-time clock in the Model 100 is housed in a chip called μ PD 1990 AC. The time and date can be written to and read from this clock by sending bits through various CPU ports. The clock chip also outputs a timing pulse that triggers an interrupt to drive the clock-cursor-keyboard background task. A crystal keeps this clock ticking by feeding it electrical pulses at a constant predetermined rate. For the Model 100, the crystal for the clock oscillates at 32,768 cycles per second.

The clock chip contains a series of counters that count seconds, minutes, hours, days of the week, days of the month, and months (see Figure 5-1).

Every 32,768 "ticks" of the crystal causes the seconds counter to increment by one; every time the seconds counter reaches 60, it is zeroed and the minutes counter is incremented; and so on through the hours, days, and months. The chip itself does not have a counter for years.

Most of these counters have a regular cycle. However, the job of the days-of-the-month counter is harder because different months have different numbers of days. The chip is specially preprogrammed to handle this.

It is important to realize that the counting goes on independently of the CPU. Thus, it is not influenced by whatever kind of programs are running on the machine. As a result it can keep accurate time.

A forty-bit *shift register* is connected to the time and date registers to assist with transferring information to and from the time and date counters. In general, a shift register is a row of bit "cells" with provision for *serial transfer* operations, in which binary information is shifted left and/or right along the row. There are also *parallel transfer* operations, in which all the bit cells can be loaded or unloaded (read) at once.

Let's look at the way the bits in the shift register are assigned (see Figure 5-2). The forty bits in this register form ten sets, each containing four bits. The first set holds the units digit of the seconds, the second set holds the tens digit of the seconds, the third set holds the units digit of the minutes, the fourth set holds the tens digit of the minutes, the fifth set holds the units

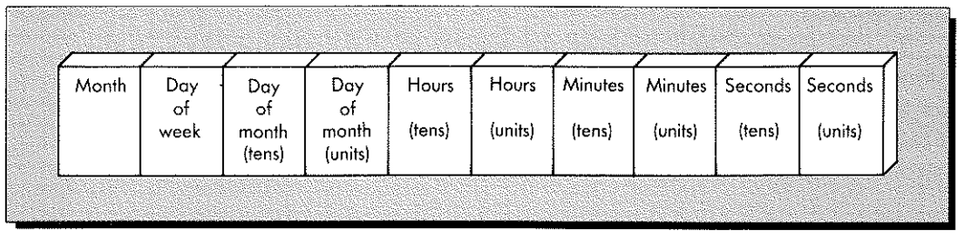

Figure 5-1. Time and date counters

digit of the hour, the sixth set holds the tens digit of the hour, the seventh set holds the units digit of day of the month, the eighth set holds the tens digit of the day of the month, the ninth set holds the day of the week, and the tenth set holds the month. Each set of four bits is a binary coded decimal digit except for the last, which is a hexadecimal encoding of the month.

When the time and date are read from the clock chip, all the bits are transferred at once (in parallel) from the time and date counters to this forty-bit shift register. Then the bits are shifted out of the shift register one by one (serially). Thus the time is "sampled" at a single instant and then moved through the computer bit by bit. Conversely, when the time is set on the clock chip, the individual bits of the time and date are first shifted into the forty-bit shift register one by one (serial transfer), and then the contents of the shift register are transferred all at once to the various time and date registers (parallel transfer) (see Figure 5-3). Other clock chips use other methods for transferring information in and out of the chip. For example, some clock chips transfer the time as a series of bytes rather than a series of bits.

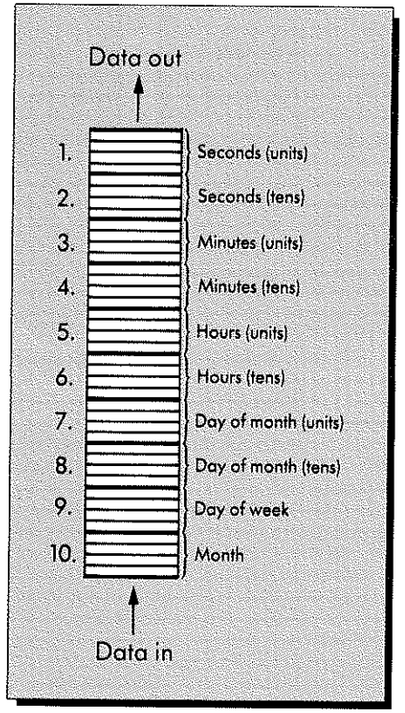

Figure 5-2. The forty-bit shift register in the clock

The clock chip is connected to a number of bits in three different ports of the Model 100 (see Figure 5-4). Three of these, C2, C1, and C0, are command bits and are connected to bits 2, 1, and 0 of port B9h = 185d. These bits form a three-bit binary number that specifies the mode of operation for the chip. When bit C2 is zero, the commands control reading and writing of the real-time clock, and when bit C2 is one, the commands control the timing pulse (see Table 5-1). The clock commands 0 (no operation), 1 (serial transfer mode), 2 (parallel transfer to set the time and date), and 3 (parallel transfer to read the time and date) must be used in combination to set and read the time and date.

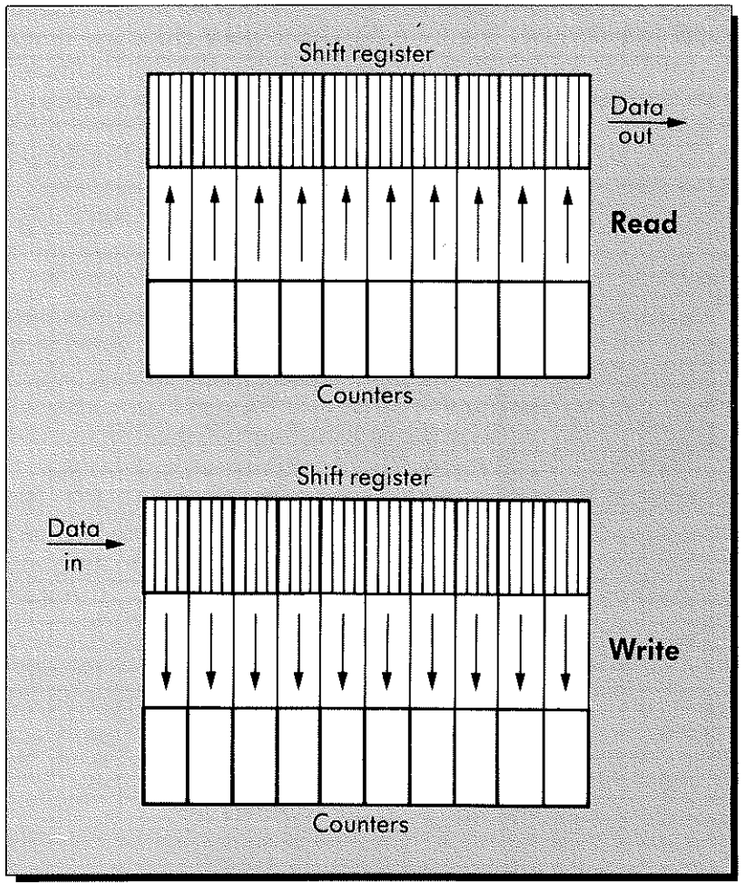

Figure 5-3. Read and write operations of the clock chip

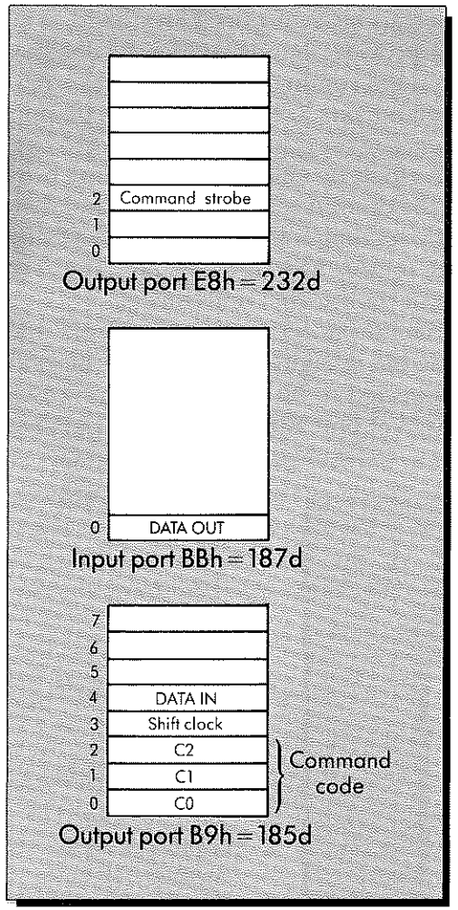

Figure 5-4. Ports for the clock

Command Code	Function
0	No operation
1	Serial transfer
2	Write
3	Read
4–7	Set TP timing

Table 5-1. Clock commands

A command strobe bit is connected to bit 2 of port E8h = 232d. The purpose of the command strobe is to provide "handshaking" for loading commands into the chip. Each pulse on the command strobe causes a new command to be read into the command bits C2, C1, and C0 in port B9h = 185d.

You can send commands to the clock by sending a byte out port B9h = 185d in which bits 2, 1, 0 form the desired command code; and then strobing the command into the chip by sending a byte to port E8h = 232d in which bit 2 is a one, and then a byte to port E8h in which bit 2 is a zero (see Figure 5-5). Later we will examine a routine that does this.

The forty-bit shift register has a data input bit, a data output bit, and data clock input bit. The data input bit is connected to bit 4 of port B9h = 185d, the data output bit is connected to bit 0 of port BBh = 187d, and the data clock bit is connected to bit 3 of port B9h = 185d. The purpose

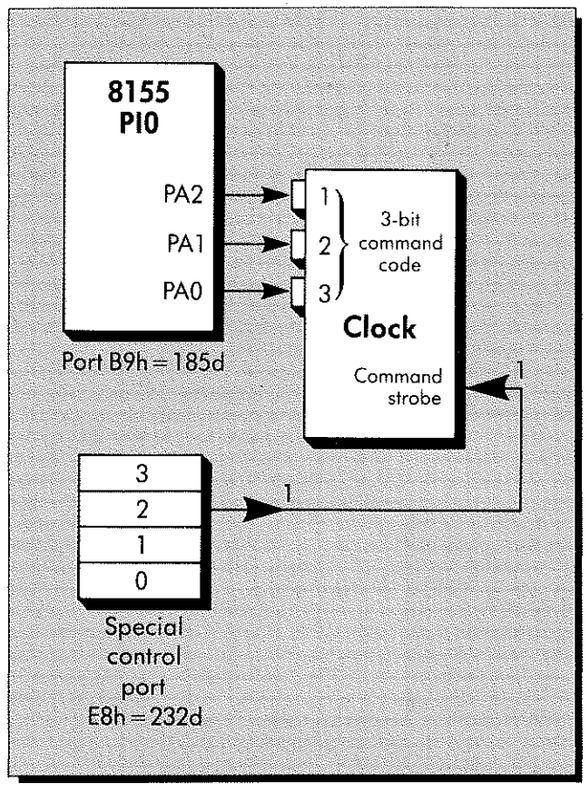

Figure 5-5. Sending a command

of the data clock is to control the serial shift operation. When the serial transfer mode has been selected, each pulse on the data clock bit forces the forty-bit shift register to shift by one place, shifting in one bit from the data input and shifting out one bit into the data output. We will describe this process further in the next two sections.

The ROM Routines

The ROM routines for the clock fall into four classes: primitive command routines, routines to set the clock, routines to read the clock, and the code for the background task.

The Primitive Command Routine

At the lowest level is a routine to send commands to the clock chip. This routine is located at 7383h = 29,571d of the Model 100's ROM (see box). In the next sections we'll see how this routine is used to read and set the time and date on the Model 100. You can also use this routine to control the clock directly from BASIC or machine language.

Routine: Clock Command

Purpose: To send a command to the clock

Entry Point: 7383h = 29,571d

Input: Upon entry, the A register must contain the code for the clock command. A value of 0 means no operation, a value of 1 means put the clock chip into register shift mode, a value of 2 means transfer the time and date from the shift register into the clock counters, and a value of 3 means read the time and date from the clock counters to the shift register.

Output: The clock is programmed accordingly.

BASIC Example:

```
CALL 29571,A
```

where A is the clock command.

Special Comments: This is the most primitive level of programming the clock.

What Happens When You Read the Time or Date

Let's begin with the routines to read the clock, starting at the highest levels (BASIC commands) and working our way down to the actual clock chip routines. The BASIC variables TIME$ and DATE$ are used to read and set the time and date. To read the time or date from BASIC (while running a program or in command mode), you must cause the corresponding variable to be "evaluated". This means that the variable must be part of an expression that might appear on the right side of an equals sign, in an IF clause, or in a PRINT statement.

You can use the information presented in this section to write your own BASIC or machine-language programs to read the clock chip.

The Time

Like the LCD routines to print a character discussed in the last chapter, the time and date routines are composed of a number of different levels. The highest level is designed to execute BASIC commands or functions, while the lowest level directly controls the physical device — in this case, the clock chip. The levels are numbered from top to bottom: the highest level is called level one, the next lower level is called level two, and so on.

The ROM routine to "evaluate" the BASIC variable TIME$ is located at 1904h = 6404d (see box). This is the level 1 time-reading routine. It calls a level 2 time-reading routine and then stores the resulting time in the proper location.

Routine: Read Time — BASIC Command (Level 1)

Purpose: To read the time from the clock chip

Entry Point: 1904h = 6404d

Input: None

Output: When the routine returns, the time is stored as a string whose address is stored at location FB8Ah = 64,394d. Some other locations used in handling string variables are also affected.

BASIC Example:

```
CALL 6404
```

Special Comments: Every time this call is made, a three-byte string descriptor is placed in memory, starting at FB6Eh = 64366d. This can be done only eight times before BASIC runs out of room and declares a ST (string too complex) error.

Here is a BASIC program that exercises the level 1 time-reading routine. It displays a number of locations that are affected by this routine. It even displays the time string itself.

```
100 ' TEST LEVEL 1 TIME READING
110 '
120   CALL 6404
130   X0=PEEK(64393)
140   X1=PEEK(64394)+256*PEEK(64395)
150   X2=PEEK(64396)+256*PEEK(64397)
160   Y =PEEK(64361)+256*PEEK(64362)
170   Z0=PEEK(Y-3)
180   Z1=PEEK(Y-2)+256*PEEK(Y-1)
190   PRINT USING "##";X0;
200   PRINT USING "######";X1;
210   PRINT USING "######";X2;
220   PRINT USING "######";Y;
230   PRINT USING "##";Z0;
240   PRINT USING "######";Z1;
250   PRINT " ";
260   FOR I = Z1 TO Z1+7
270    PRINT CHR$(PEEK(I));
280   NEXT I
290   PRINT
300   GOTO 120
```

On line 120 of this program, the level 1 time-reading routine is called. On lines 130-180, we PEEK at various values, and on lines 190-290 the values are assembled into a display line on the screen.

Let's examine the various PEEKs. The contents of location FB89h = 64,393d is placed in the variable X0. The value is 8, which is the length of the time string. The contents of locations FB8Ah = 64394d and FB8Bh = 64395d form a 16-bit integer that is placed in the variable X1. This value is the address of the time string. The contents of locations FB8Ch = 64396d and FB8Dh = 64397d form a 16-bit integer that is placed in the variable X2. This value is always one less than X1. The contents of locations FB69h = 64361d and FB6Ah = 64362d form a 16-bit integer that is placed in the variable Y. This value points just beyond the three-byte descriptor for the time string. Location Y-3 contains the length of the string and is placed in Z0. Locations Y-2 and Y-1 form a 16-bit integer that is placed in the variable Z1. This is also the address of the string.

The display line first shows the value of X0, then X1, then X2, then Y, then Z0, then Z1, and finally the contents of the string at Z1 through Z1 + 7.

Returning to the ROM routines, we find that the level 2 routine to read the time is located at 190Fh = 6415d (see box). It calls a level 3 time routine,

which reads the raw time and date data into a 10-byte area of memory starting at F923h = 63,779d (see Figure 5-6). Each set of four bits from the shift register is placed in a different byte of memory in the order it comes out of the shift register. Once the raw data is loaded into memory, the level 2 routine turns the time part of this raw data into a string of ASCII numerals with the hours, minutes, and seconds separated by colons. Then it calls a routine at 1996h = 6550d (see box) to fetch the digits and put them in the string.

Routine: Read Time — Level 2

Purpose: To read the time from the clock chip

Entry Point: 190Fh = 6415d

Input: Upon entry, the HL register pair contains the address of an eight-byte area of memory where the string will be stored.

Output: When the routine returns, the time string is stored in the eight-byte area of memory.

BASIC Example:

```
CALL 6415,0,H
```

where H points to an area of memory where the time string will be stored.

Special Comments: None

Here is a BASIC program that explores the level 2 time-reading routine. It places the time in a string variable that we have under our control. Then it repeatedly calls the time routine and prints out the contents of this string variable.

```
100 ' LEVEL 2 TIME READING
110 '
120   T$ = "        "
130   T  = VARPTR(T$)
140   H  = PEEK(T+1)+256*PEEK(T+2)
150   CALL 6415,0,H
160   PRINT T$
170   GOTO 150
```


Looking more closely at this program, we see that on line 120, some blank space is reserved in the string variable T$. In line 130 we find the address of the three-byte string descriptor for T$. In line 140, the address where the string is actually located is given. In line 150, the level 2 time-reading routine is called to dump the time into T$, using the variable H to pass the address. In line 160, T$ is printed. Line 170 loops around to line 150, where the CALL to the time routine is.

The level 3 time-reading routine is located at 19A0h = 6560d (see box). It points to the raw time and date data by placing the address F923h = 63,779d in the HL register, disables interrupts, calls a level 4 routine to get the raw time and date data, and then enables the interrupts before returning.

Routine: Read Time and Date — Level 3

Purpose: To read the time and date from the clock chip

Entry Point: 19A0h = 6560d

Input: None

Output: When the routine returns, the raw time data is in a ten-byte area of memory starting at F923h = 63,779d.

BASIC Example:

```
CALL 6560
```

Special Comments: None

Here is a BASIC program that illustrates the level 3 time and date reading routine. It loops around and around, calling the level 3 time and date reading routine and then printing out the raw time and date data.

```
100 ' LEVEL 3 TIME AND DATE READING
110 '
120   CALL 6560
130   FOR I = 63779 TO 63788
140    PRINT PEEK(I);
150   NEXT I
160   PRINT CHR$(13);
170   GOTO 120
```

Looking more closely, on line 120, the level 3 time and date reading routine is called. On lines 130-150 the raw time and date data are displayed. The first digit is the units digit of the seconds, the second digit is the tens

digit of the seconds, the third digit is the units digit of the minutes, the fourth digit is the tens digit of the minutes, the fifth digit is the units digit of the hours, the sixth digit is the tens digit of the hours, the seventh digit is the units digit of the day of the month, the eighth digit is the tens digit of the day of the month, the ninth digit is the the day of the week, and the tenth digit is the month in hex.

On line 160, a carriage return (ASCII 13) is printed, returning the cursor to the beginning of the display line. This keeps the display on one display line rather than producing a long sequence of lines of output that scroll by. On line 170 the program loops back to the CALL command.

The level 4 time-reading routine is located at 7329h = 29,481d (see box). This is in the higher area of the ROM, where other low-level, machine-dependent routines are also located.

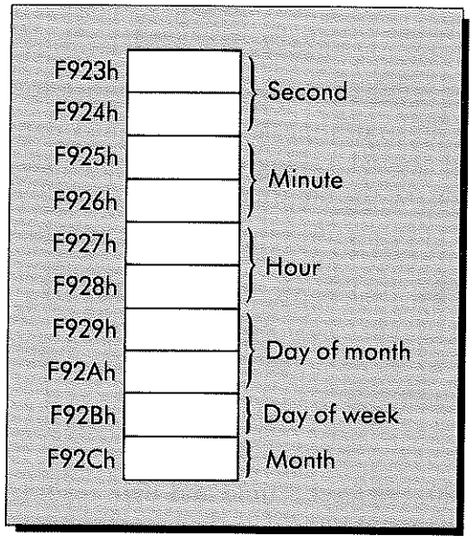

Figure 5-6. Raw time and date data

Routine: Read Time and Date — Level 4

Purpose: To read the time and date from the clock chip

Entry Point: 7329h = 29,481d

Input: Upon entry, the HL register pair points to a ten-byte area of memory.

Output: When the routine returns, the raw time and date data are in the ten-byte area of memory.

BASIC Example:

```
CALL 29481,0,H
```

Special Comments: None

We have a BASIC program that illustrates this level as well. It sets up a string to store the raw time and date data, calls the level 4 routine to dump the raw time and date in this string, and then prints out the raw data.

```
100 ' LEVEL 4 TIME AND DATE READING
110 '
120   T$ = "          "
130   T  = VARPTR(T$)
140   H  = PEEK(T+1)+256*PEEK(T+2)
150   CALL 29481,0,H
160   FOR I = H to H+9
170    PRINT PEEK(I);
180   NEXT I
190   PRINT
200   GOTO 150
```

This program is a combination of the previous two programs. On lines 120-140, it sets up the string T$ for storage. On line 150, it calls the level 4 time and date reading routine. On lines 160-190, it prints out the raw data. On line 200, it loops back for more.

First the level 4 routine strobes the read command into the chip by putting the value 3 into the A register and calling the clock command routine at 7383h = 29,571d (discussed in the previous section). In response to this command, the chip does the parallel transfer from the time and date counters to the forty-bit shift register.

Next the level 4 routine strobes the serial transfer command into the chip by calling the clock command routine again, this time with 1 in the A register. Now the system is ready to read the time and date bit by bit.

Each bit of the time and date is obtained by capturing bit 0 of port BBh = 187d. The IN BBh command gets the byte into the A register, the A register is shifted one place to the right to put the bit into the carry flag, and the contents of the carry is shifted into the D register with three more instructions. After each bit is read, the shift register is shifted by strobing the data clock (bit position 3 of register B9h = 185d). This strobe is accomplished by moving a byte into register B9h = 185d that has a zero in bit position 3; then a byte that has a one in this bit position is moved into this same port. The data bits from the shift register are collected in groups of four, one group for each of the ten digits of the raw time and date data. After all the bits are read, a "no operation" command is strobed into the chip by calling the clock command routine with zero in the A register. This stops the serial transfer.

The Date

The routine to "evaluate" the BASIC string variable DATE$ (see box) works in much the same way as the TIME$ routine. It is located at 1924h = 6436d. It calls a level 2 date routine and stores the resulting string in the appropriate place.

Routine: Read Date — BASIC Command (Level 1)

Purpose: To read the date from the clock chip

Entry Point: 1924h = 6436d

Input: None

Output: When the routine returns, the date is stored as a string whose address is placed at location FB8Ah = 64,394d. Some other locations used in handling string variables are also affected.

BASIC Example:

```
CALL 6436
```

Special Comments: As with the TIME$ routine, every time this call is made, a three-byte string descriptor is placed in memory, starting at FB6Eh = 64366d. This can be done only eight times before BASIC runs out of room and declares a ST (string too complex) error.

Here is a BASIC program that exercises the level 1 date-reading routine. It is almost identical to the level 1 time-reading program given previously: the only difference is the address of the routine.

```
100 ' TEST LEVEL 1 DATE READING
110 '
120   CALL 6436
130   X0=PEEK(64393)
140   X1=PEEK(64394)+256*PEEK(64395)
150   X2=PEEK(64396)+256*PEEK(64397)
160   Y =PEEK(64361)+256*PEEK(64362)
170   Z0=PEEK(Y-3)
180   Z1=PEEK(Y-2)+256*PEEK(Y-1)
190   PRINT USING "##";X0;
200   PRINT USING "#######";X1;
210   PRINT USING "#######";X2;
220   PRINT USING "#######";Y;
230   PRINT USING "##";Z0;
240   PRINT USING "#######";Z1;
250   PRINT " ";
260   FOR I = Z1 TO Z1+7
270    PRINT CHR$(PEEK(I));
280   NEXT I
290   PRINT
300   GOTO 120
```

Since this is almost the same as the earlier program, we will not discuss its structure.

The level 2 date routine (see box) at 192Fh = 6447d calls the level 3 routine at 19A0h = 6560d to read the time and date described above. Recall that this routine puts the raw time and date data in a ten-byte area of RAM starting at location F923h = 63,779d. After this, the level 2 date routine uses the raw data to create a string with the month, day, and year in ASCII numerals separated by slashes. The month has to be dealt with in a slightly different way than the other parts of the date because it is expressed in hexadecimal notation rather than decimal.

Routine: Read Date — Level 2

Purpose: To read the date from the clock chip

Entry Point: 192Fh = 6447

Input: Upon entry, the HL register pair contains the address of an eight-byte area of memory where the string will be stored.

Output: When the routine returns, the date string is stored in the eight-byte area of memory.

BASIC Example:

```
CALL 6447,0,H
```

where H points to an area of memory where the date string will be stored.

Special Comments: None

Here is a BASIC program that illustrates the level 2 date-reading routine. It is the same as the one for the level 2 time-reading routine except for line 150, where the level 2 date routine is called instead of the level 2 time routine.

```
100 ' LEVEL 2 DATE READING
110 '
120   T$ = "        "
130   T  = VARPTR(T$)
140   H  = PEEK(T+1)+256*PEEK(T+2)
150   CALL 6447,0,H
160   PRINT T$
170   GOTO 150
```


The Day of the Week

There is also a routine to "evaluate" the BASIC string variable DAY$, which returns the day of the week as a three-character string (see box). The routine is located at 1955h = 6485d. It calls a level 2 day routine and places the resulting string in the DAY$ variable.

Routine: Read Day of Week — BASIC Command (Level 1)

Purpose: To read the day of the week from the clock chip

Entry Point: 1955h = 6485d

Input: None

Output: When the routine returns, the day is stored as a string whose address is placed in location FB8Ah = 64,394d. Some other locations used in handling string variables are also affected.

BASIC Example:

```
CALL 6485
```

Special Comments: Every time this call is made, a three-byte string descriptor is placed in memory, starting at FB6Eh = 64,366d. This can be done only eight times before BASIC runs out of room and declares a ST (string too complex) error.

The level 2 day routine is located at 1962h = 6498d (see box). It calls the level 3 time and date reading routine at 19A0h = 6560d (described previously). It then plucks out the numerical value for the day of the week from the raw time and date data and looks at the corresponding three-byte string in a table starting in the ROM at 1978h = 6520d.

Routine: Read Day of Week — Level 2

Purpose: To read the day of the week from the clock chip

Entry Point: 1962h = 6498d

Input: Upon entry, the HL register pair contains the address of a three-byte area of memory where the string will be stored.

Output: When the routine returns, the day string is stored in the three-byte area of memory.

BASIC Example:

```
CALL 6498,0,H
```

where H points to an area of memory where the day string will be stored.

Special Comments: None

It is easy to modify the level 1 and level 2 BASIC programs given for the time and date so that they can work for the day of the week. We invite you to do that yourself.

What Happens When You Set the Time or Date

To set the time, date, or day of the week you must assign an appropriate string expression to the corresponding variable. This happens when the variable TIME$, DATE$, or DAY$ appears on the left side of an equals sign.

The Time

The BASIC routine to set the time (see box) is the reverse of the routine discussed above to evaluate TIME$. The level 1 routine is located at 19B0h = 6576d. It checks for an equals sign following the TIME$ symbol. Then it calls a routine at 1A42h = 6722d (see box), which evalutes the string expression on the right side of the equals sign, calls the level 3 time and date reading routine to read the raw time and date data from the clock into RAM starting at F923h = 63,779d, and then replaces the time part of the raw data by the appropriate new data. The level 1 routine finishes by calling a level 2 time and date setting routine to place the modified raw data back into the clock.

Routine: Set Time — BASIC Command (Level 1)

Purpose: To set the time on the clock chip

Entry Point: 19B0h = 6576d

Input: Upon entry, the HL register pair points to a command line containing a string expression that evaluates a time string.

Output: When the routine returns, the specified time is set in the clock chip.

BASIC Example:

```
CALL 6576,0,H
```

where H is the address of a valid time string.

Special Comments: None

Here is a BASIC program that shows how to use the level 1 time-setting command. It contains an infinite loop that gets a time string from the user, sends it to the time-setting routine, and then displays the new time using the BASIC TIME$ function to verify that the time has actually been set. When you run it, be sure to type in the time in exactly the same format as the Model 100 prints out the time.

```
100 ' LEVEL 1 TIME SET
110 '
120   INPUT "TIME";T$
130   S$=CHR$(221)+"T$"+CHR$(0)
140   S = VARPTR(S$)
150   H = PEEK(S+1)+256*PEEK(S+2)
160   CALL 6576,0,H
170   PRINT TIME$
180   GOTO 120
```

The main loop of this program extends over lines 120-180. On line 120, the user inputs the time into the variable T$. On line 130, the time is encased in a command line and stored in the variable S$. In lines 140-150, the address of the string is computed and stored in the variable H. On line 160, the time-setting routine is called. On line 170, the program verifies that the time has been set by printing out the TIME$ variable. On line 180 it loops back for another time.

Routine: Get Time String from Command Line

Purpose: To get the time string from the command line

Entry Point: 1A42h = 6722d

Input: Upon entry, the HL register pair points to a command line containing a valid time string.

Output: When the routine returns, the new raw time and date data are stored in a ten-byte area in memory starting at F923h.

BASIC Example: Not directly applicable

Special Comments: This routine uses the stack and therefore cannot be CALLed directly from BASIC.

The level 2 time and date setting routine is located at 732Ah = 29,482d, in the high part of the ROM, and thus is considered a low-level, machine-dependent routine. It shares much of its code with the level 4 time and date reading routine at 7329h = 29,481d (described previously). However, it first strobes the serial transfer command into the chip (by calling the command routine at 7383h = 29,571d with 1 in the A register). Then it transfers each bit of the time and date in through bit position 2 of port B9h = 185d, strobing the data clock (bit 3 of port B9h = 185d) each time. It finishes by strobing the write command and then the no operation command into the chip.

The Date

The BASIC routine to set the date (see box) works in a similar way. It is located at 19BDh = 6589d and calls many of the same routines.

Routine: Set Date — BASIC Command (Level 1)

Purpose: To set the date on the clock chip

Entry Point: 19BDh = 6589d

Input: Upon entry, the HL register pair points to a command line containing a string expression that evaluates to a date string.

Output: When the routine returns, the specified date is set in the clock chip.

BASIC Example:

```
CALL 6589,0,H
```

where H is the address of a valid date string.

Special Comments: None

Here is a BASIC program that shows how to use the level 1 date-setting command. When you run it, be sure to type in the date in exactly the same format as the Model 100 prints out the date. Because it is almost the same as the level 1 time-setting program described earlier, we will not discuss it in detail.

```
100 ' LEVEL 1 DATE SET
110 '
120   INPUT "DATE";T$
130   S$=CHR$(221)+"T$"+CHR$(0)
140   S = VARPTR(S$)
150   H = PEEK(S+1)+256*PEEK(S+2)
160   CALL 6589,0,H
170   PRINT DATE$
180   GOTO 120
```

The Day of the Week

There is also a BASIC routine to set the day of the week (see box), located at 19F1h = 6641d. You might want to modify the program above to work for this DAY$ function.

Routine: Set Day of Week — BASIC Command (Level 1)

Purpose: To set the day of week on the clock chip

Entry Point: 19F1h = 6641d

Input: Upon entry, the HL register pair points to a command line containing a string expression that evaluates to a day string.

Output: When the routine returns, the specified day of the week is set in the clock chip.

BASIC Example:

```
CALL 6641,0,H
```

where H is the address of a valid day string.

Special Comments: None

BASIC Time Interrupt Commands

BASIC has certain commands that allow you to make your portable computer into a fancy alarm clock or a controller for lab equipment. These commands are ON TIME$...GOSUB, TIME$ ON, TIME$ OFF, and TIME$ STOP. They control a BASIC interrupt that is triggered by the time of day.

The ON TIME$...GOSUB command allows you to specify a time and a BASIC subroutine that you want called at that specified time. The routine to handle this command starts at location 1B0Fh = 6927d (see box). It calls a subroutine at 1AFCh = 6908d (see box) that sets the time for the interrupt. This subroutine calls a routine at 1A42h = 6722d (described previously) to get the time from your BASIC command line and transform it into raw form. Then it calls a block-move routine at 3469h = 13,417d (see box) to transfer it into a six-byte area of RAM starting at F93Dh = 63,805d. In the next section we will see how the clock-cursor-keyboard background task continually examines this six-byte area, looking for a match with the current time.

Routine: ON TIME$...GOSUB — BASIC Command

Purpose: To set the ON TIME$ interrupt

Entry Point: 1B0Fh = 6927d

Input: Upon entry, the HL register pair contains the address of the end of a command line for the ON TIME$ command.

Output: When the routine returns, the location of the ON TIME$ subroutine and the time that it should be executed are loaded into BASIC.

BASIC Example:

```
CALL 6927,0,H
```

where H is the address of the end of the ON TIME$ command.

Special Comments: None

Routine: Block Transfer

Purpose: To transfer bytes from one location to another location in memory

Entry Point: 3469h = 13,417d

Input: Upon entry, the B register contains the number of bytes to be tranferred, the DE register pair points to the source location, and the HL register pair points to the destination location.

Output: When the routine returns, the bytes have been transferred.

BASIC Example: Not applicable

Special Comments: This routine is used many times throughout the Model 100's ROM.

The ON TIME$...GOSUB routine finishes by getting the location of the BASIC line specified after the GOSUB. It puts this location into a two-byte RAM location starting at F948h = 63,816d. This is part of a three-byte area of RAM starting at F947h = 63,815d, which stores information about the time interrupt.

The routines to handle the TIME$ ON, TIME$ OFF, and TIME$ STOP commands all start at 1AA5h = 6821d. Here the HL register is set to point to the first byte of the three-byte area of RAM starting at F947h = 63,815d, which stores information about the time interrupt. These routines call a routine at 1AEAh = 6890d, which in turn branches to individual routines to handle ON, OFF, or STOP (see boxes). These routines are also used to control other interrupts such as the ON KEY$ interrupts, which will be discussed in the next chapter.

Routine: Interrupt ON — BASIC Command

Purpose: To enable the interrupt

Entry Point: 3FA0h = 16,288d

Input: Upon entry, the HL register pair contains the address of the three-byte interrupt table.

Output: When the routine returns, the interrupt is enabled.

BASIC Example:

```
CALL 16288,0,H
```

where H is the address of the three-byte interrupt table.

Special Comments: None

Routine: Interrupt OFF — BASIC Command

Purpose: To disable the interrupt

Entry Point: 3FB2h = 16,306d

Input: Upon entry, the HL register pair contains the address of the three-byte interrupt table.

Output: When the routine returns, the interrupt is disabled.

BASIC Example:

```
CALL 16306,0,H
```

where H is the address of the three-byte interrupt table.

Special Comments: None

Routine: Interrupt STOP — BASIC Command

Purpose: To stop the interrupt

Entry Point: 3FB9h = 16,313d

Input: Upon entry, the HL register pair contains the address of the three-byte interrupt table.

Output: When the routine returns, the interrupt is stopped.

BASIC Example:

```
CALL 16313,0,H
```

where H is the address of the three-byte interrupt table.

Special Comments: None

The first byte of this interrupt storage area (at F947h = 63,815d) is called the "interrupt status byte". Three of its bits are used to manage the ON TIME$ interrupt. Bit 0 tells whether the interrupt is on or off, bit 1 tells whether the interrupt is stopped or not, and bit 2 tells whether the interrupt has occurred. There are six values that this byte normally takes on, depending upon the values of these bits (two more values are possible but never actually occur). Each value represents a *state* for the system with regard to the ON TIME$ interrupt. These states form a "finite state machine" (see Figure 5-7). This term is used by computer scientists to describe a system that has a finite number of states and a set of possible "transitions" between these states. It is a useful concept for understanding everything from the basic electrical circuits that make up a computer to the workings of programs like a sophisticated text editor. Computer scientists use diagrams such as the one in Figure 5-7 to help them visualize the workings of such systems.

Let's look at the various states of this "interrupt machine" in more detail. There are three primary states: 0, 1, and 5. A value of 0 (all bits off) indicates that the interrupt is *off* (cannot cause any action). A value of 1 (just bit 0 on) means that the interrupt is on but has not yet been "triggered" (the clock has not yet reached the time specified for the interrupt). A value of 5 means that the interrupt is on and has actually been triggered. When this happens, we say that the interrupt is *pending*.

For each of these three primary states there is a corresponding "stopped" state. In the stopped states, interrupts are "remembered" but not acted upon. A value of 2 indicates that the interrupt is off and stopped. A value

of 3 indicates that the interrupt is on but stopped. A value of 7 indicates that the interrupt has been triggered but is stopped.

The TIME$ ON, TIME$ OFF, and TIME$ STOP commands, as well as the actual triggering and processing of the ON TIME$ interrupt, cause state *transitions* within this finite state machine. For example, if the system is in state 0 (off), then the TIME$ ON command will cause it to move to state 1 (on). TIME$ OFF, on the other hand, causes the system to move to state 0 (off), no matter what state it was in previously. The arrows in the state diagram show all possible state transitions.

The ON part of the TIME$ routine (described previously) is located at 3FA0h = 16,288d; the OFF part is located at 3FB2h = 16,306d; and the

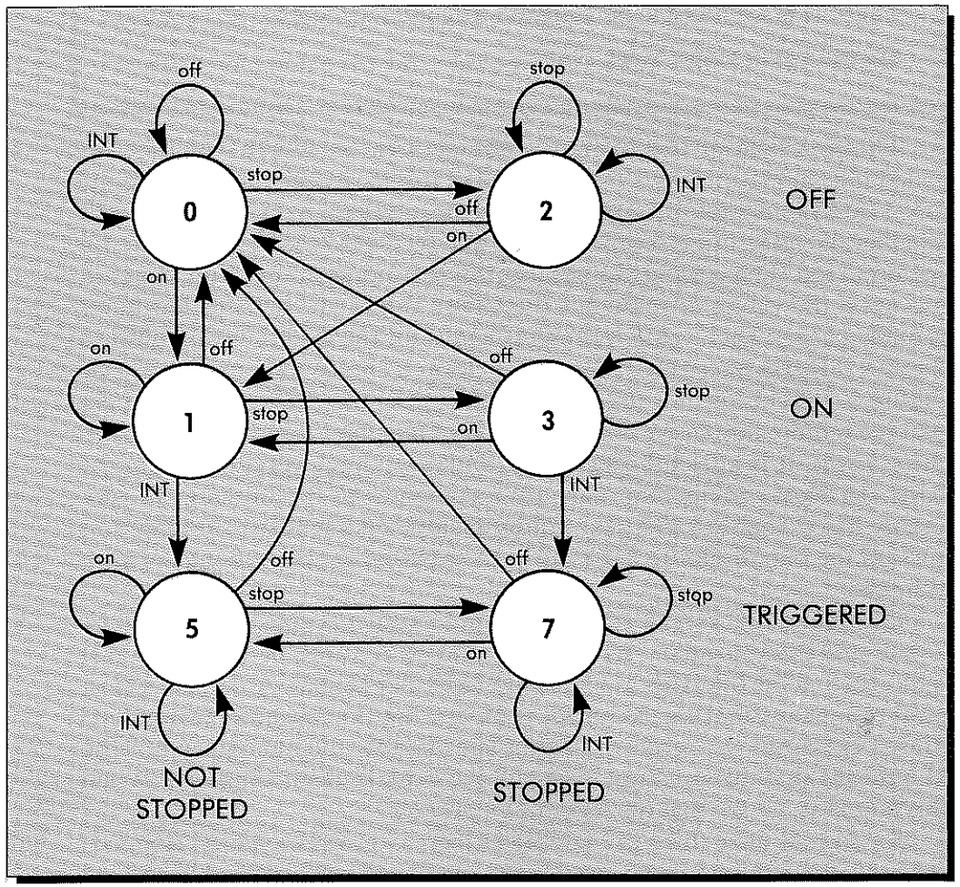

Figure 5-7. Finite state machine for interrupts

STOP part is located at 3FB9h = 16,313d. In addition, there is a routine at 3FD2h = 16,338d (see box), which adjusts this finite state machine each time an interrupt is triggered, and a routine at 3FF1h = 16,369d (see box), which adjusts the state each time the interrupt is processed.

Routine: Trigger Interrupt

Purpose: To trigger the interrupt

Entry Point: 3FD2h = 16,338d

Input: Upon entry, the HL register pair contains the address of the three-byte interrupt table.

Output: When the routine returns, the interrupt is triggered.

BASIC Example:

```
CALL 16338,0,H
```

where H is the address of the three-byte interrupt table.

Special Comments: None

Routine: Clear Interrupt

Purpose: To clear the interrupt

Entry Point: 3FF1h = 16,369d

Input: Upon entry, the HL register pair contains the address of the three-byte interrupt table.

Output: When the routine returns, the interrupt is cleared.

BASIC Example:

```
CALL 16369,0,H
```

where H is the address of the three-byte interrupt table.

Special Comments: None

Besides manipulating the interrupt status byte at F947h = 63,815d, these routines maintain an "interrupt counter byte" at F654h = 63,060d. This byte is incremented each time state 5 (interrupt pending) is entered and decremented each time it is exited (turned off, stopped, or processed). Unlike the status byte at F947h = 63,815d, this byte is shared with other BASIC interrupts such as the ON KEY, ON COM, and ON MDM interrupts. Thus this byte counts the total number of BASIC interrupts of *any* type that are pending in the computer. In Chapters 6 and 7 we will examine other BASIC interrupts.

The Clock-Cursor-Keyboard Background Task

The clock-cursor-keyboard background task helps maintain certain basic functions in the Model 100 computer. These include updating the clock, blinking the cursor, and scanning the keyboard. All these jobs must be performed very often; that is, several times a second to several times a minute. Fortunately they do not take much time to perform; thus, they do not appreciably slow down the main (foreground) tasks that the computer is asked to do.

The clock-cursor-keyboard background task involves the real-time clock in two ways. First, the clock drives the background task, causing it to be performed about 256 times a second; and secondly, the clock is read every 125 times that the background task is called, or about every half second. This is done to support the ON TIME$ interrupt. The clock part of the background task also maintains the automatic power-off feature and the year part of the date.

Generating the Interrupt

The timing pulse from the clock is initialized at 6CEBh = 27,883d in the warm start reset routine (see Chapter 3), which sends a command code 5 to the clock chip (via the clock command routine at 7383h = 29,571d, described previously. This causes the clock chip to pulse at a frequency of 256 times a second, or about once every 4 milliseconds.

The timing pulse is fed into the interrupt 7.5 input pin on the 8085 CPU. This CPU interrupt can be selected using the SIM instruction and enabled and disabled using the EI and DI instructions. In general, the SIM command is used to tell which of three interrupts (5.5, 6.5, and 7.5) will be enabled with the EI (enable interrupt) instruction. If the 7.5 interrupt is enabled, a timing pulse will cause the CPU to stop what it is doing and call the routine at 3Ch = 60d (see Chapter 3). In the Model 100, location 3Ch = 60d has some code that disables interrupts and jumps to 1B32h = 6962d, where the beginning of the clock-cursor-keyboard background task is located. Each time interrupt 7.5 is actuated, it must be rearmed before it can be used again. This is done by the routine at 765Ch = 30,300d (see Chapter 4), which uses the SIM command to turn off and rearm the interrupt. This routine was mentioned in Chapter 4 on the LCD.

The code for the background task comes in three major sections — one for the clock, one for the cursor, and one for the keyboard. We have already discussed the section for the cursor in Chapter 4, and we will discuss the section for the keyboard in Chapter 6. We will now discuss the section for the clock (see box).

The clock-cursor-keyboard background task begins at 1B32h = 6962d and consists of several subsections.

Routine: Clock Section of Background Task

Purpose: To update the system time for the ON TIME$ interrupt, maintain automatic power off, and update the year.

Entry Point: 1B32h = 6962d

Input: None

Output: System time and timing parameters are updated.

BASIC Example: Not applicable

Special Comments: None

The "Very Often" Routine

Before doing anything else, this task calls a routine located in RAM at F5FFh = 62,975d (see box). Normally, a RETurn instruction is located there, so nothing much happens, but if you place your own routine there, it will be called "very often". This "very often" routine can be used to run your own background task in addition to the background task that is already built into the computer. Each execution of the "very often" routine corresponds to one "tick" of the background task.

Routine: Very Often

Purpose: To perform one tick of a background task

Entry Point: F5FFh = 62,975d

Input: None, as it stands

Output: None, as it stands

BASIC Example:

```
CALL 62975
```

Special Comments: None

The bytes at F5FFh = 62,975d are some of the RAM locations that are initialized when the machine is first turned on. Initially, a RETurn followed by two NOP instructions is placed at F5FFh = 62,975d. To install your own "very often" routine replace these three bytes by a JuMP or CALL instruction to a routine that you have placed somewhere else in memory.

A stopwatch program is an example of a way the "very often" routine can be used. Such a program could be written in machine language and CALLed from BASIC. The main stopwatch program would set up a variable in memory for a count and use its own "very often" routine to increment this variable each time it is called. This "very often" would then count the ticks of the system clock (256 times a second). The main stopwatch program would have to monitor some keys on the keyboard (see Chapter 6) to tell when to zero the count, start and stop the count routine, and report the results.

Some Housekeeping

Let's contine the discussion of the Model 100's built-in background task. Right after the "very often" routine is called, the HL, DE, BC, A, and Flags are pushed onto the stack. This is because the background task is run as an interrupt routine and therefore must return with the contents of the CPU registers as they were upon entry. Note that if you install your own "very often" routine, you must make sure that it, too, does not modify any registers.

The next action of the background task is to allow certain interrupts and block others with the SIM command. These are 7.5 (the background task itself), which is blocked; 6.5 (the serial communications line), which is allowed; and 5.5 (the bar code reader), which is blocked. To do this, the A register is loaded with 00001101 binary before the SIM instruction is executed. The 1 in bit 3 indicates that the SIM command is being used to select interrupts; bits 2, 1, and 0 select the interrupts, as indicated above.

The Clock Section

The clock part of the clock-cursor-keyboard background task now starts. It begins by decrementing a counter (located at F92Fh = 63,791d) that causes the clock section to be executed about every half second. This works as follows: if the counter does not become zero, the clock section is skipped and the CPU goes onto the next (cursor) section of the background task. However, if the counter reaches zero, the CPU continues into the rest of the clock section, resetting the counter to 125. Since the interrupt happens about every 4 milliseconds, the cycle length of this process is about half a second.

If the rest of the clock section is executed, another counter (at F930h = 63,792d) is decremented. This counter is reset to 12 each time it reaches zero, giving a six-second timing cycle to the section of code immediately following. This particular code takes care of the Model 100's automatic power-off feature.

The Automatic Power Off

To conserve battery power, the Model 100 has an automatic power-off feature. Normally, if the machine is left alone and it is not running a program, it will shut off after about ten minutes. Fortunately, the RAM memory stays on even when the automatic power-off shuts down the rest of the computer, so you will never lose any work because of this feature.

The automatic power-off code starts at 1B4Eh = 6990d. It checks to see if you are running a BASIC program by looking at the current line number (stored at F67Ah = 63,098d). If this variable is not −1, it assumes you are running a program and renews the power-off timer counter (at F631h = 63,025d) by calling a routine at 1BB1h = 7089d (see box). The initial value for the count is stored in F657h = 63,063d.

Routine: Renew Automatic Power-Off Timer

Purpose: To reinitialize the power-off timer

Entry Point: 1BB1h = 7089d

Input: None

Output: When the routine returns, the contents of location F657h = 63,063d (the full count) are moved to location F931h = 63,793d (the counter).

BASIC Example: Not applicable (happens while BASIC is running anyway).

Special Comments: None

The power-off timer counter is decremented each time the automatic power-off code is executed. Since this code is executed about every six seconds or every tenth of a minute, and since the default count value is 100, the default time for power-off is about ten minutes. If the counter is already zero, it is not decremented, and the CPU is sent on past the power-off code. This is designed to handle the case when the user disables the power-off feature with the command POWER CONT.

Location F932h = 63,794d is used to tell the rest of the system when the power should be turned off. If the power-off timer actually decrements from 1 to 0, location F932h = 63,794d is set equal to −1. The main input routine of BASIC (in particular at 1358h = 4952d) examines this location and turns off the power if it becomes nonzero. The actual code to turn off the power is at 143Fh = 5183d.

Detecting the ON TIME$ Interrupt

After the automatic power-off code, the raw time and date data from the clock are read into a ten-byte area of memory starting at F933h = 63,795d. The low-level time and date routine at 7329h = 29,481d, which we discussed earlier, is used.

Next the current time raw data are compared with the time that was specified by the last ON TIME...GOSUB command. If there is no match, it proceeds; otherwise, it calls a routine at 3FD2h = 16,338d (described previously) to further handle the time interrupt. This routine is part of the finite state machine and thus operates upon the time interrupt status byte at F947h = 63,815d and the interrupt counter at F654h = 63,060d, causing them to indicate that the ON TIME$ interrupt has just been triggered.

Updating the Year

The next section takes care of updating the year. It gets the month from where it was just stored by the low-level time and date routine. It puts this in location F655h = 63,061d. If the value is less than or the same as what was previously stored there, the routine proceeds; otherwise, it increments the year stored at F92Dh = 63,789d and F92Eh = 63,790d.

Finally, the clock section checks the optional I/O and jumps to the cursor section.

Summary

In this chapter, we have explored the operation of the real-time clock and the routines that control it. Among these are routines to set and read the time of day, the date, and the day of the week. We have also studied routines to control the ON TIME$ interrupts. Finally, we have studied the background task, which performs a number of functions that have to be performed very often. These include handling the automatic power-off feature, updating the time for the ON TIME$ interrupt, updating the year, and performing other tasks such as cursor blinking (see Chapter 4) and keyboard scanning (see Chapter 6).