University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp. 12: Interfacing with PIC

Objectives

1. Knowing the various modes of operation of the LCD (8-bit/4-bit interface, 2-lines/1-line, CG-RAM
2. Distinguishing between the commands for the instruction register and data register.
3. To become familiar with keypad Interfacing and usage.
4.

Introduction:

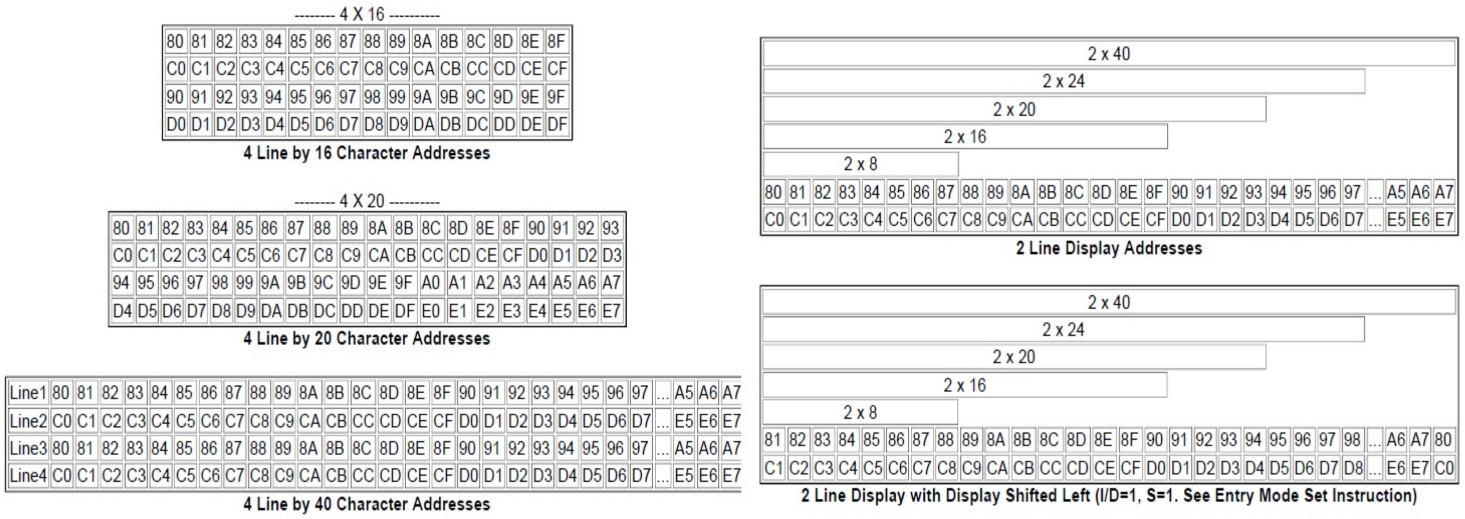
1- ***L***iquid Crystal ***D***isplays (LCD)

What is an LCD?

A **L**iquid Crystal **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 can 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, 10, 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



-------- 4 X 16 ---------

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF

4 Line by 16 Character Addresses

-------- 4 X 20 ---------

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7
D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7

4 Line by 20 Character Addresses

Line1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	...	A5	A6	A7
Line2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	...	E5	E6	E7
Line3	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	...	A5	A6	A7
Line4	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	...	E5	E6	E7

4 Line by 40 Character Addresses

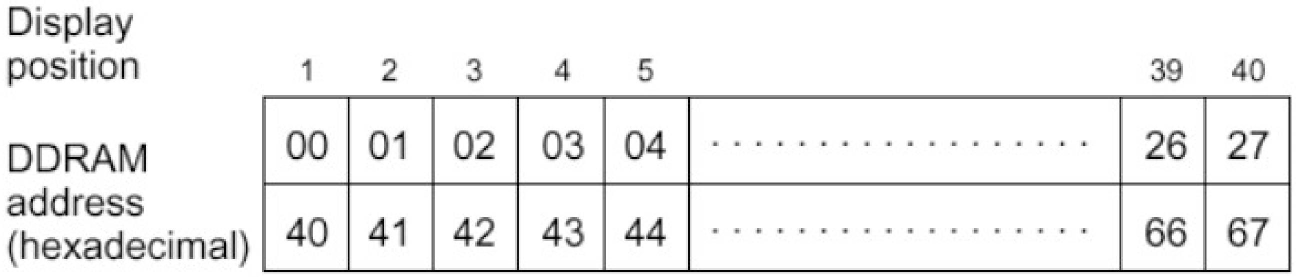
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	...	A5	A6	A7
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	...	E5	E6	E7

2 Line Display Addresses

81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	...	A6	A7	80
C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	...	E6	E7	C0

2 Line Display with Display Shifted Left (I/D=1, S=1. See Entry Mode Set Instruction)

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.

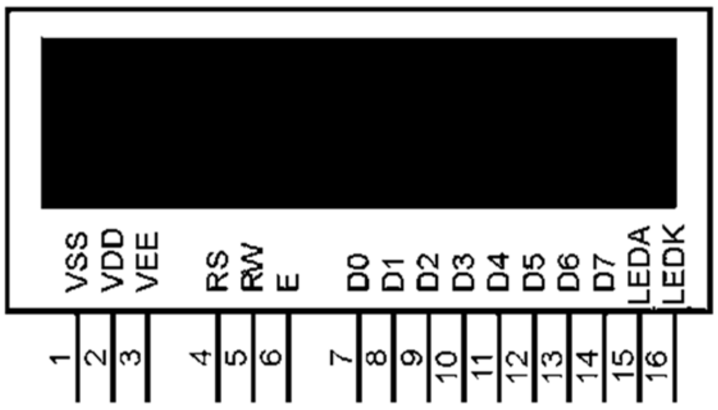


Figure 4: LCD Pinout

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.

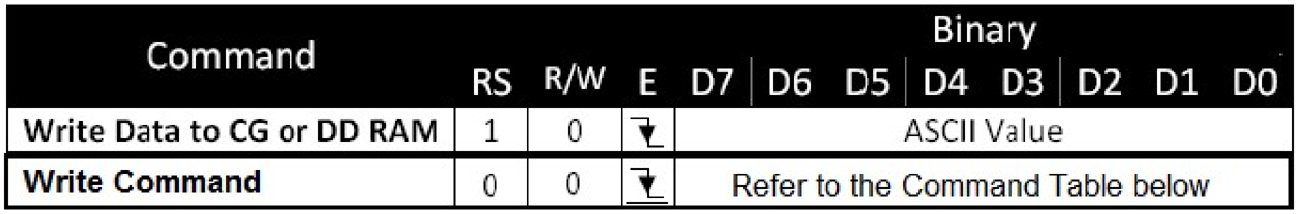
Table1: LCD pin-out details

Pin No.	Pin Name	Pin Type	Pin Description	Pin Connection
Pin 1	Ground	Source Pin	This is a ground pin of LCD	Connected to the ground of the MCU/ Power source
Pin 2	VCC	Source Pin	This is the supply voltage pin of LCD	Connected to the supply pin of Power source
Pin 3	V0/VEE	Control Pin	Adjusts the contrast of the LCD.	Connected to a variable POT that can source 0-5V
Pin 4	Register Select	Control Pin	Toggles between Command/Data Register	Connected to a MCU pin and gets either 0 or 1. 0 -> Command Mode 1-> Data Mode
Pin 5	Read/Write	Control Pin	Toggles the LCD between Read/Write Operation	Connected to a MCU pin and gets either 0 or 1. 0 -> Write Operation 1-> Read Operation
Pin 6	Enable	Control Pin	Must be held high to perform Read/Write Operation	Connected to MCU and always held high.
Pin 7-14	Data Bits (0-7)	Data/Command Pin	Pins used to send Command or data to the LCD.	In 4-Wire Mode Only 4 pins (0-3) is connected to MCU In 8-Wire Mode All 8 pins (0-7) are connected to MCU
Pin 15	LED Positive	LED Pin	Normal LED like operation to illuminate the LCD	Connected to +5V
Pin 16	LED Negative	LED Pin	Normal LED like operation to illuminate the LCD connected with GND.	Connected to ground

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	D7	D6	D5	D4	D3	D2	D1	D0
				Binary							
Write Data to CG or DD RAM	1	0	↓	ASCII Value							
Write Command	0	0	↓	Refer to the Command Table below							

Figure 5: Necessary control signals for Data/Commands

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**. 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 1x8, 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).

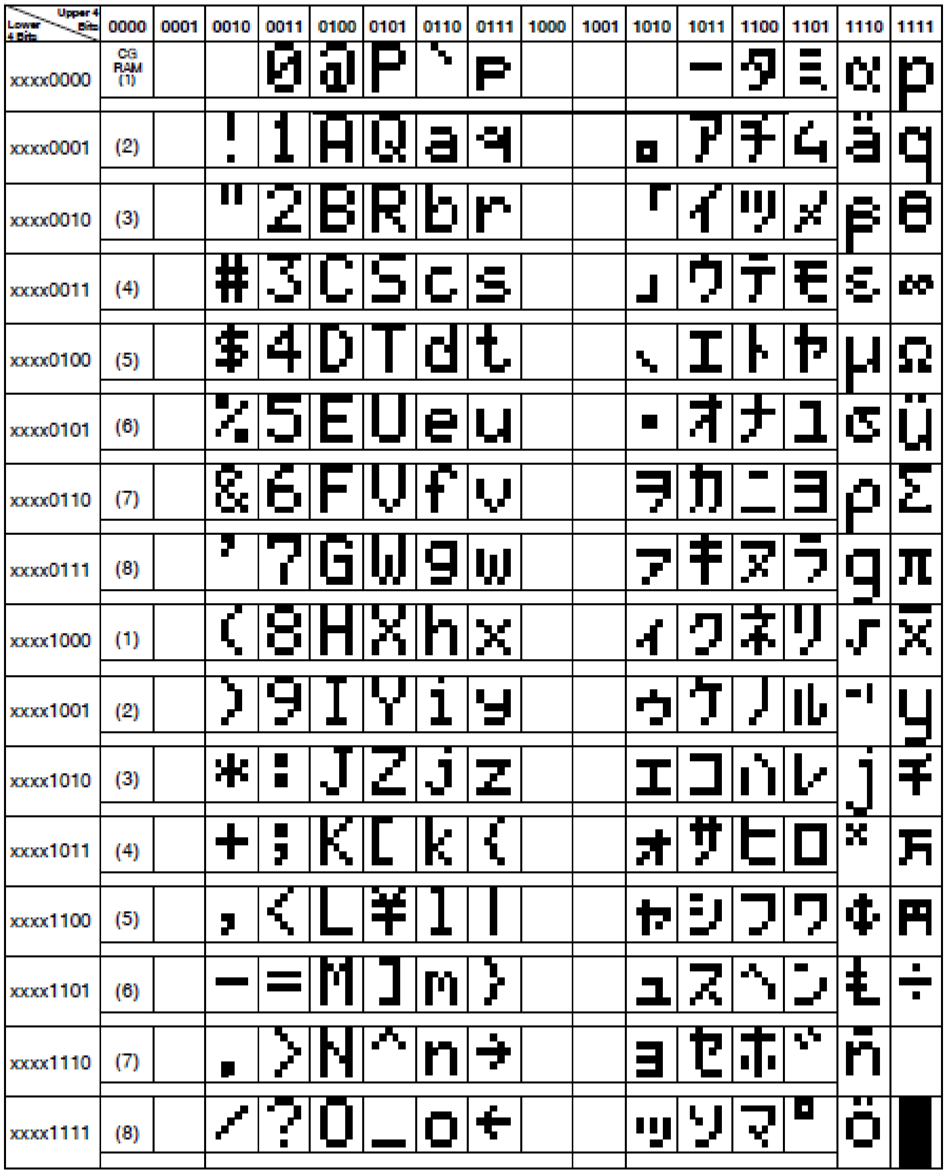
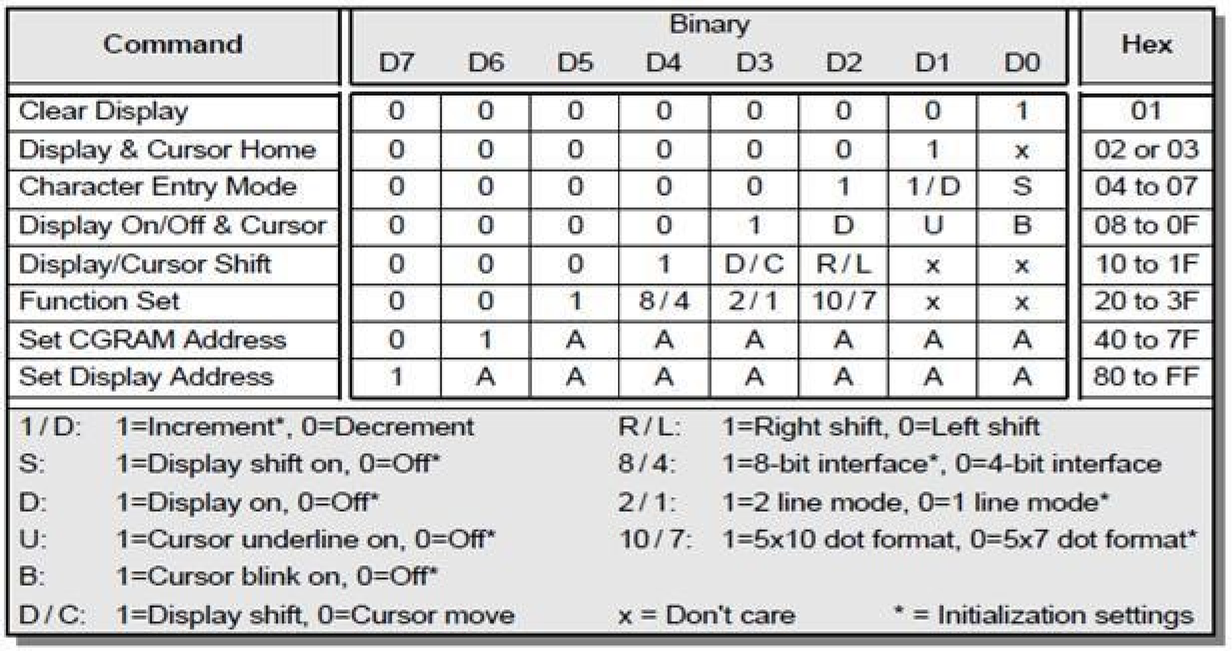


Figure6: Correspondence between Character Codes and Character Patterns (ROM Code: A00)



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 7: LCD command control codes

Set CG-RAM Address command Syntax: 01AAAAAA

If you give a closer look at Figure 6, 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 6, 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, to use our own characters, we must 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 must skip eight locations 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...

So up to this point we have defined **where** to write our characters but not how to build them. 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.

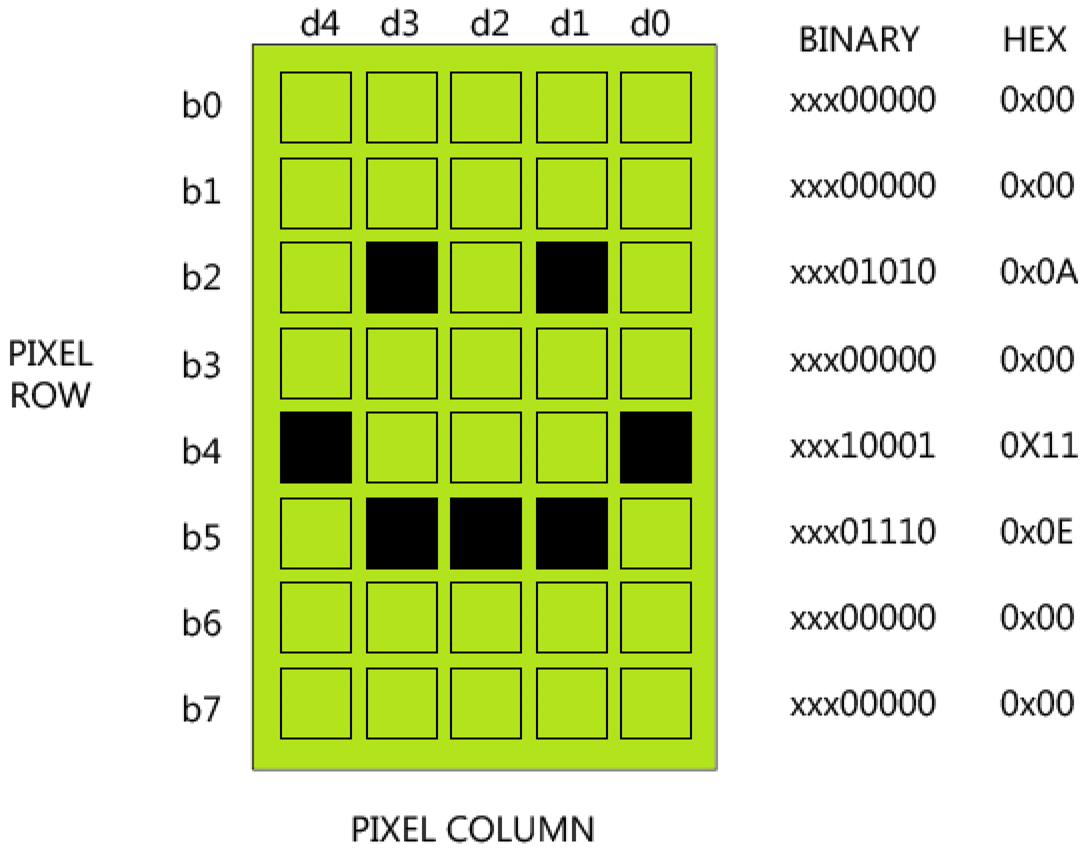


Figure 8: CG-RAM drawing example

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 ….

2- Keypad

Basic Keypad Theory

Keypads are essentially large switch arrays which allow data entries (numeric or alphanumeric) into systems. Keypads are widely used in everyday applications such as burglar alarms, cell phones and photocopiers. Keypads come in different shapes and sizes with 4x3, 4x4 buttons as common examples. It is not practical to connect each button in the keypad to its own port input as we previously did with switch and push buttons; therefore, keypads are normally constructed in a matrix format. An (n x m)

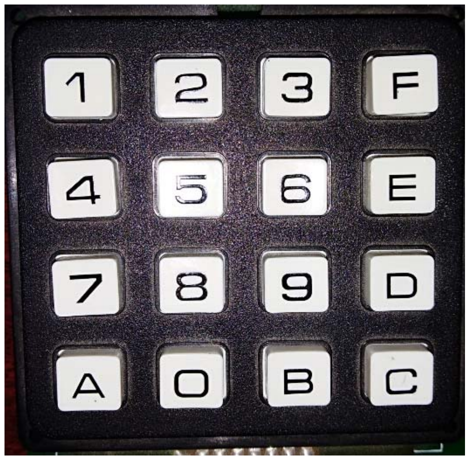


Figure 9: A 4x4 keypad

General Keypad Operation

In general, a keypad is interfaced in a way such that initially if no key is pressed you will read a certain logic level and when you press a button a signal with the negative of the original level will be read. You have two cases:
1. Fix the initial button state to be read as logic 1 (using pull-up resistors), when you press a button you will read logic 0.

2. Fix the initial button state to be read as logic 0 (using pull-down resistors), when you press a button you will read logic 1.

- Pull-up and pull-down resistors are used to limit the amount of current and protect the circuit. (not to read a floating state)

- Pull-up and pull-down resistors are normally connected externally, BUT you can make use of the internal pull-up resistors found in Microchip's PIC devices such as those in implemented in PORTB. In this experiment we will use the internal pull-up resistors.

Whether you use internal or external pull-up resistors the keypad will operate in the same way.

Technique

**It does not matter whether you start scanning rows or columns first it depends on your connections; the basic idea is the logic of the scanning technique is the same.

First the row bits are set to output, with the column bits as input. The output rows are set to logic 0. If no button is pressed all column line inputs will be read as logic 1 due to the action of the pull-up resistors. If, however, a button is pressed then its corresponding switch will connect column and row lines, and the corresponding column line will be read as low.

To detect this logic transition from high to low (that is to know whether a key has been pressed or not), we must either:
1. Keep pulling the inputs (columns) continuously until 0 is detected.
2. Make use of the interrupt (Here PORTB interrupt- on change will be beneficial)

Yet still, we have identified the column in which the key was pressed but not the button itself. So, what we do now is save the column and repeat the same procedure above with the following minor modification:

Secondly the column bits are set to output, with the row bits as input. The output columns are set to logic 0. Since the button is still pressed then its corresponding switch is still connecting column and row lines, and the corresponding row line will be read as low. If, however, the button is released all row line inputs will be read as logic 1 due to the action of the pull-up resistors. Now we have identified the row

Example

Suppose we connect the columns to PORTB 4-7 as input and the rows to PORTB 0-3 as output (with the value of 0). If we continuously read the inputs they will always be read as 1 because of the internal pull-up resistors on PORTB. If one presses number "7", this will make us read logic 0 on RB4 (identified that we have pressed a button in the first column).

Now let us exchange the inputs for the outputs, that is we connect the rows to PORTB 4-7 as input and the columns to PORTB 0-3 as output (with the value of 0).

If we read the input, we will find that RB2 is 0. Now we have identified the location of the pressed button and ready to process what it means.

So, what comes next?
The above scanning technique let us know the position of the pressed button (in terms of its row/column intersection) but not the value corresponding to the button. So obviously the next step is to use the location to retrieve the desired value.

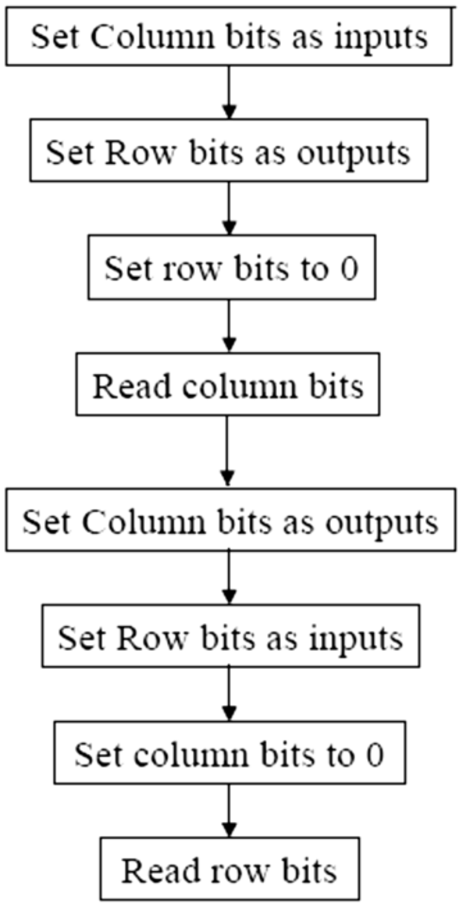


Table 2 - The values which will be read when a key is pressed

Key pressed	**Column** RB7, RB6, RB5, RB4	**Row** RB3, RB2, RB1, RB0	Look-up table index		
1	1110	1110	COL1	0	0 +0
4	1110	1101		1	0+1
7	1110	1011		2	0+2
A	1110	0111		3	0+3
2	1101	1110	COL2	4	4+0
5	1101	1101		5	4+1
8	1101	1011		6	4+2
0	1101	0111		7	4+3
3	1011	1110	COL3	8	8+0
6	1011	1101		9	8+1
9	1011	1011		10	8+2
B	1011	0111		11	8+3
F	0111	1110	COL4	12	12+0
E	0111	1101		13	12+1
D	0111	1011		14	12+2
C	0111	0111		15	12+3

Most often, you will need to display the number on a 7-segment display, LCD or use it in binary calculations. Therefore, it is natural to build a look-up table with the 7-segment representations, ASCII code or binary equivalent and use the location pattern which you saved (as in the table above) as an index to the look-up table.

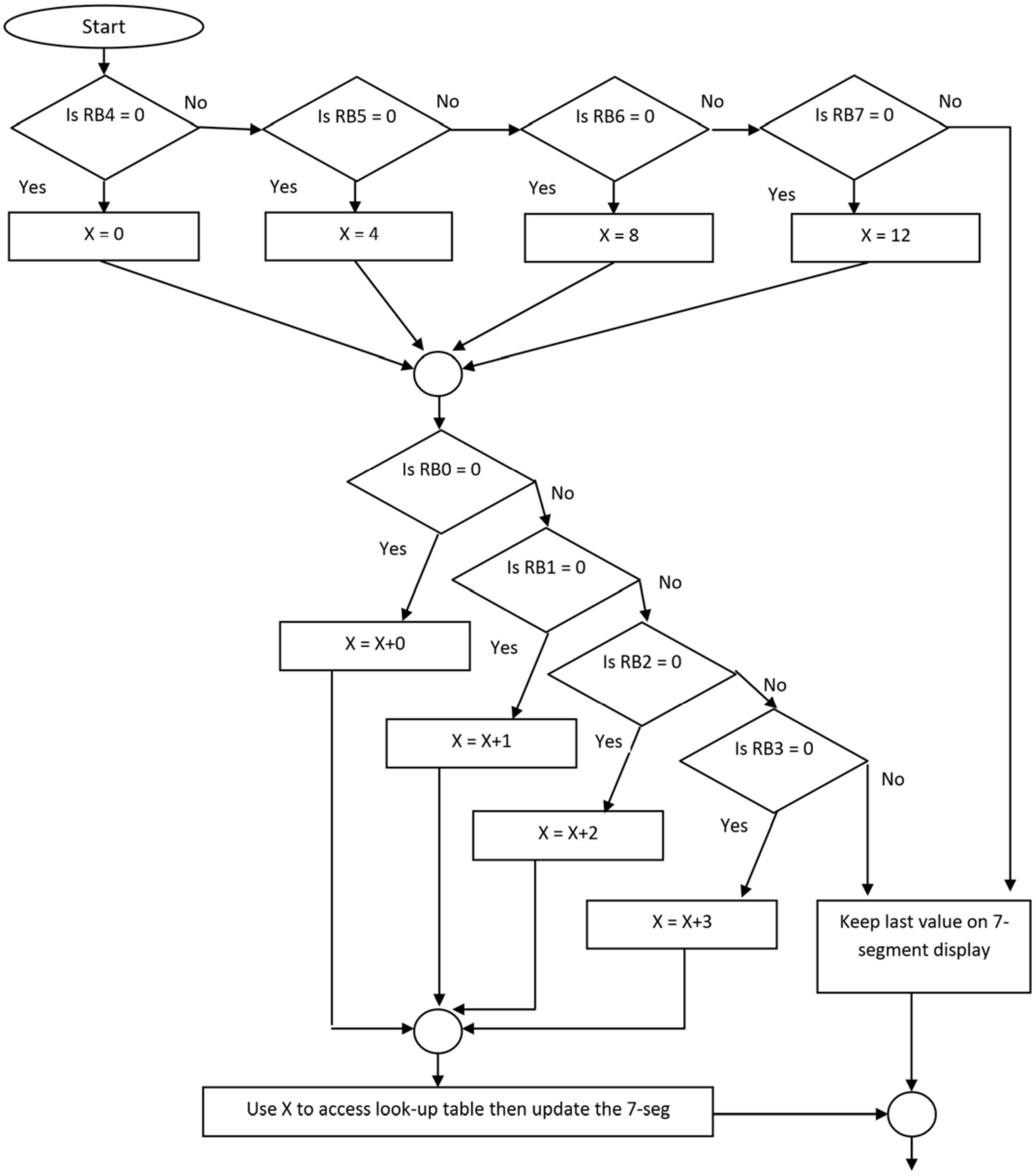
The way one organizes the look-up table entries differs from one person to another, therefore there is no specific way to translate the locations to their corresponding values. One might use a series of **btfsc or btfss** instructions or deduce a relationship and use mathematical operations or a combination of both.

What have we done in this experiment?

Study the following flowchart which is based on the table above.

1. If the key pressed is in column 1, then X might take the values 0, 1, 2, 3
2. If the key pressed is in column 2, then X might take the values 4, 5, 6, 7
3. If the key pressed is in column 3, then X might take the values 8, 9, 10, 11
4. If the key pressed is in column 4, then X might take the values 12, 13, 14, 15

These values will be added to PCL to retrieve values from the look-up table.


Start
Is RB4 = 0
Is RB5 = 0
Is RB6 = 0
Is RB7 = 0
No
No
No
No
Yes
Yes
Yes
Yes
X = 0
X = 4
X = 8
X = 12
Is RB0 = 0
No
Yes
Is RB1 = 0
No
Yes
Is RB2 = 0
No
Yes
Is RB3 = 0
No
Yes
X = X+0
X = X+1
X = X+2
X = X+3
Keep last value on 7-segment display
Use X to access look-up table then update the 7-seg