# 13 The Liquid-Crystal Display Screen

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

## How Liquid Crystals Work

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

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

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

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

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

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

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

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

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

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

## CPU Control of the Screen

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

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

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

## Character Formation

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

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

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

## Formation of Character Shapes

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

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

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

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

**Figure 13.1.** Lower-case “y”

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

| Unshifted | | SHIFTed | | GRPH | | SHIFT-GRPH | | CODE | | SHIFT-CODE | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| --- | - | --- | - | --- | - | --- | - | --- | - | --- | - |
| 39 | ' | 34 | " | 140 | ≈ | | | 160 | ´ | 164 | ` |
| 44 | , | 60 | < | 153 | ↓ | 248 | ┴ | 188 | ü | 221 | Ü |
| 45 | - | 95 | _ | 92 | \ | 124 | ¦ | 197 | ^ | 167 | ▼ |
| 46 | . | 62 | > | 151 | % | 247 | ┘ | 207 | ö | | |
| 47 | / | 63 | ? | 138 | ≠ | | | 174 | ½ | | |
| 48 | 0 | 41 | ) | 125 | } | | | 175 | ¶ | 166 | ° |
| 49 | 1 | 33 | ! | 136 | ¦ | 225 | ▘ | 192 | â | 208 | Â |
| 50 | 2 | 64 | @ | 156 | ♣ | 226 | ▝ | | | | |
| 51 | 3 | 35 | # | 157 | ◇ | 227 | ▖ | 193 | ê | 209 | Ê |
| 52 | 4 | 36 | $ | 158 | ♡ | 228 | ▗ | | | | |
| 53 | 5 | 37 | % | 159 | ♠ | 229 | ▚ | | | | |
| 54 | 6 | 94 | ^ | 180 | ¢ | 230 | ▞ | | | | |
| 55 | 7 | 38 | & | 176 | ¥ | | | 196 | û | 212 | Û |
| 56 | 8 | 42 | * | 163 | £ | | | 194 | î | 210 | Î |
| 57 | 9 | 40 | ( | 123 | { | | | 195 | ô | 211 | Ô |
| 59 | ; | 58 | : | 146 | ‡ | 245 | │ | 173 | ¼ | | |
| 61 | = | 43 | + | 141 | ± | | | 190 | ¨ | 168 | † |
| 91 | [ | 93 | ] | 96 | ` | 126 | ~ | 181 | ~ | | |
| 97 | a | 65 | A | 133 | ✝ | 235 | ▛ | 182 | ä | 177 | Ä |
| 98 | b | 66 | B | 149 | ♫ | | | | | | |
| 99 | c | 67 | C | 132 | ♣ | 255 | ▒ | 162 | ç | 171 | ▣ |
| 100 | d | 68 | D | | | 237 | ▙ | 187 | é | 215 | É |
| 101 | e | 69 | E | 143 | ◂ | 233 | ▌ | 198 | è | 214 | È |
| 102 | f | 70 | F | 130 | ▷ | 238 | ▟ | | | 191 | ƒ |
| 103 | g | 71 | G | | | 253 | ◥ | | | | |
| 104 | h | 72 | H | 134 | ☗ | 251 | ◤ | | | | |
| 105 | i | 73 | I | 142 | ¦ | 243 | ┬ | 199 | ï | 213 | Ï |
| 106 | j | 74 | J | | | 244 | ├ | 203 | ú | 219 | Ú |
| 107 | k | 75 | K | 155 | ← | 250 | ┼ | 201 | í | 217 | Í |
| 108 | l | 76 | L | 154 | → | 249 | ┤ | 202 | ó | 218 | Ó |
| 109 | m | 77 | M | 129 | ⌂ | 246 | └ | | | 165 | µ |
| 110 | n | 78 | N | 150 | ⚹ | | | 205 | Ñ | | |
| 111 | o | 79 | O | 152 | ↑ | 242 | ┐ | 183 | ö | 178 | Ö |
| 112 | p | 80 | P | 128 | ☎ | 241 | ─ | 172 | ¾ | | |
| 113 | q | 81 | Q | 147 | ¥ | 231 | ▀ | 200 | á | 216 | Á |
| 114 | r | 82 | R | 137 | ♪ | 234 | ▐ | | | 170 | ◘ |
| 115 | s | 83 | S | 139 | Σ | 236 | ▜ | 169 | § | 185 | ß |
| 116 | t | 84 | T | 135 | ▣ | 252 | ◢ | | | 186 | ™ |
| 117 | u | 85 | U | 145 | Å | 240 | ┌ | 184 | ü | 179 | Ü |
| 118 | v | 86 | V | | | | | 189 | è | 222 | È |
| 119 | w | 87 | W | 148 | ✱ | 232 | ▄ | | | | |
| 120 | x | 88 | X | 131 | ♂ | 239 | █ | 161 | à | 223 | À |
| 121 | y | 89 | Y | 144 | Ā | 254 | ◣ | 204 | ÿ | 220 | Ÿ |
| 122 | z | 90 | Z | | | 224 | | 206 | ã | | |

**Table 13.1.** LCD characters

## RAM Locations Relating to the Display

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

**Table 13.2.** Display variables in RAM

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

## Published ROM Subroutine Calls

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

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

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

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

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

**Table 13.4.** LCD Escape Sequences.

| Name | Call | Hex | Chr | ROM Address | Function |
|---|---|---|---|---|---|
| | | 41 | A | 4469 | Up one line unless already at edge |
| | | 42 | B | 446E | Down one line unless already at edge |
| | | 43 | C | 4453 | Right one space unless already at edge |
| | | 44 | D | 445C | Left one space unless already at edge |
| | | 45 | E | 4548 | Same as printing 0C-clears screen |
| | | 48 | H | 44A8 | Same as printing 0B-moves cursor to 1,1 |

*continued on following page*

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

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

* Radio Shack incorrectly lists this as ESC Y.

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

## How to Send Special Characters

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

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

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

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

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

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

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

## Sending Characters to the Printer

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

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

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

## How to Call 4B44

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

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

## Other Published LCD ROM Routines

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

## Cursor Position Routines

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

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

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

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

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

## Unpublished ROM Routines for the LCD

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

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

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

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

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

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