THE HD44780 LCD CONTROLLER APPLICATION NOTES

Name: Duane Bala
Date: 19th July 2001

INTRODUCTION

Many LCDs use controllers that either conform to or are compatible with the Hitachi HD44780 de-facto standard.

If you look at the back of an LCD which conforms to this standard, you will see that one of the chips would have HD44780 written across it. There are many (small) variations of the standard so you may see, for example, HD44780A00. If one of the chips does not have HD44780 written across it, hope is not lost as many manufacturers make controller chips that are HD44780 compatible. Such chips may not have HD44780 written across them.

LCDs may vary in the number of characters per line and in the number of lines even though they conform to the HD44780 standard. The most common LCDs have either 16 or 20 characters per line, while having 1, 2, or 4 lines in all.

THE HD44780 STANDARD

The HD44780 controller has 3 input control lines and either 4 or 8 data lines. The data lines may be either input or output lines. Together, the control and data lines form the instructions that tell the LCD module what to do.

The Instruction Register and the Data Register

The LCD module has two 8-bit registers, the Instruction Register (IR) and the Data Register (DR). The IR is used to store instructions while the DR temporarily stores data read from or to be written to the DDRAM. The DDRAM is the memory that stores all of the characters being displayed by the LCD. Therefore, to write characters on the display, we write to the DR. On the other hand, when we are specifying the parameters of the LCD module (such as 4-bit interface, 1-line display, display on, etc.) or instructing the LCD controller to perform some function (such as clear display, cursor home, etc.) we write to the IR.

The following diagram shows the logical configuration of the LCD module.

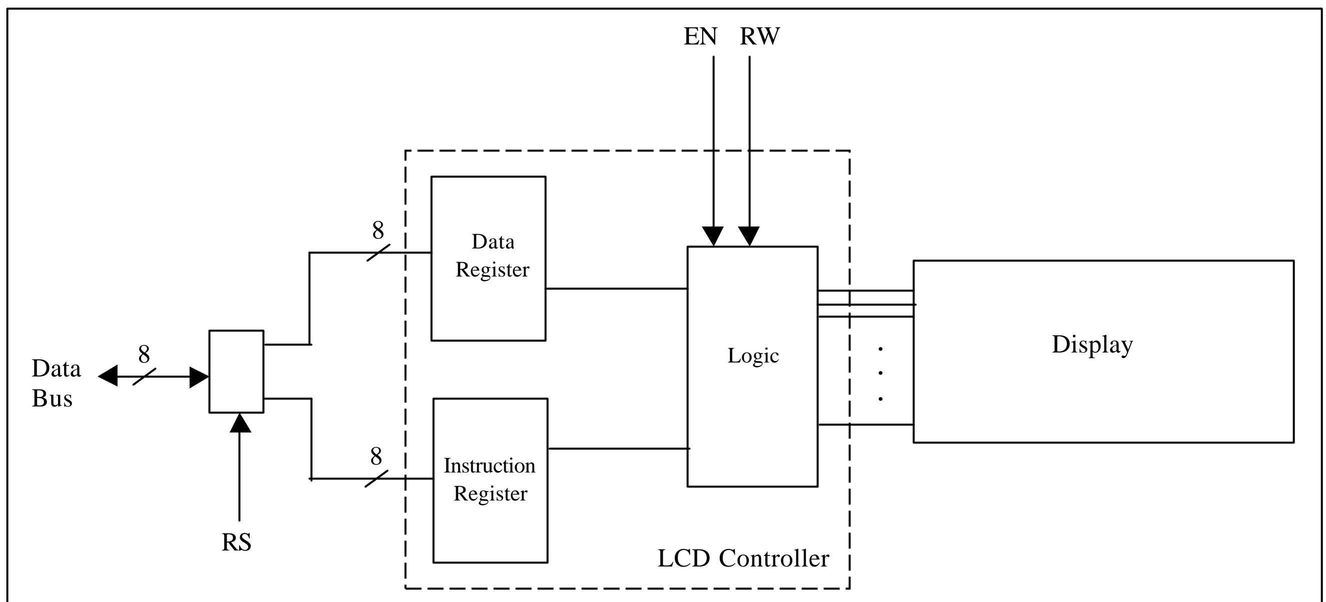

Figure 1 The Logical Configuration of the LCD Controller

The Control Lines

The 3 control lines are the Register Select Line (RS), the Read/Write Line (RW) and the Enable Line (EN).

The RS Line

This line is used to select either the Instruction Register (IR) or the Data Register (DR). RS is set (RS=1) to select the Data Register, while cleared (RS=0) to select the Instruction Register.

The RW Line

It indicates to the LCD module whether you are writing to or reading from, the register selected by RS (which would be either the IR or the DR). RW is set (RW=1) to indicate that you are reading the register, while cleared (RW=0) to indicate that you are writing to the register.

The EN Line

The EN line is used like a clock input to the controller. Raising EN and then lowering it signals the controller to perform the operation contained in the IR.

The Data Lines

The data lines are referred to as DB0, DB1, DB2…DB7. These lines make up the Data Bus with DB0 being the Least Significant Bit and DB7 being the Most Significant Bit. As mentioned before, the user can choose between a 4-bit interface and an 8-bit interface. In the case of the 8-bit interface, DB0…DB7 are used, while only DB4…DB7 are used with the 4-bit interface (DB4 being the Least Significant Bit).

The Read and Write Operations

The following tables show the basic read and write operations. When an 8-bit interface is used the operations are as in these tables. That is, only one data transfer is required (for either operation). With a 4-bit interface two data transfers are required (for either operation). Figures 3 and 5 show the write and read operations when a 4-bit interface is used.

The Write Operation: 8-bit interface	
Assume	EN is initially low (EN=0)
1	Set RS and RW to their desired states (i.e. either 0 or 1)
2	Wait a minimum of 60ns
3	Raise EN (EN=1)
4	Set the Data Bus to the desired value
5	Wait a minimum of 195ns
6	Clear EN (EN=0)
Notes	Keep RS and RW at their current states for a minimum of 20ns Keep the current value at the Data Bus for a minimum of 10ns

Table 1 The Write Operation (8-Bit Interface)

The Read Operation: 8-bit interface	
Assume	EN is initially low (EN=0)
1	Set RS and RW to their desired states (i.e. either 0 or 1)
2	Wait a minimum of 60ns
3	Raise EN (EN=1)
4	Wait a minimum of 360ns
5	Read the value across the Data Bus
6	Clear EN (EN=0)
Notes	Keep RS and RW at their current states for a minimum of 20ns The value across the Data Bus will be held (by the LCD module) for a minimum of 5ns

Table 2 The Read Operation (8-Bit Interface)

Basic Rules:

There are some basic rules that must be followed regardless of the operation (i.e. for both read and write operations).

Basic Timing Requirements
EN must be kept high a minimum of 450ns
The time between 0-1 transitions of EN must be a minimum of 1000ns
The length of the 0-1 transition of EN must be a maximum of 20ns
The length of the 1-0 transition of EN must be a maximum of 20ns. The transition occurs when EN is cleared after being high. For example, step 6 in both the read and write operations.

Table 3 The Basic Timing Requirements for Read and Write Operations

It is important to note that the LCD can operate at both 2.7 to 4.5 Volts and at 4.5 to 5.5 Volts. However, the timing requirements for both ranges are different. Therefore, to be compatible with both ranges the smaller of the maximum timing requirements (between the two ranges) were taken. Also, the larger of the minimum timing requirements (between the two ranges) were taken. The given delays meet the requirements of both ranges.

You may want to check the manual for the actual timing requirements if the given delays (in particular the maximum delays) cannot be met.

The following diagram illustrates the Write Operation

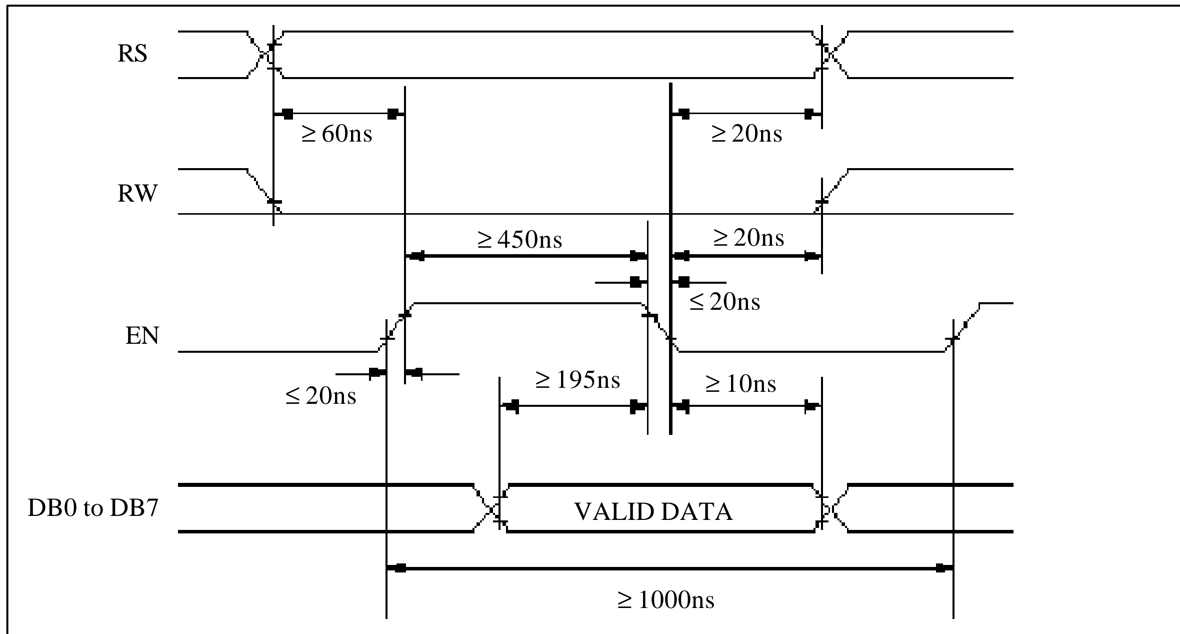

Figure 2 Write Operation 8-bit Interface

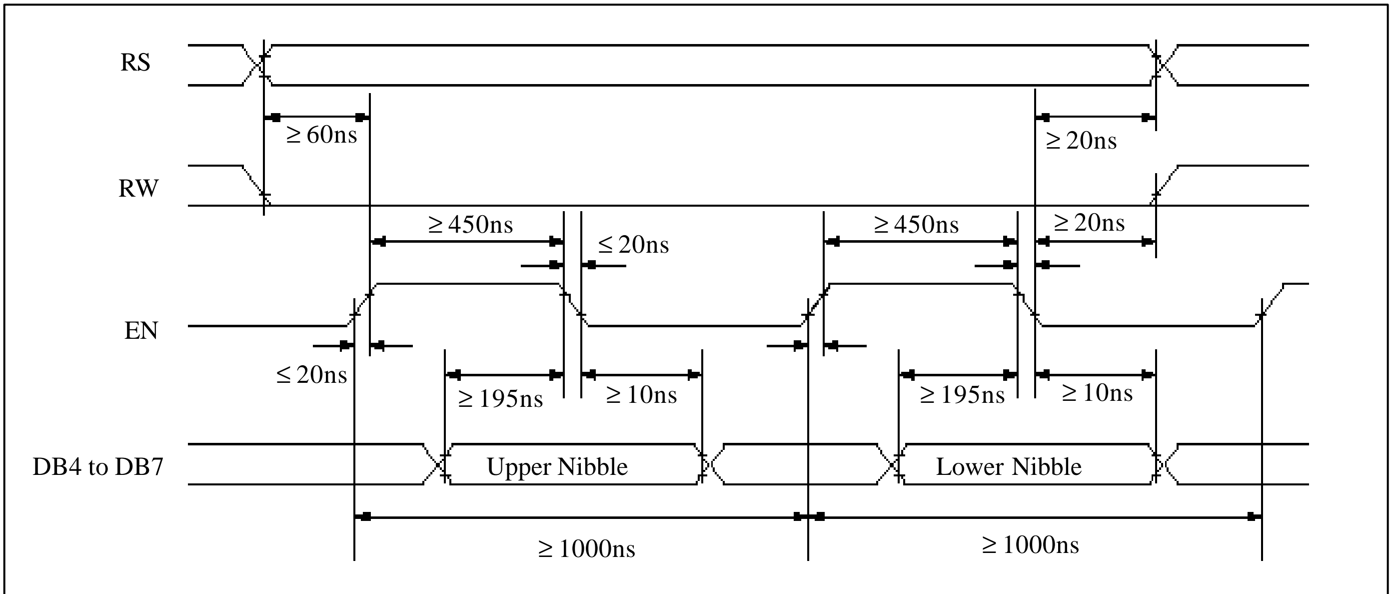

Figure 3 Write Operation 4-bit Interface

The following diagram illustrates the Read Operation

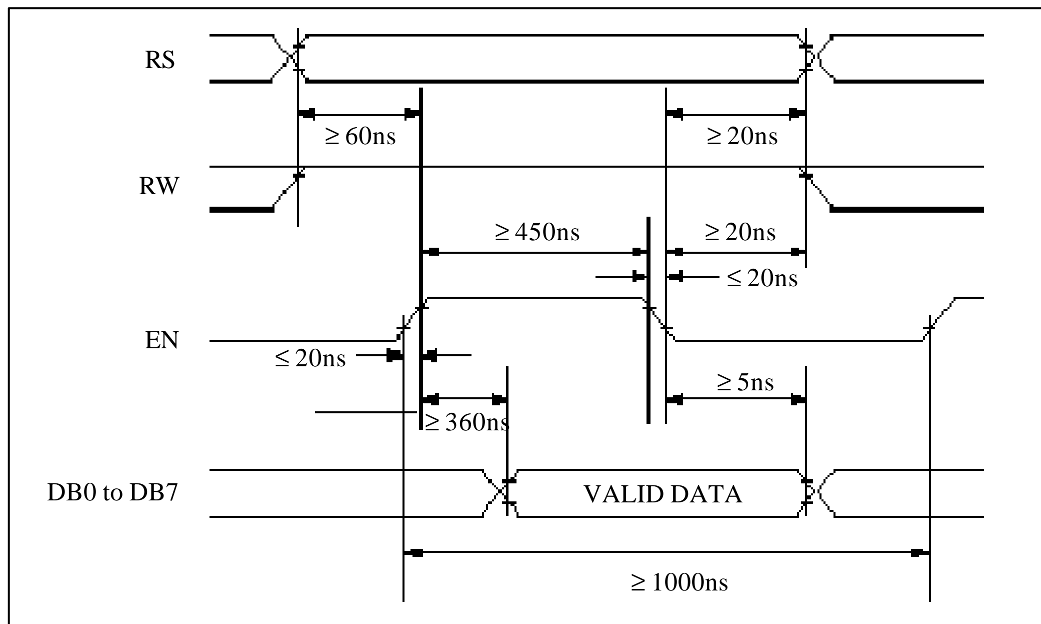

Figure 4 Read Operation 8-bit Interface

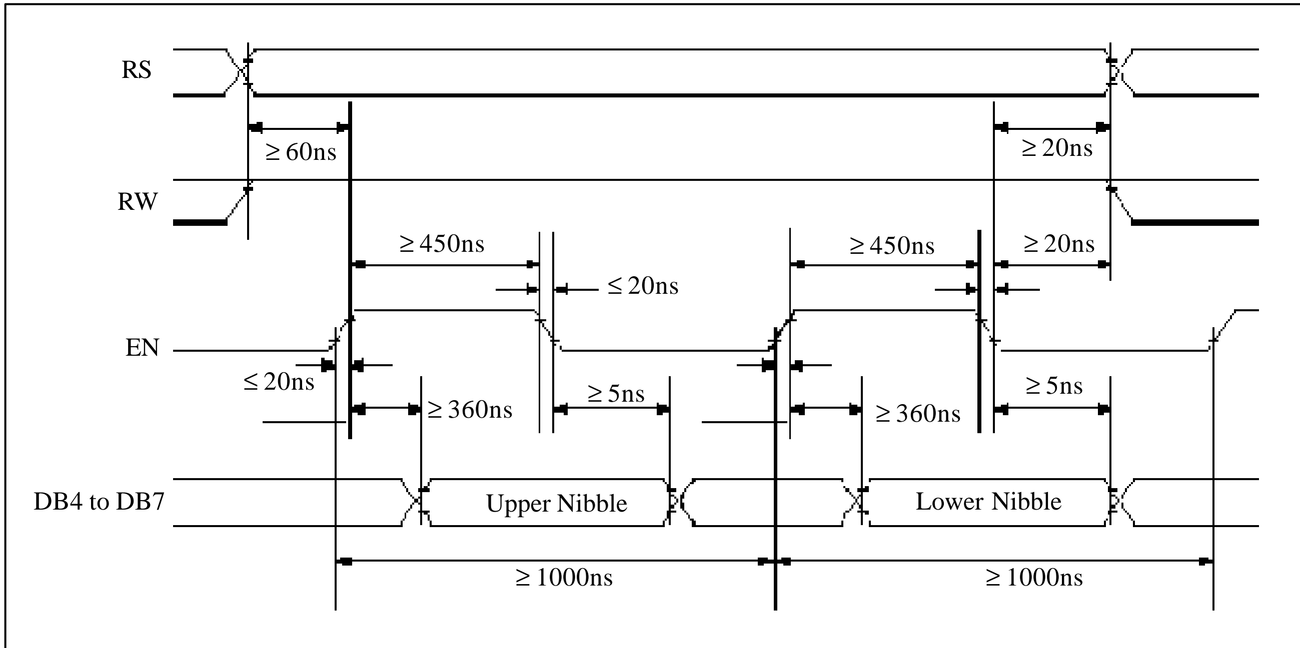

Figure 5 Read Operation 4-bit Interface

The Typical Sequence of Operation

The typical sequence of operation for the HD44780 controller is shown in the following diagram.

Instruction	Operation
Function Set	Initialisation
Check Busy Flag	
Display On/Off Control	
Check Busy Flag	
Clear Display	
Check Busy Flag	
Entry Mode Set	
Check Busy Flag	Setup
Set Cursor Position	
Check Busy Flag	
Cursor/display Shift	
Check Busy Flag	Display Text
Write to DDRAM	
…	
Check Busy Flag	
Write to DDRAM	

Figure 6 Typical Operation Sequence

The LCD module must first be initialised, the specifics of which is dealt with in the next section. The initialisation is actually a sequence of instructions that define the various parameters of the LCD module.

After the initialisation, you can then setup the display. This may involve setting the position of the cursor on the display. You may also set either the display or the cursor to shift, and the direction of the shift. This document, however, does not deal with this option in much detail. Check the manual for much information.

You can now write text on the display. The text will be displayed at whatever position of the screen the cursor is at.

Note that before every instruction the busy flag is checked. There are times during initialisation that the busy flag cannot be checked. The following section gives more details with respect to this.

OPERATION

Initialising the LCD

Before the LCD can be used to display text, it must first be initialised. Initialisation basically defines various parameters of the LCD module. For example, you must define whether the interface is 8-bit or 4-bit, whether the display is on or off, whether the cursor is on or off and whether the display is 1 line or 2 line. These are just a few of the parameters that must be defined. The following instructions define all the parameters that can be set.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (N), and character font (F).

Table 4 Instructions that define the LCDs parameters

I/D = 1: Increment I/D = 0: Decrement
S = 1: Accompanies display shift
DL = 1: 8 bits, DL = 0: 4 bits
N = 1: 2 lines, N = 0: 1 line
F = 1: 5 ´ 10 dots, F = 0: 5 ´ 8 dots

<u>Initialisation by Reset</u>

An internal reset circuit automatically initialises the LCD when the power is turned on. This though, is provided that the power to LCD meets certain conditions.

The conditions for proper initialisation by internal reset circuit:

1. The time the supply takes to rise from 0.2Volts to 4.5Volts (for 5Volt operation) or to 2.7Volts (for 3Volt operation) must have a minimum of 0.1ms and a maximum of 10ms
2. The time that the power supply is off (considered 0.2Volts) is a minimum of 1ms. This is to compensate for momentary power oscillations when the supply is switched on.

Table 5 shows the instructions that are executed during the initialisation:

Instruction	Parameter Settings
1. Display clear	
2. Function set:	DL = 1; 8-bit interface data N = 0; 1-line display F = 0; 5 × 8 dot character font
3. Display on/off control:	D = 0; Display off C = 0; Cursor off B = 0; Blinking off
4. Entry mode set:	I/D = 1; Increment by 1 S = 0; No shift

Table 5 Instructions executed during Initialisation by Reset

The busy flag is kept in the busy state (BF=1) until the initialisation ends.

The parameters set by the (initial reset circuit) initialisation (Table 5) may not be the desired parameters. You can change any of these parameters using the instructions given in Table 4. However, if you need to change the parameters defined by the function set instruction (DL, N, F), this instruction must be executed before any other instruction. From this point, the function set instruction cannot be executed unless the interface length is changed. The other parameters of the LCD can be changed at any time.

Initialisation by Instruction

If the proper power supply conditions for the internal reset circuit is not met, the LCD will not be initialised. Initialisation by instruction becomes necessary. The following diagrams show the procedures for 4-bit and 8-bit interfaces.

Power on

Wait for more than 15 ms after V_{CC} rises to 4.5 V

(Wait for more than 40 ms after V_{CC} rises to 2.7 V)

RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)

Wait for more than 4.1 ms

RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)

Wait for more than 100 μs

RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)

BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instuction time. (See Table 6.)

RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	1	1	N	F	*	*	Function set (Interface is 8 bits long. Specify the number of display lines and character font.) The number of display lines and character font cannot be changed after this point.
0	0	0	0	0	0	1	0	0	0	Display off
0	0	0	0	0	0	0	0	0	1	Display clear
0	0	0	0	0	0	0	1	I/D	S	Entry mode set

Initialization ends

Figure 7 Initialisation by Instruction for an 8-bit Interface

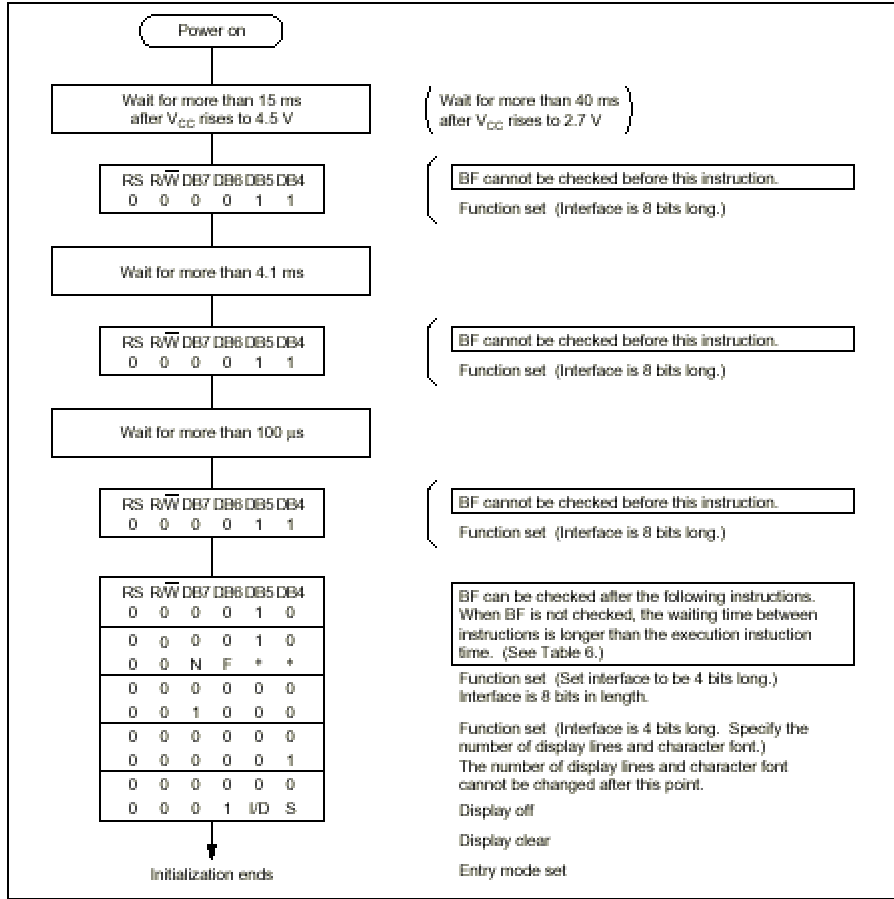

Figure 8 Initialisation by Instruction for a 4-bit Interface

<u>1-Line, 2-Line and 4-Line Initialisation</u>

It was mentioned before that initialisation defines the parameters of the LCD. One of these parameters is the number of display lines. This parameter is set by the Function Set instruction.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Function Set	0	0	0	0	0	1	N	F	*	*

N sets the number of display lines. For 1 display line, let N=0 when executing the function set instruction. For 2 or 4 display lines, set N to 1 when executing the function set instruction.

When using an LCD display that has 4 lines, for example a 20x4 display, when N=0, the display would be initialised as follows. You basically get 1line of 40 characters.

1st part of line 1 (20 characters long)
--
2nd part of line 1 (20 characters long)
--

When N=1, the display would be initialised as follows. You basically get 2 lines of 40 characters each.

1st part of line 1 (20 characters long)
1st part of line 2 (20 characters long)
2nd part of line 1 (20 characters long)
2nd part of line 2 (20 characters long)

What happens is that when writing 21 characters, starting at the first part of line one, the 21st character would end up on the second part of line one. If 41 characters were being written to the LCD, starting at the first part of line one, the 41st character would end up on the first part of line two.

To get around this arrangement, check the section, Setting Cursor Position.

Checking the busy flag

An integral part of any write instruction (to either the IR or DR) is checking the busy flag (BF). When the BF is 1 the LCD will not accept instructions. The next instruction must be written after ensuring the BF is 0.

When RS is 0 (to select the IR) and RW is 1 (to indicate a read operation), the busy flag is output to DB7 (for both 4-bit and 8-bit interfaces). The following diagrams illustrate how the busy flag is checked for both the 4-bit and the 8-bit interfaces.

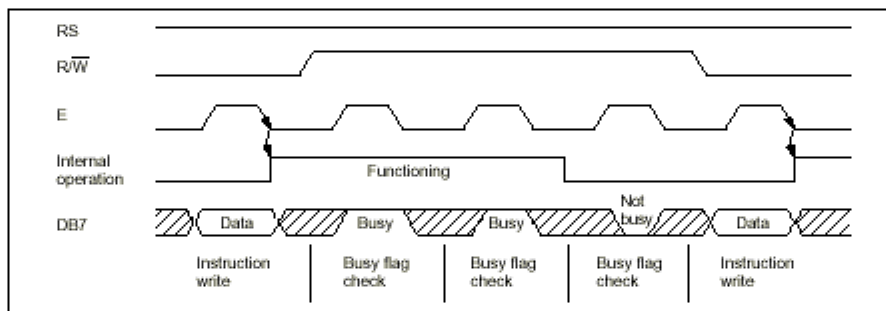

Figure 9 How to check the Busy Flag with an 8-bit Interface

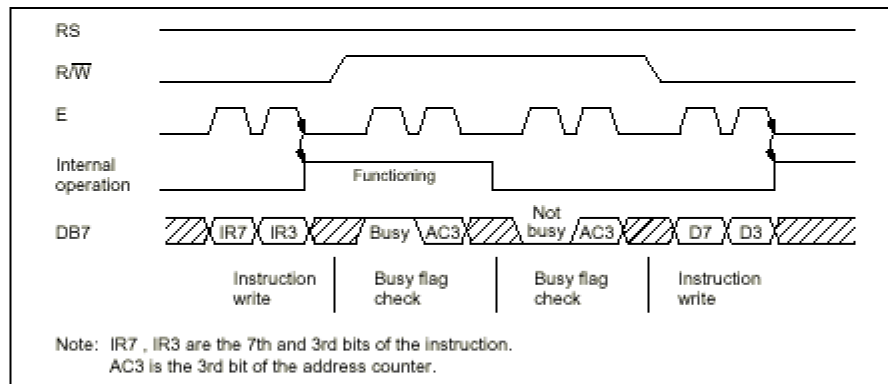

Figure 10 How to check the Busy Flag with a 4-bit Interface

Note that only when the busy flag is cleared (BF=0) would the LCD module accept a new instruction.

Writing text to the LCD

Writing text on the LCD is as easy as any write instruction. What you are in fact doing is writing data (RW=0) to the DR (RS=1). Only one character can be written to the LCD at a time. To write a character to the LCD, the following instruction must be executed.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Write character to the LCD	1	0	Code for Character							

The fact that the code for a character matches its ASCII code simplifies the implementation of the instruction. This is dealt with in the Implementation section of this document.

Setting cursor position

Setting the cursor position gives the user the ability to write a character anywhere on the LCD screen. Any character that is written to the LCD, is actually stored in the display data RAM (DDRAM). Every possible character position has a DDRAM address. Therefore to set the cursor position, we must execute the Set DDRAM Address instruction. This is write instruction (RW=0) to the IR (RS=0).

The following table shows the Set DDRAM Address instruction.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Set DDRAM Address	0	0	1	DDRAM Address						

The DDRAM addresses, though, are not completely continuous as the following diagram shows. Note that these addresses are in hex.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

Figure 11 The DDRAM Addresses for a 20x4 LCD

Although the above address map is 20 characters by 4 lines, smaller LCDs will have the same starting addresses. For example, the following shows the DDRAM addresses for a 16x2 LCD.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Figure 12 The DDRAM Addresses for a 16x2 LCD

To set the cursor to address 40h, for example, the following instruction is executed. Note, 40h is equivalent to 0b01000000.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Set DDRAM Address	0	0	1	1	0	0	0	0	0	0

Other functions

Clear Screen

This is a simple write instruction (RW=0). It calls the LCD module to execute one of its functions so we are writing to the IR (RS=0); The function clears the screen, as its name implies, and returns the cursor to home (address 0).

Cursor Home

This is also a simple write instruction (RW=0). It returns the cursor to home (address 0) but does not alter the contents of the DDRAM. This means that none of the text being displayed will be changed.

Summary of Instructions

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and shifts display without changing DDRAM contents.
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (N), and character font (F).
Set CGRAM address	0	0	0	1	CGRAM address						Sets CGRAM address. CGRAM data is sent and received after this setting.
Set DDRAM address	0	0	1	DDRAM address							Sets DDRAM address. DDRAM data is sent and received after this setting.
Read busy flag & address	0	1	BF	CGRAM / DDRAM address							Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.
Write data to CG or CCRAM	1	0	Write data								Writes data into DDRAM or CGRAM.
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.

Table 6 Instructions

I/D = 1: Increment I/D = 0: Decrement
S = 1: Accompanies display shift
S/C = 1: Display shift S/C = 0: Cursor move
R/L = 1: Shift to the right R/L = 0: Shift to the left
DL = 1: 8 bits, DL = 0: 4 bits
N = 1: 2 lines, N = 0: 1 line
F = 1: 5 ´ 10 dots, F = 0: 5 ´ 8 dots
BF = 1: Internally operating BF = 0: Instructions acceptable
DDRAM: Display data RAM
CGRAM: Character generator RAM

IMPLEMENTATION

This portion of the document deals with developing the code to use the LCD. The code is based on the PIC C Cross-Compiler by Custom Computer Services Inc.

Designing the module

To design the LCD module let us start at the lowest level. This is the level that actually interacts with the LCD. At this level, we either read a byte from the LCD or write a byte to the LCD. We must also specify which register (the DR or the IR) we are reading from or writing to. At this level, we are also concerned about the length of the interface (4-bit or 8-bit).

Moving to the middle level. At this level we want either to send a command to the LCD, to write a character to the screen, or to initialise the LCD. Also, it is realised that checking the busy flag is the only instruction that requires reading a byte.

Now at the high level, we are concerned with writing a string of characters, or going to a certain line of the display. We build upon the functions of the middle level, simplifying the way would use the LCD.

The following diagram illustrates how you can structure the LCD Module.

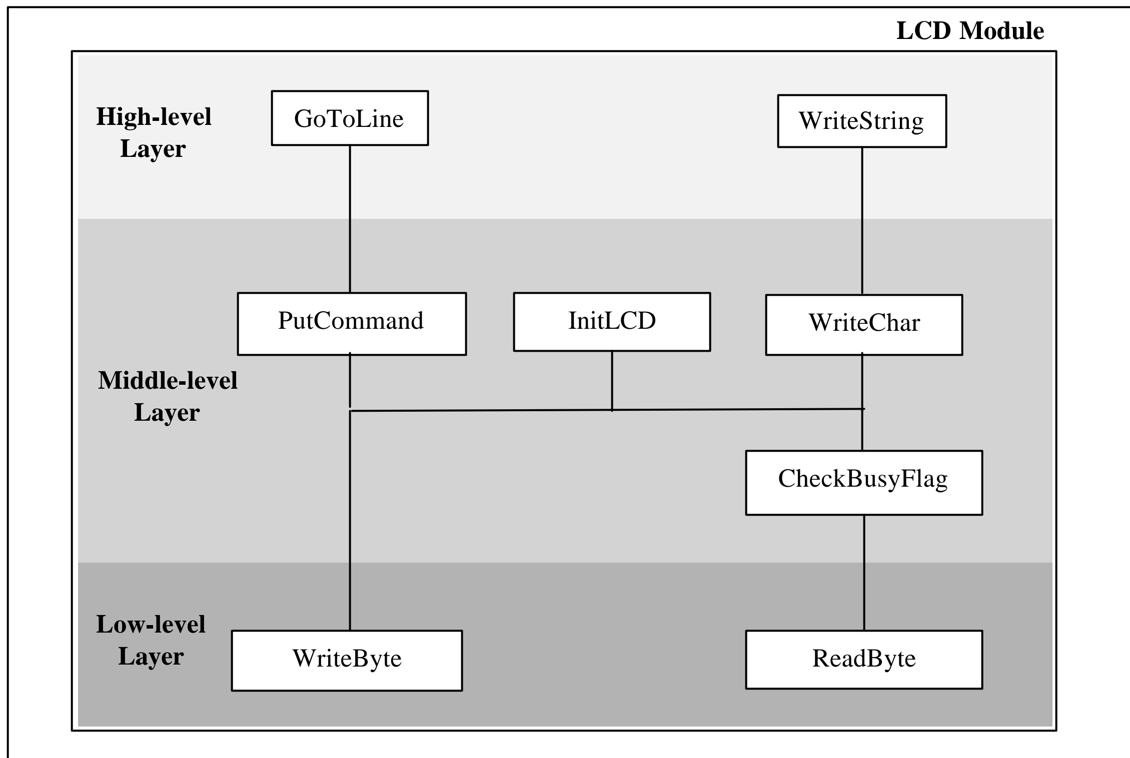

Figure 13 The LCD Module Structural Design

The advantage of such a layered approach is that the interface to the LCD gets simpler as we move to higher levels. It is easy to add functions to the high-level layer as at this level we are not concerned with the intricacies of the LCD. Also, changing between 4-bit and 8-bit interfaces require only changing the low-level layer. If done properly, the initialise function would not need changing.

General code

Before we can develop the functions shown in the design, we must determine which ports we are going to use to communicate with the LCD module. The code being developed will use Port D and last three bits of Port C. Note that you do not have to use these ports. However, using different connections would require the general code to be changed.

The following definitions make the code easier to read and use.

```
#define     IR              0
#define     DR              1
#define     READ            1
#define     WRITE           0
```

If you are using fast I/O, you must remember to define which pins are input and which are output. This would constant for the last 3 pins of Port C as EN, RS and RW are always output pins (from the PICs perspective). Port D however may be either input or output depending on whether the operation being performed is a read or a write.

The following sections show the general code needed for 8-bit and 4-bit modes. The addresses defined by this code are specific to the PIC16F877/874.

General code for 8-bit mode

The following diagram shows how the pins are connected.

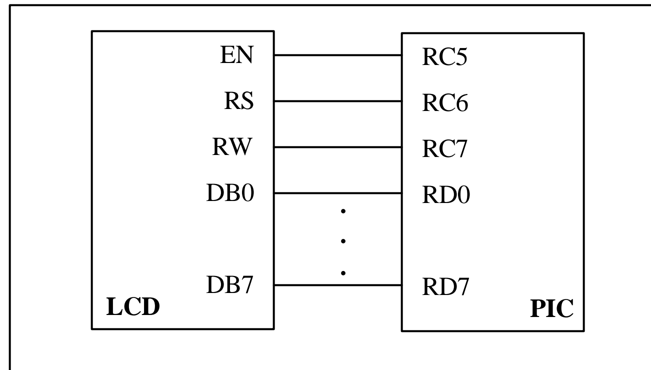

Figure 14 The Pin Connections for the LCD 8-bit Interface

The following portion of code defines the variables that the low-level functions would interact with.

```
/* Defines the bits for Port C */
struct {
      int unused:5;     //The first 5 bits are not used by the LCD
      int en:1;   //EN is the 6th bit of Port C        RC5
      int rs:1;   //RS is the 7th bit of Port C        RC6
      int rw:1;   //RW is the 8th bit of Port C        RC7
      }LCDControl;

#byte LCDData = 0x08    //Defines the address of the variable LCDData
                        //as that of Port D
#byte LCDControl = 0x07 //Defines the address of the structure
                        //LCDControl as that of Port C

#byte LCDDataDir =0x88  //Defines the address of the variable
                        //LCDDataDir as that of TrisD
#byte LCDConDir = 0x87  //Defines the address of the variable
                        //LCDConDir as that of TrisC

#define     LCD_DATA_IN       LCDDataDir|=0xFF
#define     LCD_DATA_OUT      LCDDataDir&=0x00
#define     LCD_CON_OUT       LCDConDir&=0x1F
```

To set the LCD control lines as output:

```
      LCD_CON_OUT;
```

To set the LCD data lines as output:

```
      LCD_DATA_OUT;
```

To set the LCD data lines as input:

```
      LCD_DATA_IN;
```

Writing a byte to the LCD module

To develop the code to write a byte to the LCD you may need to recap the write operation (Refer to Table 1) one of these (write) operations is required for an 8-bit interface, while a 4-bit interface requires the operation to be done twice.

We must determine the time between the two write operations required when using a 4-bit interface. The specifications say that there must be a minimum of 1000ns between 0-1transitions of EN. It is therefore necessary to find the fastest time the entire write operation can be completed. This time is determined to be 470ns as EN must be 1 for a minimum of 450ns, and RS and RW must be held constant for a minimum of 20ns after EN is cleared. We must therefore wait a minimum of 530ns between write operations. Note that this assumes that delays stated in the Control Lines section were used. As mentioned in that section, the delays stated allow for compatibly between the 2.7-4.5Volt range and the 4.5-5.5Volt range.

<u>Writing a byte 8-bit mode</u>

Now to develop the code for writing a byte, using an 8-bit interface. It is assumed that EN is initially low (EN=0). To ensure that this assumption is valid, we will ensure that EN is 0 after every operation. (This will also be applied to reading a byte)

In writing a byte, we must know to which register (DR or IR) we are writing, and also, the value that we are writing to this register. These would be the parameters that the WriteByte function accepts. So we have:

```
      WriteByte(short int rs, int data_to_lcd)
```

This function has no values to return. Additionally, we already know that RW should be 0 because this function only performs write operations.

The following code implements the write operation.

```
/***********************The WriteByte function ******************/
/*
      This function writes a byte to the LCD module with an 8-bit
      interface
Input Parameters:
int rs            This variable selects the register being written to.
                        DR selects the data register
                        IR selects the instruction register
int data_to_lcd   This variable stores the data that will be written to
                  the selected register
*/
void WriteByte(short int rs, int data_to_lcd)
{
      LCD_DATA_OUT;            //LCD Data Bus is an output
      LCDControl.rw = WRITE;  //The operation is a write operation
      LCDControl.rs = rs;     //Selects the register (DR or IR)
      delay_us(1);            //Wait a minimum of 60ns
      LCDControl.en = 1;      //Raise EN
      LCDData = data_to_lcd;  //Set the Data Bus to the desired value
      delay_us(1);            //Wait a minimum of 195ns
      LCDControl.en = 0;      //Clear EN
      delay_us(1);      //Keep RS and RW at their current states for a
                        //minimum of 20ns
                        //Also, keep the current value at the Data Bus
                        //for a minimum of 10ns
}
```

Reading a byte from the LCD module

Developing the code to read a byte from the LCD is very similar to what we previously did for the write operation. Again, you may need to recap the read operation (Refer to Table 2). Only one of these (read) operations is required for an 8-bit interface, while a 4-bit interface requires the operation to be done twice.

We must determine the time between the two read operations required when using a 4-bit interface. The specifications say that there must be a minimum of 1000ns between 0-1transitions of EN. It is therefore necessary to find the fastest time the entire read operation could be completed. This time is determined to be 470ns as EN must be 1 for a minimum of 450ns, and RS and RW must be held constant for a minimum of 20ns after EN is cleared. We must therefore wait a minimum of 530ns between read operations. Note that this assumes that delays stated in the Control Lines section were used. As mentioned in that section, the delays stated allow for compatibly between the 2.7-4.5Volt range and the 4.5-5.5Volt range.

<u>Reading a byte 8-bit mode</u>

Now to develop the code for reading a byte, using an 8-bit interface. In reading a byte, we must know from which register (DR or IR) we are reading. The ReadByte function only needs to accept this one parameter. So we have:

```
ReadByte(short int rs)
```

This function returns only one value, the byte read. Additionally, we already know that RW should be 1 because this function only performs read operations.

The following code implements the read operation.

```
/***********************The ReadByte function ******************/
/*
      This function reads a byte from the LCD module with an 8-bit
      interface
Input Parameters:
int rs            This variable selects the register being read from.
                        DR selects the data register
                        IR selects the instruction register
Output Value:     The function returns the value of the byte read
*/
int ReadByte(short int rs)
{
      int data_from_lcd;      //This variable is used to store the byte
                              //read from the Data Bus
      LCD_DATA_IN;            //Port D is an input port
      LCDControl.rw = READ;   //The operation is a read operation
      LCDControl.rs = rs;     //Selects the register (DR or IR)
      delay_us(1);            //Wait a minimum of 60ns
      LCDControl.en = 1;      //Raise EN
      delay_us(1);            //Wait a minimum of 360ns
      data_from_lcd = LCDData;//Read the value across the Data Bus
      LCDControl.en = 0;      //Clear EN
      delay_us(1);            //Keep RS and RW at their current states
                              //for a minimum of 20ns
      return data_from_lcd;
}
```

4-bit mode

General code for 4-bit mode

Note that when using a 4-bit interface, only 4 data lines are required. Therefore the entire Port D would not be needed to connect to the LCD module. Only half of Port D would be required. The following pin connection could be used.

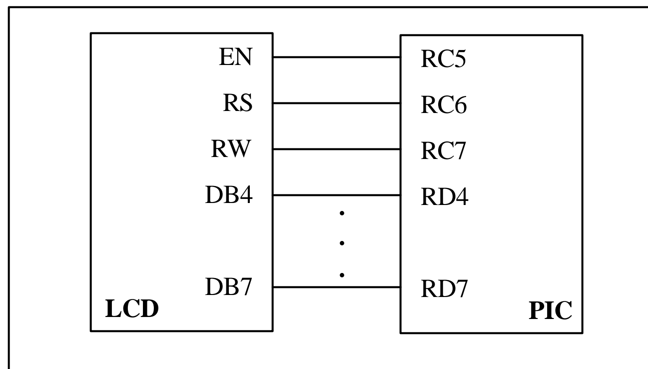

Figure 15 The Pin Connections for the LCD 4-bit Interface

The following portion of code defines the variables that the low-level functions would interact with.

```
/* Defines the bits for Port C */
struct {
      int unused:5;     //The first 5 bits are not used by the LCD
      int en:1;   //EN is the 6th bit of Port C        RC5
      int rs:1;   //RS is the 7th bit of Port C        RC6
      int rw:1;   //RW is the 8th bit of Port C        RC7
      }LCDControl;

/* Defines the bits for Port D */
struct {
      int unused:4;     //The first 4 bits are not used by the LCD
      int used:4;       //RD4 to RD7 used to communicate with the LCD
      }LCDData;

#byte LCDData = 0x08    //Defines the address of the structure LCDData
                        //as that of Port D
#byte LCDControl = 0x07 //Defines the address of the structure
                        //LCDControl as that of Port C

#byte LCDDataDir =0x88  //defines the address of the variable
                        //LCDDataDir as that of TrisD
#byte LCDConDir = 0x87  //defines the address of the variable
                        //LCDConDir as that of TrisC

#define     LCD_DATA_IN       LCDDataDir|=0xF0
#define     LCD_DATA_OUT      LCDDataDir&=0x0F
#define     LCD_CON_OUT       LCDConDir&=0x1F
```

To set the LCD control lines as output:

```
LCD_CON_OUT;
```

To set the LCD data lines as output:

```
LCD_DATA_OUT;
```

To set the LCD data lines as input:

```
LCD_DATA_IN;
```

Writing a byte 4-bit mode

Writing a byte using a 4-bit interface is now simple, having already developed the code to write a byte using an 8-bit interface. With a 4-bit interface we simply perform the write operation twice with a delay (of 550ns minimum) between the write operations. It is also necessary to break up the byte into its (two) nibbles. Note that the upper nibble is written in the first write operation. The lower nibble is written in the second operation.

The following code implements the write operation for a 4-bit interface.

```
/***********************The WriteByte function ******************/
/*    This function writes a byte to the LCD module with a 4-bit
      interface
Input Parameters:
int rs            This variable selects the register being written to.
                        DR selects the data register
                        IR selects the instruction register
int data_to_lcd   This variable stores the data that will be written to
                  the selected register
*/
void WriteByte(short int rs, int data_to_lcd)
{
      struct broken_up_data_t{      //This structure is used to break
                                    //up the byte to be written to the
                                    //LCD module into its two nibbles
            int lower_nibble:4;     //The first 4 bits are not used by
                                    //the LCD
            int upper_nibble:4;     //The second 4 bits are used to
                                    //communicate with the LCD
            };
      struct broken_up_data_t broken_up_data;
      broken_up_data = (struct broken_up_data_t)data_to_lcd;
      LCD_DATA_OUT;           //LCD Data Bus is an output
      LCDControl.rw = WRITE;  //The operation is a write operation
      LCDControl.rs = rs;     //Selects the register (DR or IR)
      delay_us(1);            //Wait a minimum of 60ns
      LCDControl.en = 1;      //Raise EN start first write operation
      LCDData.used = broken_up_data.upper_nibble;     //Set the Data
                              //Bus to the upper nibble of the desired
                              //value
      delay_us(1);            //Wait a minimum of 195ns
      LCDControl.en = 0;      //Clear EN finish first write operation
      delay_us(1);      //Keep RS and RW at their current states for a
                        //minimum of 20ns
                        //Also, keep the current value at the Data Bus
                        //for a minimum of 10ns
                        //Wait 530ns before the next write operation
      LCDControl.en = 1;      //Raise EN start second write operation
      LCDData.used = broken_up_data.lower_nibble;     //Set the Data
                              //Bus to the lower nibble of the desired
                              //value
      delay_us(1);            //Wait a minimum of 195ns
      LCDControl.en = 0;      //Clear EN finish second write operation
      delay_us(1);      //Keep RS and RW at their current states for a
                        //minimum of 20ns
                        //Also, keep the current value at the Data Bus
                        //for a minimum of 10ns
}
```

Reading a byte 4-bit mode

Reading a byte using a 4-bit interface is now simple, having already developed the code to read a byte using an 8-bit interface. With a 4-bit interface we simply perform the read operation twice with a delay (of 550ns minimum) between the read operations. Each read operation reads only one nibble so it is necessary to combine them to get a byte. Note that the upper nibble is read in the first read operation. The lower nibble is read in the second operation.

The following code implements the write operation for a 4-bit interface.

```
/************************The ReadByte function ******************/
/*    This function reads a byte from the LCD module with a 4-bit
      interface
Input Parameters:
int rs            This variable selects the register being read from.
                        DR selects the data register
                        IR selects the instruction register
Output Value:     The function returns the value of the byte read
*/
int ReadByte(short int rs)
{
      struct {    //This structure is used to form a byte
                              //from the two nibbles read
            int lower_nibble:4;     //The first 4 bits are not used by
                                    //the LCD
            int upper_nibble:4;     //The second 4 bits are used to
                                    //communicate with the LCD
            }data_from_lcd;

      LCD_DATA_IN;            //Port D is an input port
      LCDControl.rw = READ;   //The operation is a read operation
      LCDControl.rs = rs;     //Selects the register (DR or IR)
      delay_us(1);            //Wait a minimum of 60ns
      LCDControl.en = 1;      //Raise EN start first read operation
      delay_us(1);            //Wait a minimum of 360ns
      data_from_lcd.upper_nibble = LCDData.used;      //Read the value
                                                //across the Data Bus
      LCDControl.en = 0;      //Clear EN finish first read operation
      delay_us(1);      //Keep RS and RW at their current states for a
                        //minimum of 20ns
                        //Wait 530ns before the next write operation
      LCDControl.en = 1;      //Raise EN start second read operation
      delay_us(1);            //Wait a minimum of 360ns
      data_from_lcd.lower_nibble = LCDData.used;      //Read the value
                                                //across the Data Bus
      LCDControl.en = 0;      //Clear EN finish second read operation
      delay_us(1);            //Keep RS and RW at their current states
                              //for a minimum of 20ns
      return (int)data_from_lcd;
}
```

Checking the busy flag

This function reads the IR register, returning 1 if the LCD module is busy or 0 if it is not. So we have:

```
/***********************The CheckBusyFlag function *****************/
/*    This function reads a byte from the instruction register and
      tests the 8th bit, which is the busy Flag
Output Value:     The function returns
                        1 if the Busy Flag is set (LCD module busy)
                        0 if the Busy Flag is clear (LCD module is not
                        busy)
*/
short int CheckBusyFlag(void)
{
      int data_from_lcd;      //This variable is used to store the byte
                              //read from the LCD
      data_from_lcd = ReadByte(IR);       //Read the IR (rs=0)
      return (bit_test(data_from_lcd,7)); //Test the BF
                                          //Return 1 if set
                                          //Return 0 if clear
}
```

Note that the busy flag is checked to ensure that the LCD module is not busy before initiating a new instruction. The reason for this is that when the LCD is busy, it cannot accept a new instruction. The code that follows ensures that the busy flag is cleared before every instruction. Any code to be added to the middle layer has to conform to this rule. If this rule is not kept, instructions can be lost and the LCD module will not behave as expected. This rule has only to be kept at the middle layer, as high layer functions would not interact with the CheckBusyFlag function.

Initialising the LCD

To develop the InitLCD function let us assume that the power supply does not produce the conditions necessary for correct internal reset circuit initialisation. Even if the power supply does meet the conditions, the function will still work. You may want to recap the Initialisation by Instruction section of this document.

As mentioned before, initialisation defines various parameters of the LCD module. We must now decide on the setting of these parameters. The following parameters must be considered:

Parameter	Settings	
Interface	4-bit	8-bit
Number of display lines	1-line	2-line or 4-line
Cursor shift direction	Increment	Decrement
Font size	5x8dots	5x10dots
Display	On	Off
Cursor	On	Off
Cursor blink	On	Off

To demonstrate the code for this function, the following settings will be chosen:

Parameter	Setting
Interface	8-bit
Number of display lines	2-line or 4-line
Cursor shift direction	Increment
Font size	5x8dots
Display	On
Cursor	Off
Cursor blink	Off

The settings chosen determine the instructions that must be written to the LCD module. The following code initialises the LCD (with the above settings).

```
/************************The InitLCD function *******************/
/*    This function initialises the LCD module (Initialisation by
      instruction).
Initialisation Parameters:
      Interface               8-bit
      Number of display lines 2-line or 4-line
      Cursor shift direction  Increment
      Font size               5x8dots
      Display                 On
      Cursor                  Off
      Cursor blink            Cursor blink
*/
void InitLCD(void)
{
      delay_ms(15);                  //Delay a minimum of 15ms
      WriteByte(IR,0b00111000);      //Define function set
                  //8-bit interface, 2-line or 4-line display, 5x8 font
      delay_ms(5);                   //Delay a minimum of 4.1ms
      WriteByte(IR,0b00111000);      //Redefine function set
      delay_us(100);                 //Delay a minimum of 100us
      WriteByte(IR,0b00111000);      //Redefine function set
      while(CheckBusyFlag());        //Wait until BF = 0
      WriteByte(IR,0b00001100);      //Define display on/off control
                        //display on, cursor off, cursor blink off
      while(CheckBusyFlag());        //Wait until BF = 0
      WriteByte(IR,0b00000001);      //Clear Display
      while(CheckBusyFlag());        //Wait until BF = 0
      WriteByte(IR,0b00000110);      //Entry mode set
                  //cursor direction increment, do not shift display
}
```

To initialise the LCD with different parameter settings, you would need to change the bits of the various instructions. You may want to recap the Summary of Instructions section to review what parameters the various bits affect.

Writing a character to the LCD

This function would obviously accept the character that you want to write to the display. Using the WriteByte function simplifies this function. You simply need to remember that you are writing to the DR (RS=1). Also, we must ensure that the LCD module is no longer busy before continuing.

```
/***********************The WriteChar function ******************/
/*    This function displays a character on the LCD.
Input Parameters:
char character    This variable stores the character to be displayed on
                  the LCD.
*/
void WriteChar(char character)
{
      while(CheckBusyFlag()); //Wait until the LCD module is not busy
      WriteByte(DR,character);//Write character to DR
}
```

Sending a Command to the LCD

This function is similar to WriteByte. The only difference is that PutCommand writes only to the IR (RS=0). Additionally, this function waits for the busy flag to be cleared (the LCD module is not busy).

```
/***********************The PutCommand function ******************/
/*    This function writes a byte to the instruction register.
Input Parameters:
int command       This variable stores the byte to be written to the
                  instruction register.
*/
void PutCommand(int command)
{
      while(CheckBusyFlag()); //Wait until the LCD module is not busy
      WriteByte(IR,command);  //Write command to IR
}
```

Going to a Line on the LCD

In the Setting Cursor Position section, you learned about DDRAM addresses and setting the cursor to an address. The GoToLine function allows the user to set the cursor to the first address of any of the 4 lines. The following table recaps these addresses.

Line	Address
1	00h
2	40h
3	14h
4	54h

The input to the function is the line that you would like to go to. The follow code implements the function.

```
/***********************The GoToLine function ******************/
/*    This function sets the cursor to the first position of a
      specified line of the LCD.
Input Parameters:
int line          This variable selects the LCD line on which the
                  cursor is to be set.
*/
void GoToLine(int line)
{
      int address;      //This variable is used to determine the
                        //address at which the cursor is to be set
      switch (line)     //Set address to the first DDRAM address of the
                        //specified line
      {
            case 1:
            address = 0x00;
            break;

            case 2:
            address = 0x40;
            break;

            case 3:
            address = 0x14;
            break;

            case 4:
            address = 0x54;
            break;

            default:          //An undefined line set the cursor home
            address = 0x00;
            break;
      }
      bit_set(address,7);     //Bit 7 identifies the instruction as Set
                              //DDRAM address
      PutCommand(address);    //Set the DDRAM address
}
```

Writing a string of characters to the LCD

We have already developed a function that writes a character to the LCD. This function builds upon the WriteChar function, allowing the user to write a string of characters. It uses the PRINTF function of the cross compiler, which requires the STRING.H file to be included. The following statement must therefore be included at the beginning of the program.

```
#include <string.h>
```

The following code implements the function.

```
#define     TOTAL_CHARACTERS_OF_LCD 80
void WriteString(char LineOfCharacters[TOTAL_CHARACTERS_OF_LCD])
{
      printf(WriteChar,"%c", LineOfCharacters);
}
```

You simply place whatever text you want to write to the LCD within the inverted commas.