Controlling Text LCD HD44780 (or LM016L)

In this tutorial we will discuss about how and Text LCD work. Especially the most common Text LCD HD44780 (or LM016L).

I) Architecture of Text LCD

First of all to clearly understand how a Text LCD work we need to examine the architecture of a Text LCD.

A) Exterior:

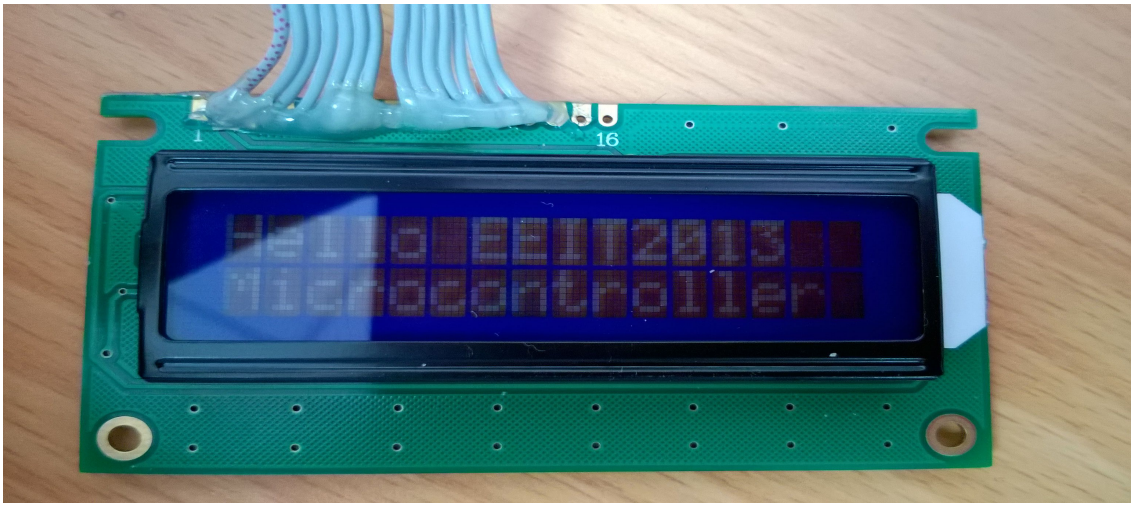


Figure 1: Screen of Text LCD

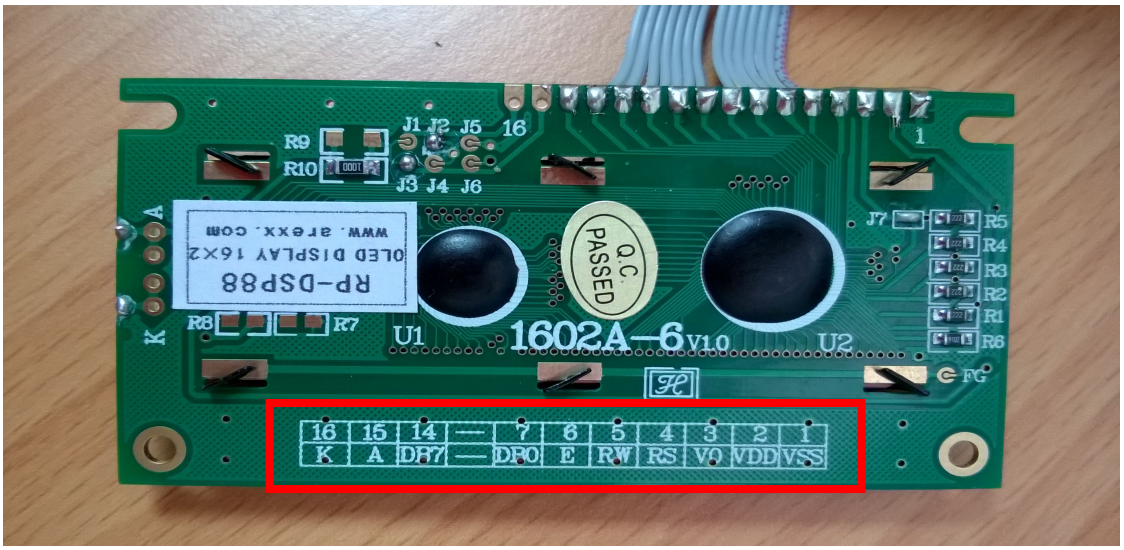


Figure 2: The bottom side of Text LCD

As we can see in the figure, on the screen of the LCD (figure 1) there are 32 separated display squares with 2 lines (16 display squares per line). Therefore we called this a 16x2 LCD. There is many type of LCD 16x4, 20x2, etc.. , but in this tutorial we only discuss how to controlled a 16x2 LCD. To control the others type we use similar principle.

Each square on the LCD can display only one character and that character must be available in the table below.

Table 1: Character Set

We can define other character for ourselves, but we will not discuss it here. The LCD can display 2 type of font, one is 5x7 and on is 5x8. But ***it is recommended to use 5x7 font.***

B) Interior:

On a Text LCD it is actually a controller IC and a flash memory which is actually separate into many areas.

1) Instruction Register

The first one to mention here is IR (Instruction Register). There are many functions that the controller IC can perform and Instruction Register is an 8-bit Register which we will use to send instructions to controller IC to tell it which function should be perform.

Now let look again on Figure 2 (above), we can see on the back of the LCD there is a pin-port map (marked by red-rectangular) to let us know how pins are connect to ports on LCD. We have here port DB0->DB7 is connected to pin7->pin 14. These ports will be used to send instruction to LCD. These instruction is describe in the table below (here we only consider DB7->DB0, and neglect port RS, R/W. We will discuss it later).

Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.	40μs~1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and enables/disables the display.	40μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blink of the character at the cursor position (B).	40μs
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N$	F	*	#	Sets the data width (DL), the number of lines in the display (L), and the character font (F).	40μs
Set CG RAM Address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs
Set DD RAM Address	0	0	1	A_{DD}							Sets the DD RAM address. Data may be written or read after making this setting.	40μs
Read Busy Flag & Address	0	1	BF	AC							Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.	1μs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	46μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	46μs
	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: 5x10 dots F = 0: 5 x 7 dots BF = 1: Busy BF = 0: Can accept data *# Set to 1 on 24x4 modules* *$ With KS0072 is Address Mode.*										DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM Address A_{DD}: DD RAM Address Corresponds to cursor address. AC: Address counter Used for both DD and CG RAM address.	Execution times are typical. If transfers are timed by software and the busy flag is not used, add 10% to the above times.

Table 2: Instruction Table

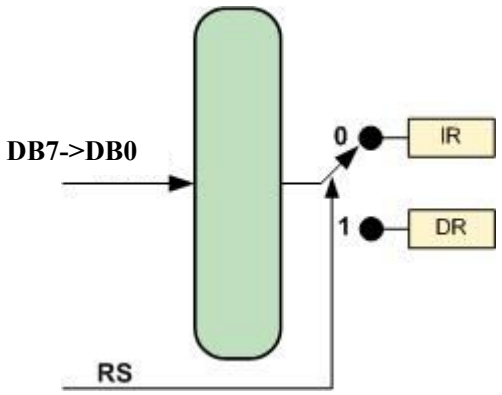
2) Data Register

To let the LCD know which character it should display, we need to send that character to the LCD via port DB7->DB0. Now we see that port DB7->DB0 is used to send Instruction and also Data. Therefore we will need another port call RS (Register Selector). If we send bit 0 to port RS, all the other bits that we send to port DB7->DB0 will be record to IR. Otherwise if we send bit 1 to port RS, all the bits in BD7->BD0 will be record to DR.

So if you want the LCD to clear its display we must first send bit 0 to port RS (because clear display is what we want the LCD to do for us, so it is an instruction), then we send the following bits to port DB7->DB0=0->0->0->0->0->0->0->1 (as can looked up on instruction table above).

Then if you want to send character 'A' to the LCD to let it display on the screen, you must first sending bit 1 to port RS, then we send the binary code of character 'A' to port DB7->DB0. Binary code of character 'A' is 01000001 (you could figure it out by searching for ASCII table), therefore we should send to port DB7->DB0=0->1->0->0->0->0->0->1. The last thing is that Data Register is also an 8-bits Register.

The figure below describe how RS port work as a register selecting switch.



Beside receiving data sent from controller, DR also have the ability to store data which are sent from LCD to be read by micro-controller. However there is only one situation that we can read data sent out from LCD: checking whether the LCD is busy or not. This situation happened because LCD is working much slower than the micro-controller so that we must wait for the LCD to finish its present task before sending another instruction or data.

To read data from LCD, first we need to consider port RW on the LCD. Most of the time we set RW port to bit 0, but when having to read data from the LCD we need to set it to bit 1. We all so need to set RS port to bit 1 because this time we need to transfer data from LCD to DR (which are connected to port DB7->DB0) and from then micro-controller can read. Then we also need to set all the ports on micro-controller which are connected to port DB0->DB7 on LCD, to be input ports.

If bit 7 of the DR (bit 7 of DR is connected to port DB7) is 1 , that means LCD is still busy and then we have to wait until is become 0.

3) DDRAM (Display Data RAM)

When we send a character to micro-controller via ports DB0->DB7, it will first be stored in DR register. Then the IC on LCD will transfer data from DR into DDRAM. To be clearly understand how DDRAM works, we first need to look at the architecture of DDRAM. DDRAM of HD44780 LCD is a memory which contains 80 8-bits registers in side it. Each register can be considered as a memory cell and can be described by the picture below.

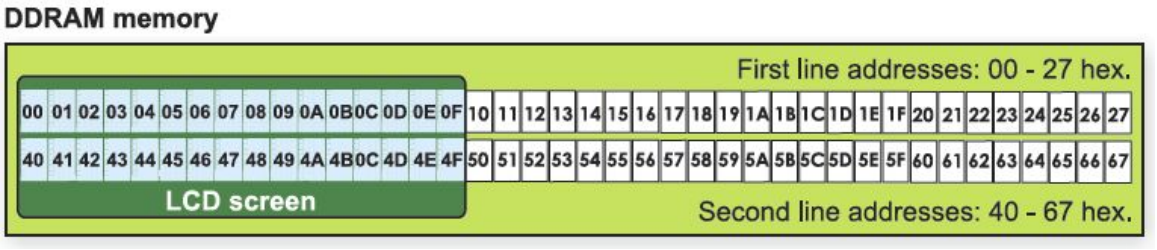


Figure 3: DDRAM Architecture

We notice that each memory cell on LCD have it own address which can be defined by binary, decimal or hex.

First let know that only characters that lies in memory cells which have the address from 0x00->0x0F and 0x40->0x4F, will be display on LCD screen. As can be see on Figure 3 above. So if we want to display a character on the screen we must tell the LCD to store it in one of the memory cell above and we will discuss how to do it right now.

When ever we send a new character to DR on LCD, the IC on LCD will transfer that character to a memory cell in DDRAM automatically. However there are four ways that a LCD will transfer a new character into DDRAM.

*** Way 1: new character** will be **transfer** to the memory cell that lies **next to the right** of the memory cell contains the previous character.

-> This mean if we first send character 'A' to the LCD, the LCD will first transfer that character to the first memory cell (which have the address 0x00). Then we send character "B", LCD will auto transfer character "B" into the memory cell 0x01 next to the right of 0x00.

*** Way 2: new character** will be **transfer** to the memory cell that lies **next to the left** of the memory cell contains the previous character.

* **Way 3: shift previous** character to the memory cell on the **right** and transfer new character into the memory cell which stored the previous character before.

-> This mean if we first send character 'A' to the LCD, the LCD will first transfer that character to the first memory cell(which have the address 0x00). Then if we send character "B" next, LCD will shift character "A" into the memory cell 0x01 and then transfer character "B" into memory cell 0x00.

***Way 4: shift previous** character to the memory cell on the **left** and transfer new character into the memory cell which stored the previous character before.

4) CGROM (Character Generator ROM)

Each display square on the LCD has 5x8 small dots, by enabling or disabling these dots we can generate a character. CGROM is a static memory in LCD which storing the definition to let LCD know which dot should be enabled and which dot should be disable and from then generating a character.

When we transfer a character to LCD, we actually transfer the ASCII code of that character into DR. Then that ASCII code will be transfer into DDRAM by LCD's IC. The LCD then will read that ASCII code and look for the definition of the character stored in the memory cell (inside CGROM) which has the address equal to the ASCII code.

*** Example:** From micro-controller we send character 'A' to the LCD, in the ASCII table character A has the hex code is 0x41. So we actually send the hex value 0x41 to the DR in LCD. The LCD then will transfer the hex value 0x41 into the memory cell 0x00 of DDRAM. Thus the LCD then first look in the memory cell (of CGROM) which have the address 0x41. Then the LCD will use the definition stored inside that memory cell to display the character 'A'. Otherwise, because hex value 0x41 is stored in the memory cell 0x00 of DDRAM so that LCD will display character 'A' on the first display square in the screen.

5) CGRAM (Character Generator RAM)

CGRAM is used to store definition of characters that we defined by ourselves. In this tutorial we will not discuss about this.

II) Controlling Port of Text LCD

A) RS (Register Selector)

As we discussed before, RS port on LCD is used to select whether DR or IR. Each bit on DR or IR (bit0->bit7) can be connected to DB0->BD7 just by switching RS. When set RS = bit 0, we connect bit0->bit7 of IR to DB0->DB7. And when set RS = bit 1, we connect bit0->bit7 of DR to DB0->DB7.

B) RW (Read-Write)

Usually when controlling an LCD we don't need to use RW port and we could just connect it to GND (0V = bit 0). Most of the time, we don't have to check whether the LCD is busy or not, because we just could simple delay the program about 50us->1.63ms. The delay time needed for each task on LCD could be looked up in Table 2 above. However, if you want to check the LCD for some reasons, make sure that you set this RW port to 1 so that the LCD will send it busy_flag bit.

C) E (Enable)

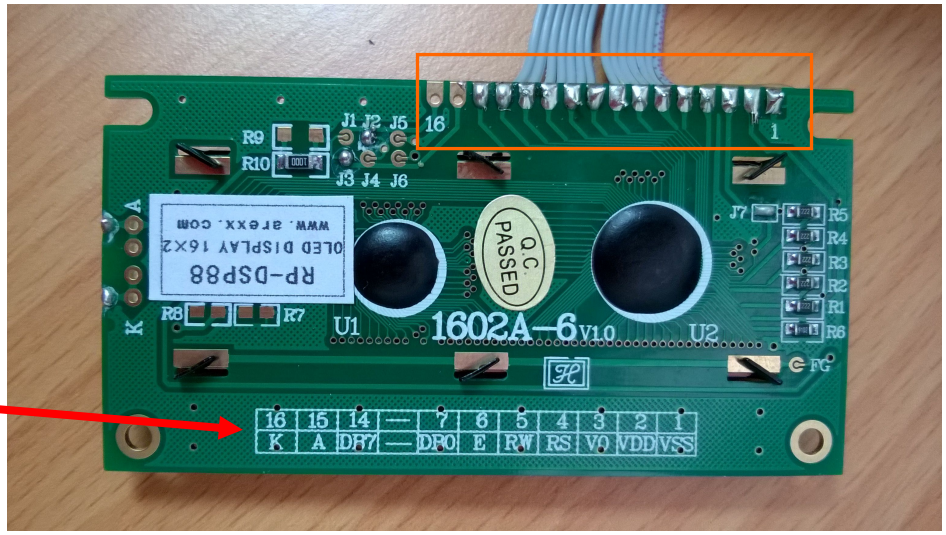
Enable port is the most confusing port here. Let just remember this.

+ always set E to 0 before you want to transfer anything to the LCD.
+ then transfer
+ finally set E to 1 and then set it to 0 again right after.

III)Let's do an real experiment

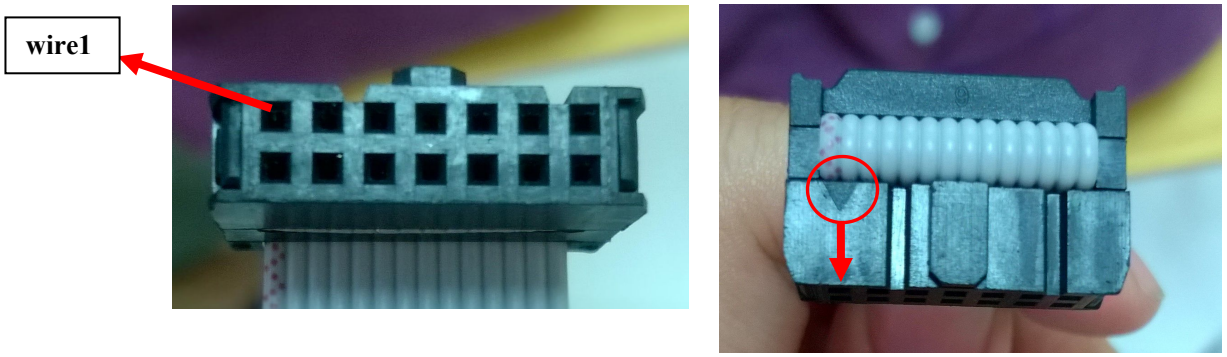
A) Hardware configuration

Step 1: First let look again at this figure below. That is the back-side of the LCD.



We can see that on the top of the LCD, there are 16 pins (from right to left: pin1->pin16). Pin1 to pin14 is already wired for us by a flat cable (marked by a red rectangle). We don't have to use pin 15 and 16 here. Also notice the flat cable, the wire with the red-strip will be wire1 and it is connected to pin1 on LCD. The wire next to wire1 is wire2 and then wire3 and so on..... Look again on the pin-port map, we see that pin1 is connected to VSS, pin2 is connected to VDD,....,pin7->pin14 is connected to DB0->DP7.

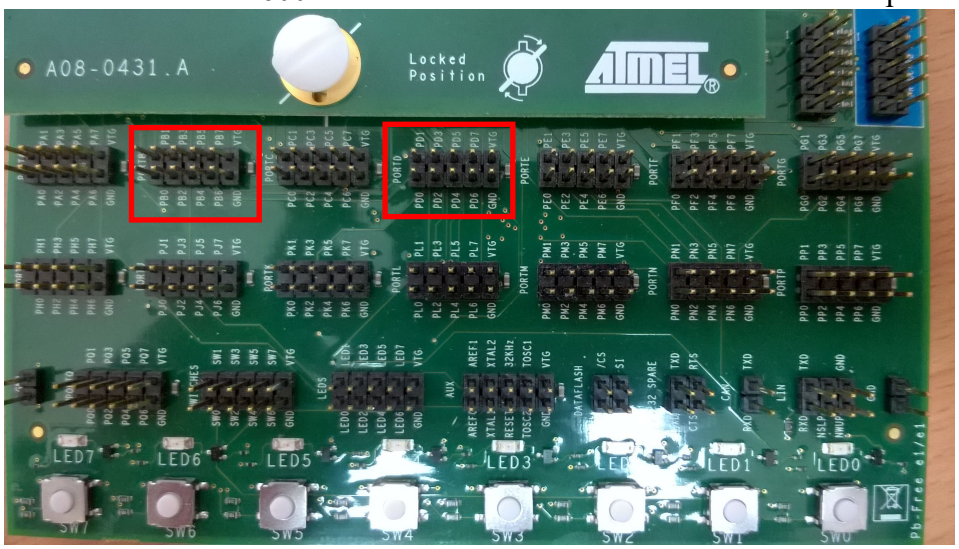
Now let look at the other head of the flat cable. There is a connector.



Notice that the hole with the little triangular on the top is connected to wire1. The other holes are connected to wires as the table below.

wire1	**wire 3**	**wire5**	**wire7**	**wire9**	**wire11**	**wire13**
wire2	**wire4**	**wire6**	**wire8**	**wire10**	**wire12**	**wire14**

Step 2: Next look at the STK600. We will use PORTB and PORTD for this experiment



Step 3: Let' connect the ports on LCD to the ports on STK600 as the table below by using male-female bus wire.

LCD Port 1	STK600 Port
VSS	GND
VDD	VCC (or VTG)
VO (OR VEE)	GND
RS	PD0
RW	PD1
E	PD2
DB0	PB0
DB1	PB1
DB2	PB2
DB3	PB3
DB4	PB4
DB5	PB5
DB6	PB6
DB7	PB7

B) Writing Software

An LCD can be controlled by 2 mode, 4bit mode (only use port DB4->DB7) and 8bit mode (use all port DB0->DB7). However in this tutorial we only use 8bit mode to control it.

1) Write driver library: 'LCDControl8BitMode.h'

Step 1: first create a library file and named it 'LCDControl8BitMode.h'.

Step 2: then we have to define some stuff first

```
#define F_CPU 16000000UL //define MCU clock-speed for precise delay
#define sbi(sfr,bit) sfr|=_BV(bit) //set port bit to 1 _BV(3) <=> 1<<3
#define cbi(sfr,bit) sfr&=~(_BV(bit)) //clear port bit to 0
#define DATA PORTB //use PORTB to send data from MCU to LCD
#define DDR_DATA DDRB //config data direction for PORTB
#define CTRL PORTD //use PORTD to communicate with Control Port on LCD
#define DDR_CTRL DDRD //config data direction for PORTD
#define RS 0 //port RS on LCD is connected to bit0 of CTRL port
#define RW 1 //port RW on LCD is connected to bit1 of CTRL port
#define E 2 //port E on LCD is connected to bit2 of CTRL port
#include <util/delay.h> //we need to use delay function here
#include <string.h>
```

Step 3: then we need to define function prototype

```
void InitLCD(); //some initial setup for LCD
void ClearLCD();
void Curs_home(); //move cursor to position 0x0 on the screen
void Curs_move(uint8_t row, uint8_t col); //move cursor to position at row-col
void WriteChar(uint8_t chr); //send character to LCD
void WriteString(char* str); //send a string to LCD
void WriteInstruction(uint8_t ascii_char); // send an instruction to LCD
```

Step 4: next we firt have to declare 'WriteInstruction' function

```
void WriteInstruction(uint8_t instruct)
{
    cbi(CTRL, RS); //clear RS port to 0
    cbi(CTRL, E); //clear E port to 0 before sending data
    DATA=instruct; //sending data
    _delay_us(80); //delay 80 micro-second to wait until the LCD finish reading data.
    sbi(CTRL, E); //set E port to 1
    cbi(CTRL, E); //clear E port immediately to 0
}
```

Step 5: next we declare for the 'WriteChar' function.

```
void WriteChar(uint8_t chr) //send a character to LCD to display it
{
  sbi(CTRL, RS); //this is sending data so we set RS port to 1
  cbi(CTRL, E);
  DATA=chr;
  _delay_us(80);
  sbi(CTRL, E);
  cbi(CTRL, E);
}
```

Step 6: Now we declare the Init Function

```
void InitLCD()
{
    _delay_ms(100); //wait until LCD is fully powered

  //set data direction of port B and D
    DDR_DATA=0b11111111; //set all pint on PORTB to be output
    DDR_CTRL=0b00000111; //set pin 0,1,2 on PORTD to be output

  //reset all control port on LCD to 0
    cbi(CTRL, RS);
    cbi(CTRL, RW);
    cbi(CTRL, E);

  //set 8 bit mode
    WriteInstruction(0b00111000); //tell LCD we want to controlled it by 8bit mode

  //turn LCD display off
    WriteInstruction(0b00001000); //let first turn LCD screen off to refresh it

  //display control
    WriteInstruction(0b00001100); //turn it on again and tell it don't display cursor and don't blinking

  //Entry mode set
    WriteInstruction(0b00000110); //tell it that we want new character to be transfered next to the right of
                                  // previous character. (Way 1)
}
```

Step 7: next is 'ClearLCD' , 'Curs_home' , 'Curs_move' functions. Notice that in writing character to LCD, we use an cursor to indicate where we want to write the character in. So if we want to write a character into the display square with row 2 and column 2, it means that we want to write that character to DDRAM's memory cell that have the address 0x41. Then we have to move out cursor to position 0x41 on DDRAM. That why we have the function Curs_move.

```
void ClearLCD()
{
  WriteInstruction(0b00000001);
}

void Curs_home()
{
  WriteInstruction(0b00000010);
}

void Curs_move(uint8_t row,uint8_t col)
{
/* because LCD only display characters that lie in DDRAM with memory cell address from 0x00->0x0F
   and from 0x40->0x4F, so we have to calculate the address of the memory cell to write the
   character into */
  uint8_t Ad;
  uint8_t instruct;
  Ad=0x40*(row-1)+(col-1); //calculate DDRAM address
  instruct=Ad+0x80; //add that address to the instruction 'Set DDRAM Address'
  WriteInstruction(instruct); // Send the instruction to set the cursor point to DDRAM address
}
```

Step 8: if you want to write a String in to LCD. Declare WriteString funtion like this.

```
void WriteString(char* str)
{
  unsigned char i;
  for (i=0;i<strlen(str);i++)
  {
    if (str[i]>0) WriteChar(str[i]); else WriteChar(' ');
  }
}
```

2) Write ‘main.c’ for testing

```
#ifndef F_CPU
#define F_CPU 16000000UL //check whether F_CPU has been defined or not
#endif
#include <avr/io.h>
#include <util/delay.h>
#include "LCDControl8BitMode.h"

int main(void)
{
    InitLCD();
    ClearLCD(); //clear LCD for sure
    Curs_home(); //reset cursor to first display square on row 1
    WriteString("Hello EEIT2013");
    Curs_move(2,1); //to cursor to the first display square on row 2
    WriteString("Micro-controller");
    while (1)
    {
    }
}
```

3) Finally we compile and load the program into STK600.

*** You can download file ‘main.c’ and ‘LCDControl8BitMode.h’ from here: https://goo.gl/UjIpLS**