University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering
Embedded Systems Laboratory 0907334


# 5 Experiment 5: LCD


## Objectives

- To become familiar with HD44780 controller based LCDs and how to use them
- Knowing the various modes of operation of the LCD (8-bit/4-bit interface, 2-lines/1-line, CG-ROM).
- Distinguishing between the commands for the instruction register and data register.
- Stressing software and hardware co-design techniques by using the *Proteus* IDE package to simulate the LCD.

# Introduction

*What is an LCD?*

A ***L***iquid ***C***rystal ***D***isplays (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. It is often utilized in battery-powered electronic devices because it uses very small amounts of electric power.

LCDs have the ability to display numbers, letters, words and a variety of symbols. This experiment teaches you about LCDs which are based upon the Hitachi HD44780 controller chipset. LCDs come in different shapes and sizes with 8, 16, 20, 24, 32, and 40 characters as standard in 1, 2 and 4–line versions. **However, all LCD's regardless of their external shape are internally built as a 40x2 format. See Figure 2 below**


Figure 1: A typical LCD module

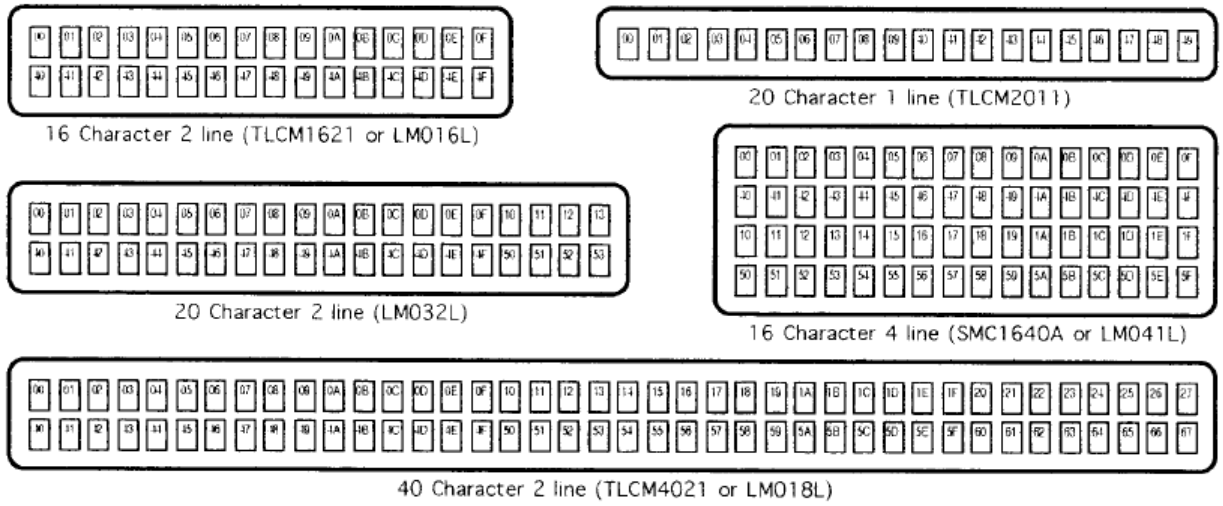



Figure 2: Different LCD modules shapes and sizes

| Display position | 1 | 2 | 3 | 4 | 5 | . . . . . | 39 | 40 |
|---|---|---|---|---|---|---|---|---|
| DDRAM address (hexadecimal) | 00 | 01 | 02 | 03 | 04 | . . . . . | 26 | 27 |
| | 40 | 41 | 42 | 43 | 44 | . . . . . | 66 | 67 |

Figure 3: Display address assignments for HD44780 controller based LCDs

*LCD I/O*

Most LCD modules conform to a standard interface specification. A 14-pin access is provided having eight data lines, three control lines and three power lines as shown below. Some LCD modules have 16 pins where the two additional pins are typically used for backlight purposes

Note: This image might differ from the actual LCD module, the order can be from left to right or vice versa therefore you should pay attention, pin 1 is marked to avoid confusion (printed on one of the pins).

Powering up the LCD requires connecting three lines: one for the positive power ***Vdd*** (usually +5V), one for negative power (or ground) ***Vss***. The ***Vee*** pin is usually connected to a potentiometer which is used to vary the contrast of the LCD display. We will connect this pin to the GND.

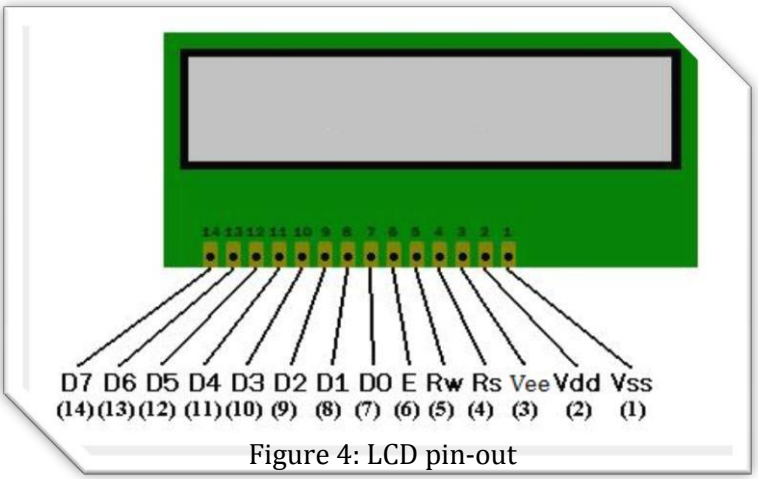


Figure 4: LCD pin-out

As you can see from the figure, the LCD connects to the microcontroller through three control lines: RS, RW and E, and through eight data lines D0-D7.

With 16-pin LCDs, you can use the L+ and L- pins to turn the backlight (BL) on/off.

| PIN NO | NAME | FUNCTION |
|---|---|---|
| L+ | Anode | Background Light |
| L- | Cathode | Background Light |
| 1 | Vcc | 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 |

Figure 5: LCD pin-out details

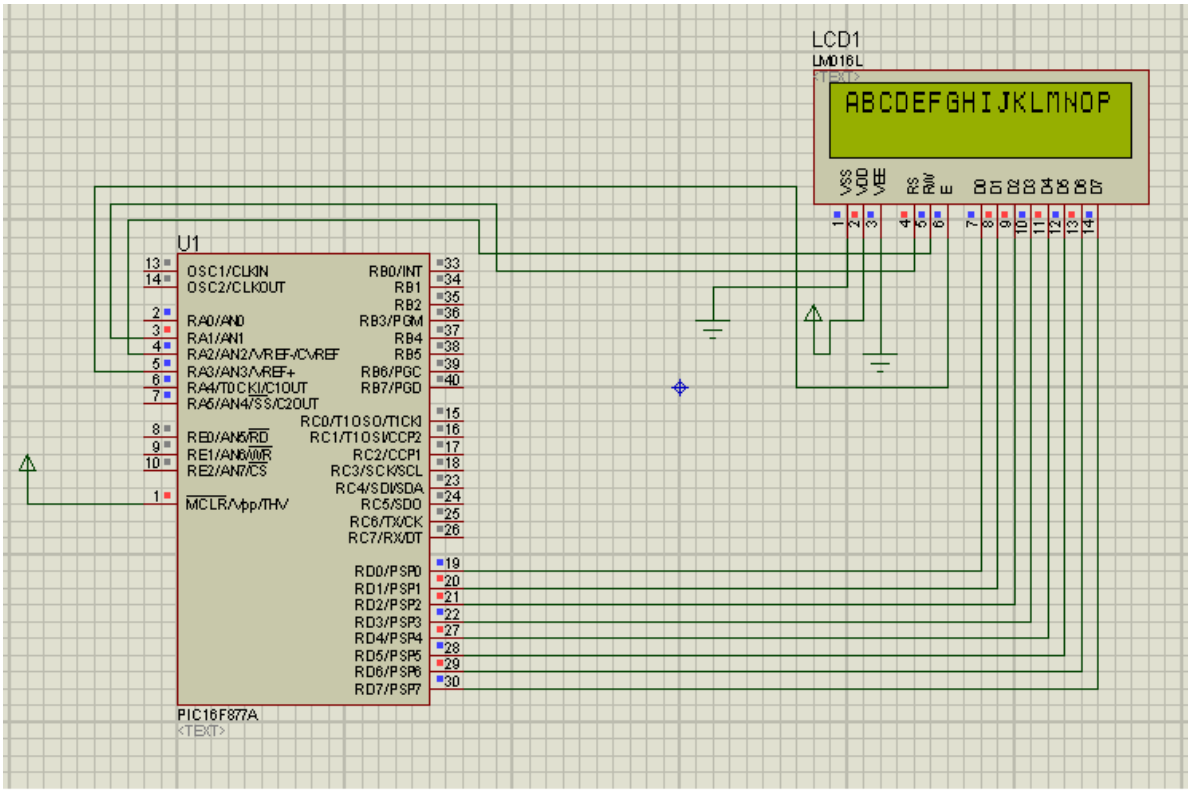



Figure 6: A typical interfacing between a PIC16F877A and an LCD module

When powered up, the LCD display should show a series of dark squares. These cells are actually in their off state. When power is applied, the LCD is reset; therefore we should issue a command to set it on. Moreover, you should issue some commands which configure the LCD. (See the table which lists all possible configurations below in the code and the explanation to each field)

**Sending Commands/Data to the LCD**

Using an LCD is a simple procedure once you learn it. Simply put you will place a value on the LCD lines D0-D7 (this value might be an ASCII value (character to be displayed), or another hexadecimal value corresponding to a certain command). So how will the LCD differentiate if this value on D0-D7 is corresponding to data or command?

Observe the figure below, as you might see the only difference is in the RS signal (**R**egister **S**elect), this is the only way for the LCD controller to know whether it is dealing with a character or a command!

| Command | RS | R/W | E | Binary D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Write Data to CG or DD RAM** | 1 | 0 | ↓ | ASCII Value | | | | | | | |
| **Write Command** | 0 | 0 | ↓ | Refer to the Command Table below | | | | | | | |

Figure 7: Necessary control signals for Data/Commands

Setting the necessary control signals in software:

For this experiment assume that **RS (Register Select)** is connected to PORTA1 , **R/W (Read/Write)** to PORTA2 (In this lab experiment we are only writing to the LCD, reading from the LCD is left to the student as home study)and **E(Enable)** is connected to PORTA3. Moreover, assume that the LCD lines D0-D7 are directly connected to PORTD.

we will introduce two subroutines; one will set the necessary control signals for sending a character (send_char), the other for sending a command (send_cmd).

```
1  send_char
2          movwf  PORTD
3          bsf    PORTA,1
3          bsf    PORTA, 3
3          nop
3          bcf    PORTA, 3
4          bcf    PORTA, 2
           call   delay
           return
```

Steps to send character to LCD

1.Place the ASCII character on the D0-D7 lines

2. Register Select (RS) = 1 to send characters

3. "Enable" Pulse (Set High – Delay – Set Low)

4. Delay to give LCD the time needed to display the character

```
1  send_cmd
2          movwf  PORTD
3          bcf    PORTA, 1
3          bsf    PORTA, 3
3          nop
3          bcf    PORTA, 3
4          bcf    PORTA,2
           call   delay
           return
```

Steps to send a command to LCD

1.Place the command on the D0-D7 lines

2. Register Select (RS) = 0 to send commands

3. "Enable" Pulse (Set High – Delay – Set Low)

4. Delay to give LCD the time needed to carry out the command

Table 1: Sending Characters/Commands Steps

**Displaying Characters**

All English letters and numbers (as well as special characters, Japanese and Greek letters) are built in the LCD module in such a way that it **conforms to the ASCII standard**. In order to display a character, you only need to send its ASCII code to the LCD which it uses to display the character.

To display a character on the LCD simply move the ASCII character to the working register (for this experiment) then call send_char subroutine.

Notice that from column 1 to D, the character resolution is 5 pixels wide x 7 pixels high (5x7) (column 0 is a special case, it is 5x8, but considered as 5x7, more on this later) whereas the character resolution of columns E and F is 5 pixels wide x 10 pixels high (5x10). We should change the resolution if we are to use characters from different resolution columns, this can be done using a command discussed later.

Figure 8: LCD Characters Map

| 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 |

Figure 9: LCD command control codes

To issue any of these commands to the LCD, all you have to do is place the command value in the working register, then issue the instruction "Call Send_cmd"

```
;*****************************************************************************************************
;                    Explaining the commands and their parameters in the LCD command table
;*****************************************************************************************************
```

### Clear Display

Moving the value 01 to the working register followed by "call send_cmd" will clear the LCD display, however the cursor will remain at it last position, so any future character writes will start from the last location, to reset the cursor position use the Display and Cursor Home command.

### Display and Cursor Home

Resets cursor location to position 00 of the LCD screen (Figure 3), future writes will start at the first location of the first line.

### Character Entry Mode

This command has two parameters 1/D and S:

**1/D:** By default, the cursor is automatically set to move from location 00 to 01 and so on (Increment mode). Suppose now that you are to write from right to left (as in the Arabic language), then you have to set the cursor to the Decrement mode.

**S:** Accompanies the D/C parameter, explained below

### Display On/OFF and Cursor

This command has three parameters:

**D:** Turns on the display (when you see the black dots on the LCD, it means that it is POWERED on, but not yet ready to operate), this command activates the LCD in order to be ready to use.

**U:** This displays the cursor (in the form of a horizontal line at the bottom of the character) when it is high and turns the cursor off when it is low

**B:** If the underline cursor option is enabled, this will blink the cursor if high.

### Display/Cursor Shift

All LCDs based on the HD44780 format - whatever their actual physical size is - are internally built in to be 40 characters x 2 lines with the upper row having the display addresses $0\text{-}27_H$ ($27_H$ = 39D → 0-39 =

40 Characters!!) and the lower row from $40_H$ -$67_H$. Now suppose you bought an LCD with the physical size of 20 char. x 2 lines, when you start writing to the LCD and the cursor reaches locations $20_D$, $21_D$, and $22_D$ …, you will not see them BUT don't worry, they are not lost. They were written in their respective locations but you could not see them because your bought LCD is 20 **visible** Characters wide from the outside and 40 from the inside. All you have to do is shift the display. So all you do is

1. Determine the direction of the shift (R/L)
2. Issue the shift Command D/C

**R/L**: Determines the direction of the shift, this might be useful if you are writing Arabic characters …
**D/C**: if this bit has a value of 0, the display is not shifted and the cursor moves the same way it was, if the its value is logic high, the display is shifted once, you might need to issue this command multiple times in order to shift the display by multiple locations!

### Function Set

This command has three parameters:
**8/4:** Eight/Four bits mode
8 – Bit interface: you send the whole command/character (8 bits) in one stage to the D0-D7 lines
4 – Bit interface: you send the command/character in two stages as nibbles to D4-D7 lines.
When to use the 4-bit mode?

1. Interfacing LCD with older devices which have 4-bit wide I/O Bus
2. You don't have enough I/O pins remaining, or you want to conserve the I/O pins for other HW

**2/1:** Line mode, determines whether you want to use the upper line of the LCD or both lines

**10/7**: Dot format, based on the LCD built-in characters table, note the following:

* 5x7 format (Default) is used whenever you use the characters found in columns 1 to D
* 5x7 format is also used whenever you use the built in characters in CG-RAM ***(EVEN THOUGH THE CG-RAM CHARACTERs ARE 5X8!!!)***
* 5x10 format is only used when displaying the characters found in columns **E** and **F**

********** *In LCD initialization, we normally set "**Clear Display", "Display and Cursor Home", "Display On/OFF" and "Cursor, and Function Set",** we place the value of the command then use the call send_cmd instruction.*
**********

### Set Display Address command

Syntax: 1AAAAAAA
This command allows you to move the cursor to whichever location you want, suppose you want to start writing in the middle of the display (assuming the ***visible*** width of the LCD screen is 20), then from Figure 2 you will observe that location 06 is approximately in the middle so you replace the A's with 06:
1*AAAAAAA* →1*0000110*→0x86
Moreover, suppose you wish to move to the second line which starts at location 40, same as above
1*AAAAAAA* → 1*1000000* → 0xC0
After calculating this value, you place it in the working register and then use the call send_cmd instruction.

```
;*******************************************************************************
;                                   EXAMPLE CODE 1
;*******************************************************************************
;  This code displays on the first “upper” row of the LCD the 26 English letters in alphabetical order
;  The code starts with LCD initialization commands such as clearing the LCD, setting modes and
;  display shifting.
;
; Outputs:
;       LCD Control:
;                               RA1: RS (Register Select)
;                               RA3: E   (LCD Enable)
;       LCD Data:
;                               PORTD 0-7 to LCD DATA 0-7 for sending commands/characters
; Notes:
;       The RW pin (Read/Write) - of the LCD - is connected to RA2
;       The BL pin (Back Light) – of the LCD – is connected to potentiometer
;*******************************************************************************
        include         "p16f877A.inc"
;*******************************************************************************
        cblock          0x20
                        tempChar        ;holds the character to be displayed
                        charCount       ;holds the number of the English alphabet
                        lsd             ;lsd and msd are used in delay loop calculation
                        msd
        endc
;*******************************************************************************
; Start of executable code
                org     0x000
                goto    Initial
;*******************************************************************************
; Interrupt vector
INT_SVC         org     0x0004
                goto    INT_SVC
;*******************************************************************************
; Initial Routine
; INPUT:        NONE
; OUTPUT:       NONE
; RESULT:       Configure I/O ports (PORTD and PORTA as output, PORTA as digital)
;               Configure LCD to work in 8-bit mode, with two lines of display and 5x7 dot format.
;               Set the cursor to the home location (location 00), set the cursor to the visible state
;               with no blinking
;*******************************************************************************
Initial
                Banksel TRISA           ;PORTD and PORTA as outputs
                Clrf    TRISA
                Clrf    TRISD
                Banksel ADCON1          ;PORTA as digital output
                Movlw   07
                mowf    ADCON1
                Banksel PORTA
                Clrf    PORTA
                Clrf    PORTD
                movlw   d'26'
```

```
54                  Movwf   charCount       ; initialize charCount with 26 Number of Characters in the English language
55                  Movlw   0x38            ;8-bit mode, 2-line display, 5x7 dot format
56                  Call    send_cmd
57                  Movlw   0x0e            ;Display on, Cursor Underline on, Blink off
58                  Call    send_cmd
59                  Movlw   0x02            ;Display and cursor home
60                  Call    send_cmd
61                  Movlw   0x01            ;clear display
62                  Call    send_cmd
63  ;*************************************************************************
64  ; Main Routine
65  ;*************************************************************************
66  Main
67                  Movlw   'A'
68                  Movwf   tempChar
69  CharacterDisplay                        ; Generate and display all 26 English Letters
70                  Call    send_char
71                  Movf    tempChar ,w     ; 'A' has the ASCII code of 65 decimal (0x41), by
72                  Addlw   1               ; adding 1 to it we have 66, which is B. Therefore, by
73                  movwf   tempChar        ; continuously adding 1 to tempChar we are cycling
74                  movf    tempChar ,w     ; through the ASCII table (here: alphabets)
75                  decfsz  charCount
76                  goto    CharacterDisplay
77  Mainloop
78                  Movlw   0x1c            ;This command shifts the display to the right once
79                  Call    send_cmd
80                  Call    delay
81                  Goto    Mainloop        ; This loop makes the character rotate continuously
82  ;*************************************************************************
83  send_cmd
84                  movwf   PORTD           ; Refer to table 1 on Page 5 for review of this subroutine
85                  bcf     PORTA, 1
86                  bsf     PORTA, 3
87                  nop
88                  bcf     PORTA, 3
89                  bcf     PORTA, 2
90                  call    delay
91                  return
92  ;*************************************************************************
93  send_char
94                  movwf   PORTD           ; Refer to table 1 on Page 5 for review of this subroutine
95                  bsf     PORTA, 1
96                  bsf     PORTA, 3
97                  nop
98                  bcf     PORTA, 3
99                  bcf     PORTA, 2
100                 call    delay
101                 return
102 ;*************************************************************************
103 delay
104                 movlw   0x80
105                 movwf   msd
106                 clrf    lsd
107 loop2
```

```
107                 decfsz  lsd,f
109                 goto    loop2
110                 decfsz  msd,f
111 endLcd
112                 goto    loop2
113                 return
114 ;*****************************************************************************
115                 End
116
```

### Set CG-RAM Address command
### Syntax: 01AAAAAA

If you give a closer look at Figure 8, you will clearly see that the table only contains English and Japanese characters, numbers, symbols as well as special characters! Suppose now that you would like to display a character not found in the built-in table of the LCD (i.e. an Arabic Character). In this case we will have to use what is called the CG-RAM (Character Generation RAM), which is a reserved memory space in which you could draw your own characters and later display them.

Observe column one in Figure 8, the locations inside this column are reserved for the CG-RAM. Even though you see 16 locations (0 to F), you only have the possibility to use the first 8 locations 0 to 7 because locations 8 to F are mirrors of locations 0 – 7.

So, to organize things, in order to use our own characters we have to do the following:

1. Draw and store our own defined characters in CG-RAM
2. Display the characters on the LCD screen as if it were any of the other characters in the table

***Drawing and storing our own defined characters in CG-RAM***

As stated earlier, we have eight locations to store our characters in. So how do we choose which location out of these to start drawing and building our characters in?
The answer is quite simple; follow this rule as stated in the datasheet of the HD44780 controller

1. To write (build/store a character in location 00 (crossing of the row and column)), you send the CG-RAM address command as follows: 01*AAAAAA* → 01*000000* → 0x40
2. However, to write in any location from 01 to 07, you have to skip eight locations (WHY?)
   So the CG-RAM address command will send **0x48** (to store a character in location 1**),** **0x50** (to store a character in location 2) and so on...

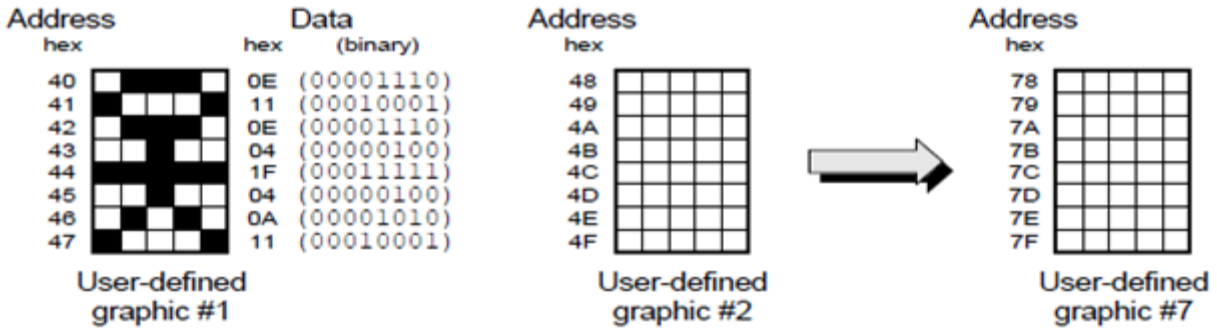
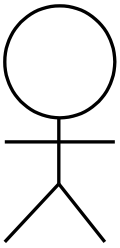



**Figure10 Showing how the CGRAM addresses correspond to individual pixels.**

So up to this point we have defined **where** to write our characters but not how to build them! This is the fun part☺, draw a 5x8 Grid and start drawing your character inside, then replace each shaded cell with one and not shaded ones with zero. Append three zeros to the left (B5-B7) and finally transform the sequence into hexadecimal format. This is the sequence which you will fill in the CG-RAM SEQUENTIALLY once you have set the CG-RAM Address before.



| B4 | B3 | B2 | B1 | B0 | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ■ | ■ | ■ | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | 0x0E |
| ■ | | | | ■ | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | | 0x11 |
| | ■ | ■ | ■ | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | 0x0E |
| | | ■ | | | → | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | → | 0x04 |
| ■ | ■ | ■ | ■ | ■ | | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | | 0x1F |
| | | ■ | | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | 0x04 |
| | ■ | | ■ | | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | | 0x0A |
| ■ | | | | ■ | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | | 0x11 |

```
1                                          Example
2   DrawStick1                             Setting the CGRAM address at which we draw the stick man
3               Movlw  0x40                ;Here it is address 0x00 in Figure 8 which transforms into
4               Call   send_cmd            ; command 0x40
5               Movlw  0X0E                Sending data that implements the Stick man
6               Call   send_char           ; Notice the address where to store the character in CG-RAM
7               Movlw  0X11                ;is a command thus use send_cmd, whereas the
8               Call   send_char           ;data bits of the stickman are sent as Data
9               Movlw  0X0E                ;using send_char
10              Call   send_char
11              Movlw  0X04
12              Call   send_char
13              Movlw  0X1F
14              Call   send_char
15              Movlw  0X04
16              Call   send_char
17              Movlw  0X0A
18              Call   send_char
19              Movlw  0X11
20              Call   send_char
21              Return
22
```

***Displaying the user generated (drawn) characters on the LCD screen***

Simply,if we stored our character in location 0, we move 0 to the working register then issue the **"call send_char"** command, if we stored it in location 2, move 2 to the working register and so on ….

```
;*******************************************************************************
;                                   EXAMPLE CODE 2
;*******************************************************************************
;  This code stores two shapes of a stickman, one in location 00 (of Figure 8), and another at location
; 01. The first stickman is written on the leftmost location of the upper line, the second stick man
; shape is also written above the first one, then the first stick man is rewritten on the same location
; that is display: first stickman shape → second stickman shape → first stickman shape and so on ..
; thus the stickman will appear as if it is moving ! ☺
;
; Outputs:
;       LCD Control:
;                               RA1: RS (Register Select)
;                               RA3: E   (LCD Enable)
;       LCD Data:
;                               PORTD 0-7 to LCD DATA 0-7 for sending commands/characters
; Notes:
;       The RW pin (Read/Write) - of the LCD - is connected to RA2
;       The BL pin (Back Light) – of the LCD – is  connected potentiometer
;*******************************************************************************
        include         "p16f877A.inc"
;*******************************************************************************
        cblock          0x20
                        lsd             ;lsd and msd are used in delay loop calculation
                        msd
        endc
;*******************************************************************************
; Start of executable code
                org     0x000
                goto    Initial
;*******************************************************************************
; Interrupt vector
INT_SVC         org     0x0004
                goto    INT_SVC
;*******************************************************************************
; Initial Routine
; INPUT:        NONE
; OUTPUT:       NONE
; RESULT:       Configure I/O ports (PORTD and PORTA as output, PORTA as digital)
;               Configure LCD to work in 8-bit mode, with two lines of display and 5x7 dot format.
;               Set the cursor to the home location (location 00), set the cursor to the visible state
;               with no blinking
;*******************************************************************************
Initial
                Banksel TRISA           ;PORTA and PORTD as outputs
                Clrf    TRISA
                Clrf    TRISD
                Banksel ADCON1          ;PORTA as digital output
                movlw 07
                mowf ADCON1
                Banksel PORTA
                Clrf    PORTA
                Clrf    PORTD
                Movlw   0x38            ;8-bit mode, 2-line display, 5x7 dot format
```

```
54                  Call    send_cmd
55                  Movlw   0x0e            ;Display on, Cursor Underline on, Blink off
56                  Call    send_cmd
57                   Movlw  0x02            ;Display and cursor home
58                   Call   send_cmd
59                   Movlw  0x01            ;clear display
60                   Call   send_cmd
61                   Call   DrawStick1      ;The subroutines draw and store the Stick man inside the
62                   Call   DrawStick2      ;CG-RAM. This DOES NOT mean that the character is
63                                          ;displayed on the LCD, it was only stored inside the CG-RAM
64                                          ;of the LCD.
65                   Movlw  0x01            ;the datasheet says you have to clear display command after
66                   Call   send_cmd        ;storing the characters or the code will not work
67
68  ;*****************************************************************************
69  ; Main Routine
70  ;*****************************************************************************
71  Main
72                   Movlw  0x00            ;Display character stored in location 00 (Figure 8), which in
73                   Call    send_char      ;this case is our first stickman in CG-RAM
74                   Movlw  0x02             ;Cursor Home Command
75                   Call    send_cmd
76                   Movlw  0x01            ;Display character stored in location 00 (Figure 8), which in
77                   Call    send_char      ;this case is our first stickman in CG-RAM
78                   Movlw  0x02             ;Cursor Home Command
79                   Call    send_cmd
80                   Goto    Main           ; This loop makes the character rotate continuously
81  ;*****************************************************************************
82  send_cmd
83                   movwf  PORTD           ; Refer to table 1 on Page 5 for review of this subroutine
84                   bcf    PORTA, 1
85                   bsf    PORTA, 3
86                   nop
87                   bcf    PORTA, 3
88                    bcf   PORTA, 2
89                   call   delay
90                   return
91  ;*****************************************************************************
92  send_char
93                   movwf  PORTD           ; Refer to table 1 on Page 5 for review of this subroutine
94                   bsf    PORTA, 1
95                   bsf    PORTA, 3
96                   nop
97                   bcf    PORTA, 3
98                    bcf   PORTA, 2
99                   call   delay
100                  return
101 ;*****************************************************************************
102 delay
103                  movlw  0x80
104                  movwf  msd
105                  clrf   lsd
106 loop2
107                  decfsz  lsd,f
```

```
108                 goto    loop2
109                 decfsz  msd,f
110 endLcd
111                 goto    loop2
112                 return
113 ;*****************************************************************************
114 DrawStick1                            Setting the CGRAM address at which we draw the stick man
115                 Movlw  0x40           ; Here it is address 0x00 in Figure 8 which transforms
116                 Call    send_cmd      ; into command 0x40
117                 Movlw  0X0E           ;Sending data that implements the Stick man
118                 Call    send_char
119                 Movlw  0X11
120                 Call    send_char
121                 Movlw  0X0E
122                 Call    send_char
123                 Movlw  0X04
124                 Call    send_char
125                 Movlw  0X1F
126                 Call    send_char
127                 Movlw  0X04
128                 Call    send_char
129                 Movlw  0X0A
130                 Call    send_char
131                 Movlw  0X11
132                 Call    send_char
133                 Return
134 ;*****************************************************************************
135 DrawStick2                            ;Setting the CGRAM address at which we draw the stick man
136                 Movlw  0x48           ;Here it is address 0x01 in Figure 8 which transforms
137                 Call    send_cmd      ; into command 0x48
138                 Movlw  0X0E           ;Sending data that implements the Stick man
139                 Call    send_char
140                 Movlw  0X0A
141                 Call    send_char
142                 Movlw  0X04
143                 Call    send_char
144                 Movlw  0X15
145                 Call    send_char
145                 Movlw  0X0E
146                 Call    send_char
147                 Movlw  0X04
148                 Call    send_char
149                 Movlw  0X0A
150                 Call    send_char
151                 Movlw  0X0A
152                 Call    send_char
153                 Return
154 ;*****************************************************************************
155                 End
```