HOW TO USE
INTELLIGENT L.C.D.S

Part One

By Julyan Ilett

This paper was originally published as the first half of a two-part article in the February 1997 issue of *Everyday Practical Electronics* magazine (www.epemag.wimborne.co.uk), and is reproduced here with their kind permission.

Recreated in Adobe Acrobat PDF format for your web-based reading pleasure by
Maxfield & Montrose Interactive Inc.

www.maxmon.com

How to use Intelligent L.C.D.s

By Julyan Ilett

An utterly “practical” guide to interfacing and programming intelligent liquid crystal display modules.

Part One

Recently, a number of projects using intelligent liquid crystal display (l.c.d.) modules have been featured in *EPE*. Their ability to display not just numbers, but also letters, words and all manner of symbols, makes them a good deal more versatile than the familiar 7-segment light emitting diode (l.e.d.) displays.

Although still quite expensive when purchased new, the large number of surplus modules finding their way into the hands of the “bargain” electronics suppliers, offers the hobbyist a low cost opportunity to carry out some fascinating experiments and realise some very sophisticated electronic display projects.

Basic Reading

This article deals with the character-based l.c.d. modules which use the Hitachi HD44780 (or compatible) controller chip, as do most modules available to the hobbyist. Of course, these modules are not quite as advanced as the latest generation, full size, full colour, back-lit types used in today's laptop computers, but far from being “phased out,” character-based l.c.d.s are still used extensively in commercial and industrial equipment, particularly where display requirements are reasonably simple.

The modules have a fairly basic interface, which mates well with traditional micro-processors such as the Z80 or the 6502. It is also ideally suited to the PIC microcontroller, which is probably the most popular microcontroller used by the electronics hobbyist.

However, even if, as yet, you know nothing of microcontrollers, and possess none of the PIC paraphernalia, don't despair, you can still enjoy all the fun of experimenting with l.c.d.s, using little more than a handful of switches!

Shapes and Sizes

Even limited to character-based modules, there is still a wide variety of shapes and sizes available. Line lengths of 8, 16, 20, 24, 32 and 40 characters are all standard, in one, two and four-line versions.

Several different liquid crystal technologies exist. “Supertwist” types, for example, offer improved contrast and viewing angle over the older “twisted nematic” types. Some modules are available with back-lighting, so that they can be viewed in dimly-lit conditions. The back-lighting may be either “electro-luminescent,” requiring a high voltage inverter circuit, or simpler l.e.d. illumination.

Few of these features are important, however, for experimentation purposes. All types are capable of displaying the same basic information, so the cheaper types are probably the best bet initially.

Connections

Most l.c.d. modules conform to a standard interface specification. A 14-pin access is provided (14 holes for solder pin insertion or for an IDC connector) having eight data lines, three control lines and three power lines. The connections are laid out in one of two common configurations, either two rows of seven pins, or a single row of 14 pins. The two layout alternatives are displayed in Figure 1.

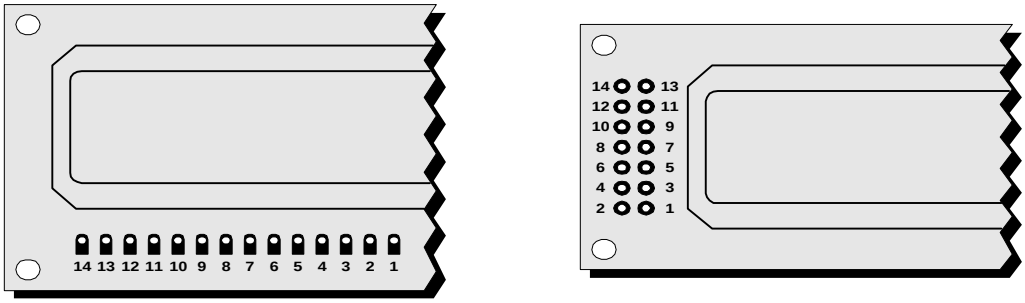


Figure 1: Pinouts of the two basic l.c.d formats.

On most displays, the pins are numbered on the l.c.d.'s printed circuit board, but if not, it is quite easy to locate pin 1. Since this pin is connected to ground, it often has a thicker p.c.b. track connected to it, and it is generally connected to the metalwork at some point.

The function of each of the connections is shown in Table 1. Pins 1 and 2 are the power supply lines, Vss and Vdd. The Vdd pin should be connected to the positive supply, and Vss to the 0V supply or ground.

Pin No.	Name	Function
1	Vss	Ground
2	Vdd	+ve supply
3	Vee	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data bit 0
8	D1	Data bit 1
9	D2	Data bit 2
10	D3	Data bit 3
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7

Table 1. Pinout functions for all the l.c.d. types.

Although the l.c.d. module data sheets specify a 5V d.c. supply (at only a few milliamps), supplies of 6V and 4.5V both work well, and even 3V is sufficient for some modules. Consequently, these modules can be effectively, and economically, powered by batteries.

Pin 3 is a control pin, Vee, which is used to alter the contrast of the display. Ideally, this pin should be connected to a variable voltage supply. A preset potentiometer connected between the power supply lines, with its wiper connected to the contrast pin is suitable in many cases, but be aware that some modules may require a negative potential; as low as 7V in some cases. For absolute simplicity, connecting this pin to 0V will often suffice.

Pin 4 is the Register Select (RS) line, the first of the three command control inputs. When this line is low, data bytes transferred to the display are treated as commands, and data bytes read from the display indicate its status. By setting the RS line high, character data can be transferred to and from the module.

Pin 5 is the Read/Write (R/W) line. This line is pulled low in order to write commands or character data to the module, or pulled high to read character data or status information from its registers.

Pin 6 is the Enable (E) line. This input is used to initiate the actual transfer of commands or character data between the module and the data lines. When writing to the display, data is transferred only on the high to low transition of this signal. However, when reading from the display, data will become available shortly after the low to high transition and remain available until the signal falls low again.

Pins 7 to 14 are the eight data bus lines (D0 to D7). Data can be transferred to and from the display, either as a single 8-bit byte or as two 4-bit “nibbles.” In the latter case, only the upper four data lines (D4 to D7) are used. This 4-bit mode is beneficial when using a microcontroller, as fewer input/output lines are required.

Prototype Circuit

For an l.c.d. module to be used effectively in any piece of equipment, a microprocessor or microcontroller is usually required to drive it. However, before attempting to wire the two together, some initial (and very useful) experiments can be performed, by connecting up a series of switches to the pins of the module. This can be quite a beneficial step, even if you are thoroughly conversant with the workings of microprocessors.

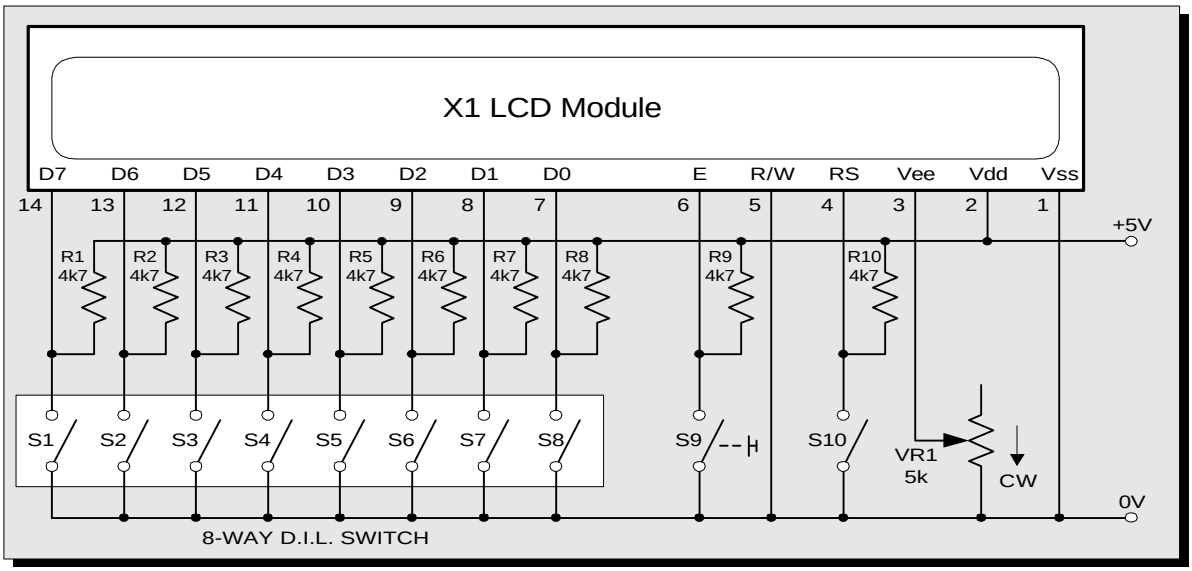


Figure 2: Circuit diagram for an l.c.d. experimental rig.

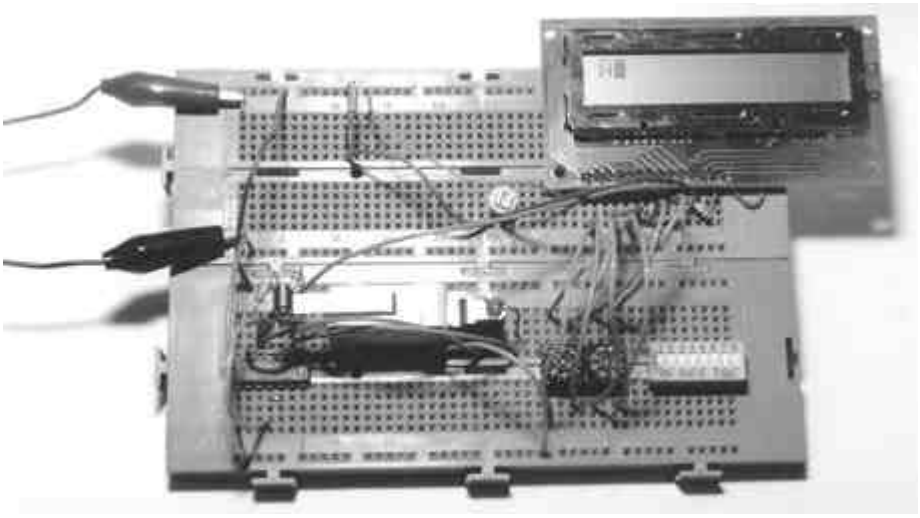
In Figure 2 is shown the circuit diagram of an l.c.d. experimentation rig. The circuit can be wired-up on a “plug-in” style prototyping board, using d.i.l. (dual-in-line) switches for the data lines (S1 to S8), a toggle switch for the RS input (S10), and a momentary action switch (or microswitch) for the E input (S9). The R/W line is connected to ground (0V), as the display is only going to be written to for the time being.

All of the resistors (R1 through R10) are 4K7 ohms. It is probably most convenient to use a s.i.l. (single-in-line) resistor pack for the eight pull-up resistors (R1 to R8) on the data lines. The other two resistors, R9 and R10, can be discrete types. Preset potentiometer VR1 (5K ohms) is used for the contrast control and is shown with one end left disconnected. If desired, this end can be connected to the positive line via a resistor of about 47K ohms (it should be connected to a negative supply, via a similar resistor, for those modules which require negative biasing).

All the switches should be connected so that they are “on” when in the “down” position, so that “down” generates a logic 0 (low) and “up” provides a logic 1 (high). The switches should also be arranged so that data bit D7 is on the left, and data bit D0 is on the right. In this way, binary numbers can be entered the right way round.

Initially, the contrast control should be adjusted fully clockwise, so that the contrast control input (Vee) is connected to ground. The initial settings of the switches are

unimportant, but it is suggested that the RS switch (S10) is “up” (set to logic 1), and the E switch (S9) is left unpressed. The data switches, S1 to S8, can be set to any value at this stage. All is now prepared to start sending commands and data to the l.c.d. module.



The experimental circuit can be built on plug-in prototyping boards.

Experiment 1: Basic Commands

When powered up, the display should show a series of dark squares, possibly only on part of the display. These character cells are actually in their off state, so the contrast control should be adjusted anti-clockwise (away from ground) until the squares are only just visible.

The display module resets itself to an initial state when power is applied, which curiously has the display blanked off, so that even if characters are entered, they cannot be seen. It is therefore necessary to issue a command at this point, to switch the display on.

A full list of the commands that can be sent is given in Table 2, together with their binary and hexadecimal values. The initial conditions of the l.c.d. after power-on are marked with an asterisk.

Throughout this article, emphasis will be placed on the binary value being sent since this illustrates which data bits are being set for each command. After each binary value, the equivalent hexadecimal value is quoted in brackets, the $ prefix indicating that it is hexadecimal.

The Display On/Off and Cursor command turns on the display, but also determines the cursor style at the same time. Initially, it is probably best to select a Blinking Cursor with Underline, so that its position can be seen clearly, i.e. code 00001111 ($0F).

Command	Binary D7	D6	D5	D4	D3	D2	D1	D0	Hex
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1 / D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D / C	R / L	x	x	10 to 1F
Function Set	0	0	1	8 / 4	2 / 1	10 / 7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

1 / D:	1=Increment*, 0=Decrement	R / L:	1=Right shift, 0=Left shift
S:	1=Display shift on, 0=Off*	8 / 4:	1=8-bit interface*, 0=4-bit interface
D:	1=Display on, 0=Off*	2 / 1:	1=2 line mode, 0=1 line mode*
U:	1=Cursor underline on, 0=Off*	10 / 7:	1=5x10 dot format, 0=5x7 dot format*
B:	1=Cursor blink on, 0=Off*		
D / C:	1=Display shift, 0=Cursor move	x = Don't care	* = Initialization settings

Table 2. The command control codes.

Set the data switches (S1 to S8) to 00001111 ($0F) and ensure that the RS switch (S10) is “down” (logic 0), so that the device is in Command mode. Now press the E switch (S9) momentarily, which “enables” the chip to accept the data, and Hey Presto, a flashing cursor with underline appears in the top left hand position!

If a two-line module is being used, the second line can be switched on by issuing the Function Set command. This command also determines whether an 8-bit or a 4-bit data transfer mode is selected, and whether a 5 x 10 or 5 x 7 pixel format will be used. So, for 8-bit data, two lines and a 5 x 7 format, set the data switches to binary value 00111000 ($38), leave RS (S10) set low and press the E switch, S9.

It will now be necessary to increase the contrast a little, as the two-line mode has a different drive requirement. Now set the RS switch to its “up” position (logic 1), switching the chip from Command mode to Character mode, and enter binary value 01000001 ($41) on the data switches. This is the ASCII code for a capital *A*.

Press the E switch, and marvel as the display fills up with capital *A*'s. Clearly, something is not quite right, and seeing your name in pixels is going to have to wait a while.

Bounce

The problem here is contact bounce. Practically every time the E switch is closed, its contacts will bounce, so that although occasionally only one character appears, most attempts will result in 10 or 20 characters coming up on the display. What is needed is a “debounce” circuit.

But what about the commands entered earlier, why didn't contact bounce interfere with them? In fact it did, but it doesn't matter whether a command is entered (“enabled”) just once, or several times, it gets executed anyway. A solution to the bounce problem is shown in Figure 3.

Here, a couple of NAND gates are cross-coupled to form a set-reset latch (or flip-flop) which flips over and latches, so that the contact bounce is eliminated. Either a TTL 74LS00 or a CMOS 74HC00 can be used in this circuit. The switch must be an s.p.d.t. (single-pole, double-throw) type, a microswitch is ideal.

After modifying the circuit, the screen full of *A*'s can be cleared using the Clear Display command. Put binary value 00000001 ($01) on the data switches, set the RS switch to the "down" position and press the new modified E switch. The display is cleared.

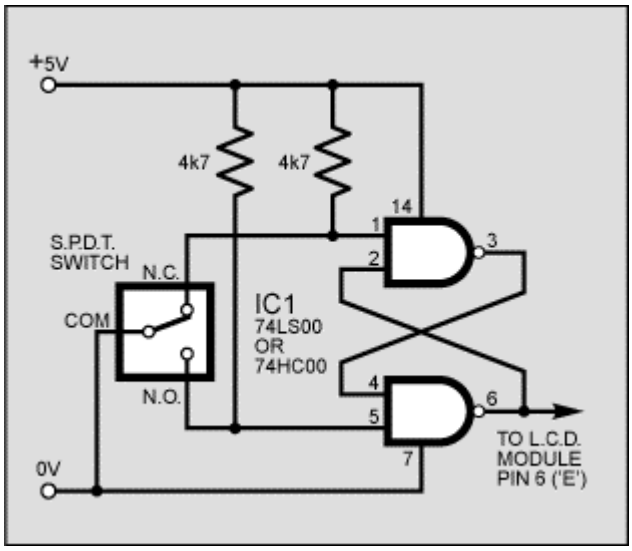


Figure 3. Switch debounce circuit.

Note that the output of the "de-bounce" circuit is high when the switch is pressed and low when the switch is released. Since it is the high to low transition that actually latches data into the l.c.d. module, it will be observed that characters appear on the display, not when the button is pressed, but when it is released.

Experiment 2: Entering Text

First, a little tip: it is manually a lot easier to enter characters and commands in hexadecimal rather than binary (although, of course, you will need to translate commands from binary into hex so that you know which bits you are setting). Replacing the d.i.l. switch pack with a couple of sub-miniature hexadecimal rotary switches is a simple matter, although a little bit of re-wiring is necessary.

The switches must be the type where On = 0, so that when they are turned to the zero position, all four outputs are shorted to the common pin, and in position "F", all four outputs are open circuit.

All the available characters that are built into the module are shown in Table 3. Studying the table, you will see that codes associated with the characters are quoted in binary and hexadecimal, most significant bits ("left-hand" four bits) across the top, and least significant bits ("right-hand" four bits) down the left.

Most of the characters conform to the ASCII standard, although the Japanese and Greek characters (and a few other things) are obvious exceptions. Since these intelligent modules were designed in the "Land of the Rising Sun," it seems only fair that their Katakana phonetic symbols should also be incorporated. The more extensive Kanji character set, which the Japanese share with the Chinese, consisting of several thousand different characters, is not included!

Table 3. Standard l.c.d character table.

Using the switches, of whatever type, and referring to Table 3, enter a few characters onto the display, both letters and numbers. The RS switch (S10) must be “up” (logic 1) when sending the characters, and switch E (S9) must be pressed for each of them. Thus

the operational order is: set RS high, enter character, trigger E, leave RS high, enter another character, trigger E, and so on.

The first 16 codes in Table 3, 00000000 to 00001111, ($00 to $0F) refer to the CGRAM. This is the Character Generator RAM (random access memory), which can be used to hold user-defined graphics characters. This is where these modules really start to show their potential, offering such capabilities as bargraphs, flashing symbols, even animated characters. Before the user-defined characters are set up, these codes will just bring up strange looking symbols.

Codes 00010000 to 00011111 ($10 to $1F) are not used and just display blank characters. ASCII codes "proper" start at 00100000 ($20) and end with 01111111 ($7F). Codes 10000000 to 10011111 ($80 to $9F) are not used, and 10100000 to 11011111 ($A0 to $DF) are the Japanese characters.

Codes 11100000 to 11111111 ($E0 to $FF) are interesting. Although this last block contains mainly Greek characters, it also includes the lower-case characters which have "descenders." These are the letters *g*, *j*, *p*, *q* and *y*, where the tail drops down below the base line of normal upper-case characters. They require the 5 x 10 dot matrix format, rather than the 5 x 7, as you will see if you try to display a lower-case *j*, for example, on a 5 x 7 module.

Some one-line displays have the 5 x 10 format facility, which allows these characters to be shown unbroken. With 5 x 7 two-line displays, the facility can be simulated by borrowing the top three pixel rows from the second line, so creating a 5 x 10 matrix.

For this simulation, set line RS low to put the chip into Command mode. On the data switches, enter the Function Set command using binary value 00110100 ($34). Press and release switch E. Return RS to high, and then send the character data for the last 32 codes in the normal way (remembering to trigger line E!).

Experiment 3: Addressing

When the module is powered up, the cursor is positioned at the beginning of the first line. This is address $00. Each time a character is entered, the cursor moves on to the next address, $01, $02 and so on. This auto-incrementing of the cursor address makes entering strings of characters very easy, as it is not necessary to specify a separate address for each character.

It may be necessary, however, to position a string of characters somewhere other than at the beginning of the first line. In this instance, a new starting address must be entered as a command. Any address between $00 and $7F can be entered, giving a total of 128 different addresses, although not all these addresses have their own display location. There are in fact only 80 display locations, laid out as 40 on each line in two-line mode, or all 80 on a single line in one-line mode. This situation is further complicated because not all display locations are necessarily visible at one time. Only a 40-character, two-line module can display all 80 locations simultaneously.

To experiment with addressing, first set the l.c.d. to two-line mode (if two lines are available), 8-bit data and 5 [P3] 7 format using the Function Set command, i.e. code 00111000 ($38). Note that the last two bits of this command are unimportant, as indicated by the X in the columns of Table 2, and either of them may be set to 0 or 1.

(From now on, we won't constantly remind you that RS must be set appropriately before Command or Character data is entered, or that E must be triggered after data has been entered ¾ you should know by now!)

Using the Display On/Off and Cursor command, set the display to On, with Underline and Blinking Cursor, code 00001111 ($0F). Now set the cursor to address 00001000 ($08). This is done by sending a Set Display Address command, binary value 10001000 ($88).

The cursor will jump to the ninth position on the display, at which point text can now be entered. The Set Display Address command is always 10000000 ($80) greater than the display address itself.

Experiment with different display addresses and note their display locations. Be aware that display addresses 00101000 to 00111111 ($28 to $3F) and 01101000 to 01111111 ($68 to $7F) cannot be used on any of the display types.

The relationship between addresses and display locations varies, depending on the type of module being used, but some typical examples are shown in Figure 4.

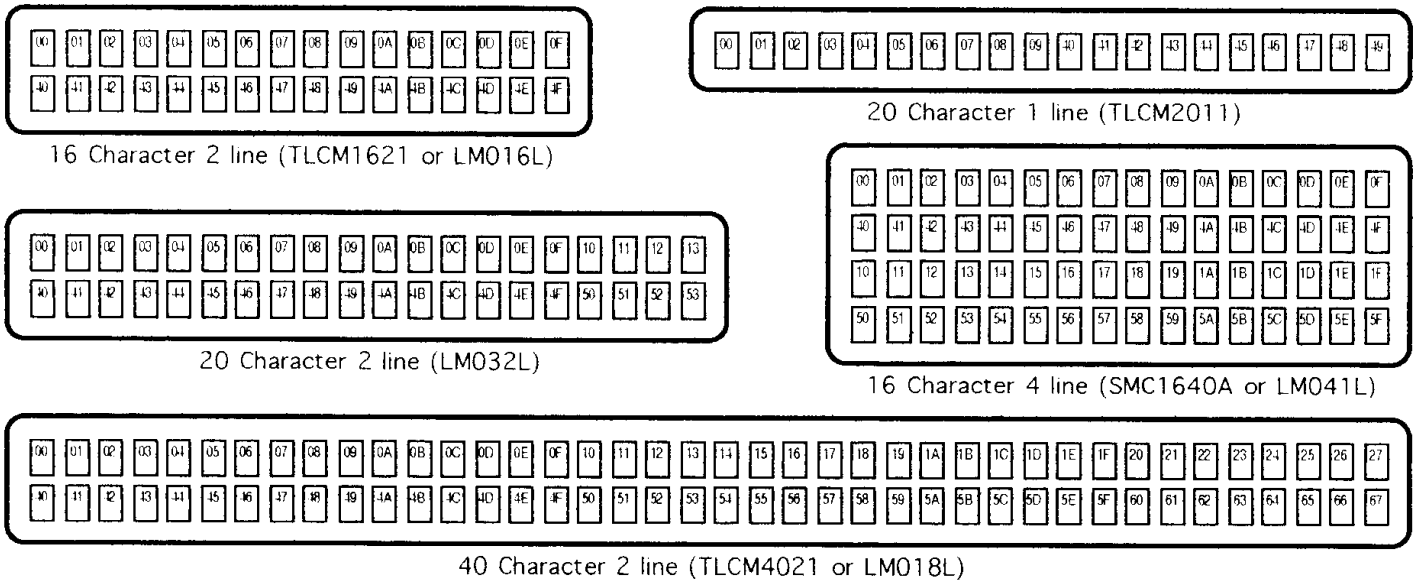


Figure 4: Examples of the relationship between addresses and display locations for typical module formats

Most are laid out conventionally, with two lines of characters, the first line starting at address 00000000 ($00) and the second line at address 01000000 ($40).
Two interesting exceptions were discovered during this article's research. The single-line module shown in Figure 4 is actually a two-line type, with the second line placed to the right of the first. In one-line mode, only the first 10 characters were visible.

The rather magnificent 4-line module is, actually, also a two-line type, with the two lines split and interlaced. This complicates the addressing a little, but can be sorted out with a bit of software.

Experiment 4: Shifting the Display

Regardless of which size l.c.d. module is being used, there are always 80 display locations that can be written to. On the smaller devices, not all 80 fit within the visible window of the module, but can be brought into view by shifting them all, either left or right, “beneath” the window area. This process must be carried out carefully, however, as it alters the relationship between addresses and their positions on the screen.

To experiment with shifting, first issue suitable Function Set, Display On/Off and Cursor commands, and, if necessary, the Clear Display command (you've met their codes above). Then enter all 26 letters of the alphabet as character data, e.g. 01000001 ($41) to 01011010 ($5A).

On a 16-character display, only *A* to *P* will be visible (the first 16 letters of the alphabet), and the cursor will have disappeared off the right-hand side of the display screen.

The Cursor/Display Shift command can now be used to scroll all the display locations to the left, “beneath” the l.c.d. window, so that letters *Q* to *Z* can be seen. The command is binary 00011000 ($18). Each time the command is entered (and using the E switch), the characters shift one place to the left. The cursor will re-appear from the right-hand side, immediately after the *Z* character.

Carry on shifting (*wasn't that a film title? Ed!*), and eventually the letters *A*, *B*, *C*, and so on, will also come back in from the right-hand side. Shifting eventually causes complete rotation of the display locations.

The binary command 00011100 ($1C) shifts the character locations to the right. It is important to note that this scrolling does not actually move characters into new addresses, it moves the whole address block left or right “underneath” the display window.

If the display locations are not shifted back to their original positions, then address $00 will no longer be at the left-hand side of the display. Try entering an Address Set command of value 10000000 ($80), after a bit of shifting, to see where it has moved to.

The Cursor Home command, binary 00000010 ($02), will both set the cursor back to address $00, and shift the address $00 itself back to the left-hand side of the display. This command can be used to get back to a known good starting position, if shifting and address setting gets a bit out of control.

The Clear Display command does the same as Cursor Home, but also clears all the display locations.

One final word about the Cursor/Display Shift command; it is also used to shift the cursor. Doing this simply increments or decrements the cursor address and actually has very little in common with shifting the display, even though both are achieved using the same command.

Experiment 5: Character Entry Mode

Another command listed in Table 2 is Character Entry Mode. So far, characters have been entered using auto-incrementing of the cursor address, but it is also possible to use auto-decrementing. Furthermore, it is possible to combine shifting of the display with both auto-incrementing and auto-decrementing.

Consider an electronic calculator. Initially, a single zero is located on the right-hand side of the display. As numbers are entered, they move to the left, leaving the cursor in a fixed position at the far right. This mode of character entry can be emulated on the l.c.d. module. Time for another experiment:

Send suitable Function Set, Display On/Off and Cursor commands as before. Next, and assuming a 16-character display, set the cursor address to 00010000 ($10). Then send the Character Entry Mode command, binary 00000111 ($07). This sets the entry mode to auto-increment/display shift left.

Finally, enter a few numbers from 0 to 9 decimal, i.e. from 00110000 to 00111001 ($30 to $39). Characters appear on the right-hand side and scroll left as more characters are entered, just like a normal calculator.

As seen in Table 2, there are four different Character Entry modes, 00000100 to 00000111 ($04 to $07), all of which have their different uses in real life situations.

Experiment 6: User-Defined Graphics

Commands 01000000 to 01111111 ($40 to $7F) are used to program the user-defined graphics. The best way to experiment with these is to program them “on screen.” This is carried out as follows:

First, send suitable Function Set, Display On/Off and Cursor commands, then issue a Clear Display command. Next, send a Set Display Address command to position the cursor at address 00000000 ($00). Lastly, display the contents of the eight user character

locations by entering binary data 00000000 to 00000111 ($00 to $07) in turn. These characters will initially show up as garbage, or a series of stripes.

Now, send a Set CGRAM Address command, to start defining the user characters. Any value between 01000000 and 01111111 ($40 and $7F) is valid, but for now, use 01000000 ($40). The cursor will jump to the beginning of the second line, but ignore this, as it is not important.

Data entered from now on will build up the user-defined graphics, row by row. Try the following sequence of data: 00001110, 00010001, 00001110, 00000100, 00011111, 00000100, 00001010, 00010001 ($0E, $11, $0E, $04, $1F, $04, $0A, $11). A little "stick man" will appear on the display, with his feet in the gutter (the cursor line)!

By entering another set of eight bytes, the second user character can be defined, and so on.

How the CGRAM addresses correspond to the individual pixels of the user-defined graphics characters is illustrated in Figure 5. Up to eight graphics can be programmed, which then become part of the character set and can be called up using codes 00000000 to 00000111 ($00 to $07), or codes 00001000 to 00001111 ($08 to $0F), both of which produce the same result, i.e. 64 command codes available for user programming.

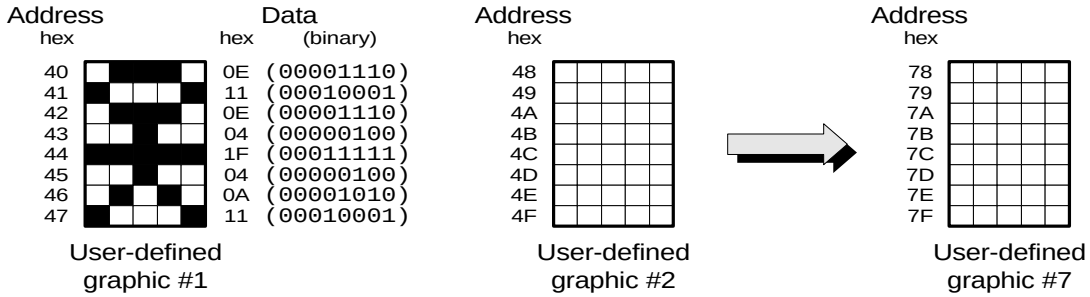


Figure 5: Showing how the CGRAM addresses correspond to individual pixels.

It can be seen that the basic character cell is actually eight pixels high by five pixels wide, but most characters just use the upper seven rows. The bottom row is generally used for the underline cursor. Since each character is only five pixels wide, only data bits 0 to 4 are used, bits 5 to 7 (the three "left-hand" bits) are ignored.

The CGRAM is volatile memory, which means that when the power supply is removed from the l.c.d. module, the user-defined characters will be lost. It is necessary for the microprocessor to load up the user-defined characters, by copying data from its own EPROM, early on in the program, certainly before it intends to display them.

Experiment 7: 4-Bit Data Transfer

The HD44780 l.c.d. control chip, found in most l.c.d. modules, was designed to be compatible with 4-bit microprocessors. The 4-bit mode is still very useful when interfacing to microcontrollers, including the PIC types.

Microcontroller input/output (I/O) pins are often at a premium and have to be rationed carefully between the various switches, displays and other input and output devices in a typical circuit. Bigger microcontrollers are available, which have more I/O pins, but miniaturisation is a key factor these days, along with cost, of course.

Once the display is put into 4-bit mode, using the Function Set command, it is a simple matter of sending two “nibbles” instead of one byte, for each subsequent command or character.

Nibble is a name devised by early computer enthusiasts in America, for half a byte, and is one of the more frivolous terms that has survived. By the time the 16-bit processors arrived, computing was getting serious, and the consumption analogies “gobble” and “munch” were never adopted!

When using 4-bit mode, only data lines D4 to D7 are used. On the prototype test rig, set the switches on the other lines, D0 to D3, to logic 0, and leave them there. Another experiment is now imminent.

In normal use, the unused data I/O lines D0 to D3 should either be left floating, or tied to one of the two power rails via a resistor of somewhere between 4k7[C24] and 47k[C24]. It is undesirable to tie them directly to ground unless the R/W line is also tied to ground, preventing them from being set into output mode. Otherwise the device could be programmed erroneously for 8-bit output, which could be unkind to lines D0 to D3, even though current limiting exists.

After power on, the l.c.d. module will be in 8-bit mode. The Function Set command must first be sent to put the display into 4-bit mode, but there is a difficulty. With no access to the lower four data lines, D0 to D3, only half the command can be applied.

Fortunately, or rather, by clever design, the 8-bit/4-bit selection is on data bit D4, which, even on the modified test rig, remains accessible. By sending a command with binary value 00100000 ($20), the 4-bit mode is invoked.

Now, another Function Set command can be sent, to set the display to two-line mode. Binary value 00101000 ($28) will do the trick. The value 00111000 ($38) may be a more familiar number, but it cannot be used now, or the display would be put straight back into 8-bit mode! Also, from now on, all commands and data must be sent in two halves, the upper four bits first, then the lower four bits.

Start by setting data lines D7, D6, D5 and D4 to 0010 ($2), the left-hand four bits of the 8-bit code, and press the E switch. Then we set the same four data lines to 1000 ($8), the right-hand four bits of the 8-bit code, and press the E switch again. It's a lot more laborious for a human being, but to a microcontroller, it's no problem!

Finish off by experimenting with other commands in 4-bit mode, and then try putting a few characters on the display.

A Final Note

The data sheets warn that under certain conditions, the l.c.d. module may fail to initialise properly when power is first applied. This is particularly likely if the Vdd supply does not rise to its correct operating voltage quickly enough.

It is recommended that after power is applied, a command sequence of three bytes of value 0011XXXX ($3X) is sent to the module. The value $30 is probably most convenient. This will guarantee that the module is in 8-bit mode, and properly initialised. Following this, switching to 4-bit mode (and indeed all other commands) will work reliably.

That’s it – For Now!

Well, that's about it, really. You've made it this far, so now you know everything there is to know about l.c.d. modules. Well, almost everything!

The next step, of course, is to connect the display up to a controller of some sort, such as a PIC microcontroller, as will be seen next month. Then we shall also consider such things as signal timing and instruction delays.

Acknowledgement

The author expresses his gratitude to Bull Electrical in Hove and Greenweld Electronics in Southampton for their help in connection with this article.