

LCD Graphics Display Input/Output Tips
Mike Safonov 70506,1473

The readily available information on techniques for manipulating, saving and loading screen graphics and custom characters to and from the LCD is precious little. Following is a discussion of various techniques including some little-known but useful ROM calls for manipulating, displaying and retrieving 6x8 blocks of LCD graphics data. The discussion is organized as follows:

- I. THEORY: HOW GRAPHICS DATA IS DISPLAYED ON AND READ FROM THE LCD
- II. USEFUL LCD ROM CALLS FOR GRAPHICS READ AND WRITE
- III. PRINTER INTERFACING: UPSIDE-DOWN BIT-IMAGE PROBLEMS

I. THEORY: HOW GRAPHICS DATA IS DISPLAYED ON AND READ FROM THE LCD

Each 6x8 character location of the screen is represented by six one-byte (i.e., six eight-bit) numbers ranging from 0 to 255. The six numbers contain the data for the six columns of a the character. For example the letter T is printed when the computer sends the following six decimal numbers to the LCD:

Decimal:	1	1	127	1	1	0
Binary: bit 0	1	1	1	1	1	0
bit 1	0	0	1	0	0	0
bit 2	0	0	1	0	0	0
bit 3	0	0	1	0	0	0
bit 4	0	0	1	0	0	0
bit 5	0	0	1	0	0	0
bit 6	0	0	1	0	0	0
bit 7	0	0	0	0	0	0

Note how the ones in the binary representation of the six numbers form the letter T. Each bit of the binary number represents one pixel. The bit is one is the pixel is set and it is zero otherwise.

Pixel data is sent to and from the LCD one byte at a time, with each byte containing the data for one eight-bit tall column of pixels. The actual mechanism for sending and retrieving this data to and from the screen involves the use of the use of CPU ports 185, 186, 254, and 255. It is somewhat complex, but not altogether incomprehensible. The process proceeds in the following sequence: First, a short machine code sequence (CALL 30300) is used to disable interrupts until the transfer of data to/from the LCD is complete. Second, ports 185 and 186 are used to select one of ten hardware drivers, each of which addresses a subset of the 6x320 bytes that make-up the LCD. The portions of the LCD controlled by each of the drivers 0-9 is indicated by the following diagram:

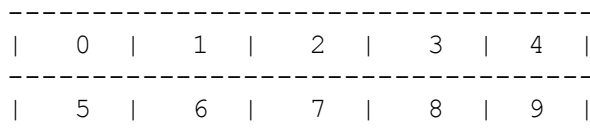


Fig. 1 Portions of LCD controlled by each of the 9 hardware drivers

A BASIC subroutine for selecting driver DZ% (where DZ% is the driver number 0 to 9) is:

```

10 CALL 30300
20 M%=2^DZ%
30 M1=PEEK(VARPTR(M%)) : REMARK M1=((2^N) AND 255)
40 M2=PEEK(VARPTR(M%)+1) : REMARK M2=INT(2^(N-8))
50 OUT 185,M1
60 P=(INP(186) AND 252) OR M2
70 OUT 185,P
80 RETURN

```

There is a ROM call at 30011 which implements lines 30-70 in machine code. An equivalent program is:

```

10 CALL 30300
20 M%=2^DZ%
30 CALL30011,0,VARPTR(M%)
80 RETURN

```

After one of the ten drivers has been selected, OUT254,xxx is used to select which eight-pixel-tall byte in the driver is to be used. The portion of

the LCD associated with each of the ten drivers is 32 pixels high and 50 pixels wide (except drivers 4 and 9 for which the width is only 40). The height of 32 pixels is divided into four rows (numbered 0 to 3) of eight-pixel-tall columns.

Each eight pixel tall column in a row contains one byte of LCD image data.

A BASIC subroutine to select byte number CZ% (0 to 50) in row RZ% (0 to 3) in

the previously selected driver is:

```
70 M3=CZ%+64*RZ%
80 OUT254,M3
90 RETURN
```

Before actually sending/retrieving a byte of bit-image data to/from the selected LCD screen location, one must first check that INP(254)<128. If not, then the LCD is still responding to the previous commands and one must wait before attempting to send or receive a byte of display data.

Once the driver DZ%, row RZ% and column CZ% have been selected, a byte containing the bit-image of data to be displayed at the selected location is sent via OUT255,byte. A BASIC subroutine for doing this is:

```
100 IF INP(254)>127 THEN 100
110 OUT 255,byte: REMARK byte is between 0 and 255
120 RETURN
```

If instead of displaying data one wishes to read the data that is currently on the LCD at the selected LCD screen location, then byte=INP(255) is substituted for OUT255,byte in line 110. In either case, the LCD automatically increments the selected byte column CZ% by one after each INP(255) or OUT255,byte instruction, so several bytes may be input or output in succession without repeating the driver, row, and column select procedures PROVIDED THAT ALL BYTES ARE TO BE DISPLAYED IN THE AREA UNDER THE CONTROL OF THE SAME DRIVER. It is an unfortunate fact that 24 of the 320 6-byte print positions on the LCD straddle two adjacent drivers.

For additional insight into the process of sending data to the LCD, see

the file CHRDEF.BA in DL4 (which contains a .BA program for displaying a six byte custom character on the LCD using CPU ports 185,186, 254, and 255).

Fortunately, there are machine code routines in the M100's ROM which can be used efficiently handle to task of sending blocks of six-bytes to and from the LCD and to handle the somewhat involute process of selecting right locations on the screen and switching drivers when the 6-bytes straddle two adjacent drivers. Consequently, in most applicatons there is no need for actually using any of the BASIC subroutines above (or for using the relatively involute CHRDEF.BA) to read/write custom characters or graphics to/from the LCD.

II. USEFUL LCD ROM CALLS FOR GRAPHICS READ AND WRITE

A crude but popular method used by various graphics screen dump and pixel tester programs is to read graphic data from the LCD is to flash the cursor on then off, which reads the six byte bit-image of that portion of the LCD under the cursor to RAM memory locations 65616 to 65621 (equivalently, locations -20 to -15) The data can then be PEEK'ed from these locations. For example, a BASIC program to read the six bytes a screen location 37 is:

```
10 PRINT@37,CHR$(27)"P" 'cursor on at 37
20 FOR I=1TO10:NEXT 'delay while LCD catches up
30 PRINTCHR$(27)"Q"; cursor off
40 FOR I=1TO10:NEXT 'delay while LCD catches up
50 FOR I=0TO5:BYTE(I)=PEEK(I-20):NEXT 'read six bytes LCD bit-image
data
60 RETURN
```

See, for example, PGM TIP.008 for a discussion of how to use this for testing whether or not a given pixel on the LCD is set. One criticism of this method is that it is relatively slow and can be unreliable. It seems that even BASIC is faster than the LCD and so that a program without appropriate additional delays inserted may occassionally attempt to PEEK the character data before the LCD has finished sending it. Another unpleasant feature of the approach to

reading graphics data from the LCD is the cursor flash which is visible and can be distracting or even annoying in some applications.

A closely related technique which is much faster and completely reliable involves the ROM call, CALL 17786, which INVERSES the six byte bit image at the current cursor position, storing a copy or the inverse bit image at RAM locations -20 to -15. The following BASIC program will read the six byte bit image of the LCD at location 49:

```
10 PRINT@49,; 'move cursor to location 49
20 CALL 17786 'inverse 6 bytes on screen & save
                'in RAM locations -20 to -15
30 CALL 17786 'do it again to re-invert it
50 FOR I=0TO5:BYTE(I)=PEEK(I-20):NEXT 'read six bytes LCD bit-image
data
60 RETURN
```

The machine code call 17786 has built-in a check on INP(254)<128 which seems to circumvent the need for adding delays, but it still suffers from the distracting cursor flash.

There is a simple to use ROM routine than this for sending data from the LCD. It is CALL 29744. The following BASIC program prints a six-byte custom character (an upside-down "T") at screen location 53 (which is more or less what CHRDEF.BA in DL4 does, but in a more involute fashion):

```
10 PRINT@53,; 'more cursor to desired display location
20 'store six byte bit image in integer array B%(I)
30 FOR I=0TO2:READ B1,B2:B%(I)=B1+256*B2:NEXT
40 CALL30300 'disable interrupts
50 CALL29744,0,VARPTR(B%(0)) 'display 6 byte bit-image of the vector
B%
60 RETURN
70 DATA 64,64,127,64,64,0
```

The machine code CALL 29744,0,HL transfers 6 bytes beginning at RAM memory location HL and displays them at the row(0 to 7) and column (0 to 39) position on the LCD specified by the current contents of RAM memory locations -12 and -11 respectively. The contents of these locations is automatically set to the current cursor position by any print command (in this case PRINT@53,;). However, the any row and column can be POKE'd to these locations directly if desired.

The first machine code instruction of in CALL 29744,0,HL is in 8085 assembly language MVI D,1 (which means load the D-register of the CPU with the number 1). The actual machine code consists of the two integers (22,1) which are stored in ROM at memory locations 29744 and 29755.

It turns out that the foregoing routine can be used to READ six-bytes FROM the LCD TO memory locations HL to HL+5 if the D-register is first loaded with 0 instead of 1, and then one uses CALL29746,0,HL bypassing the MVI D,1 instruction. This may be accomplished by the following position-independent machine code sequence 22,0,205,50,116,201,0. This can be poked anywhere in ROM and then called. For those familiar with 8085 assembly language, this code translates as:

```

    22 MVI D,0      'load 0 into D-register
      0
    205 CALL 29746  '29746=50+116*256
      50
     116
    201 RET        'return

```

The following BASIC subroutine loads this 6 byte machine code program into RAM at memory location VARPTR(MZ%(0)) so that it may be executed by the BASIC command CALL VARPTR(MZ%(0)),0,HL

```

10 MZ%(0)=22      ' =22+0*256
20 MZ%(1)=13005   ' =205+50*256
30 MZ%(2)=-13964 ' =(116+201*256) MOD 65536
40 RETURN

```

CALL VARPTR(MZ%(0)),0,HL pokes the 6 byte bit-image of LCD to the six RAM locations HL to HL+5.

This program can be easily modified to display the data at HL to HL+5 on the LCD (exactly like CALL 29744,0,HL) if one changes line 10 to read:

```

10 MZ%(0)=278     ' =22+1*256

```

As an example, consider the following BASIC program which reads and writes the entire LCD image including graphics to and from the 1921 byte machine code data file IMAGE.CO.

```

0 CLEAR256,MAXRAM-1921

```

```

10 ' ... (any program generating a plot, picture or
20 ' ... text on the LCD goes here)
30 INPUT$"[S]ave LCD image";QZ$
40 GOSUB 100
90 END
100 POKE HIMEM,201 'POKE a machine code "return" at beginning of file
101 HL=HIMEM+1
102 IFQZ$="S"THEN MZ%(0)=22 ELSE LOADM"IMAGE":MZ%(0)=278
103 MZ%(1)=13005
104 MZ%(2)=-13964
105 FOR RZ%=0TO6
106     FORCZ%=0TO39
107         CALL30300
108         POKE-12,RZ%
109         POKE-11,CZ%
110         CALLVARPTR(MZ%(0)),0,HL
111         HL=HL+6
112     NEXT
113 NEXT
114 IFQZ$="S" THEN SAVEM "IMAGE",HIMEM,HIMEM+1921
115 RETURN

```

If QZ\$="S" on entry then this program copies the LCD to the machine code file IMAGE.CO, otherwise it does the reverse, i.e., it displays the 1920 bytes of bit-image data in the file IMAGE.CO on the LCD. (Note: a similar routine is used for saving pictures from the LCD by CVSMOD.002 in DL3).

III. PRINTER INTERFACING: UPSIDE-DOWN BIT-IMAGE PROBLEMS

Many dot matrix printers accept bit-image graphics exactly as it comes from the LCD. An exception is certain popular EPSON printers, e.g., the EPSON RX-80 and the EPSON FX-80. These printers display graphics data "upside-down" relative to the M100's LCD bit-image because the EPSON's think bit 0 is at the bottom of each 8-bit-tall column of pixels whereas the M100 thinks it is at the top. Basic programs for turning a byte "upside-down" are much too slow for efficient dumping of LCD image data to such printers--unless you like watching radishes grow and other leisurely activities.

One solution is the following machine code BASIC subroutine to quickly turn a byte "upside down". It temporarily defines a machine code routine CALL 65024,A,HL which takes a one-byte number A (0 to 255) and stores its "upside-

down" bit-image at memory address HL.

```
400 RESTORE 403
401 FORI=0TO31:READZ:POKE65024+I,Z:NEXT
402 RETURN
403 DATA
0,87,62,0,6,1,14,128,54,0,183,216,120,162,120,23,71,121,31,79,202,11,254
404 DATA 245,23,182,119,241,195,11,254,0
```

After this subroutine has been run and until the routine is overwritten or destroyed by a PRINT command, CALL65024,A%,VARPTR(A%) takes an integer A% (0 to 255) as input and on exit A% is replaced its the "upside-down" inverse (i.e., bit 0 goes to bit 7, bit 1 to bit 6, etc.). The machine code subroutine resides at locations 65024-65055 and is NOT relocatable. This area of RAM is the first line of the LCD screen-image memory, not to be confused with the screen itself--it merely contain's a copy of the ASCII characters PRINT'ed on the LCD). The machine code program POKE'd here does not affect the LCD display in any way and remains until it is detroyed by either an ASCII character being PRINT'ed on the first line of the LCD or by the LCD being scrolled. (Note: This routine is used in the BASIC program EPSCHR.101 in DL0 which copies the M100's special character set to an EPSON FX80 printer).