Lab 4 – Interfacing to keypad and LCD

Objectives

□ To learn to display an alphanumeric string on an LCD.

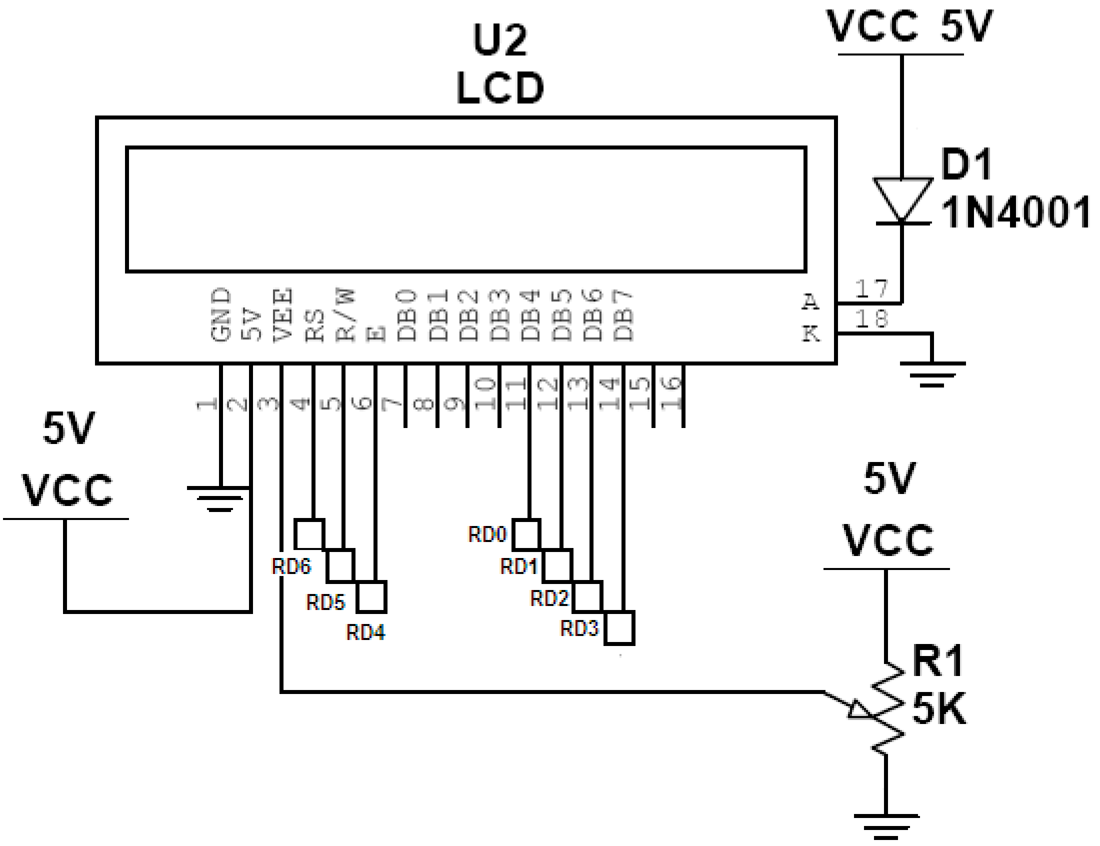
□ To learn to read an input from a 4x4 keypad (using a 74922 keypad encoder).

Introduction / Briefing

LCD at Port D

□ In this experiment, you will be displaying an alphanumeric string (numbers and characters) on an LCD connected to Port D. The LCD can display 2 lines of 16 or 20 characters.

□ Examine the connection below. Other than the power supply pins, you should be able to locate the pins VEE, RS, R/W, E, DB7-0.



□ The connections & purpose of the pins are shown below:

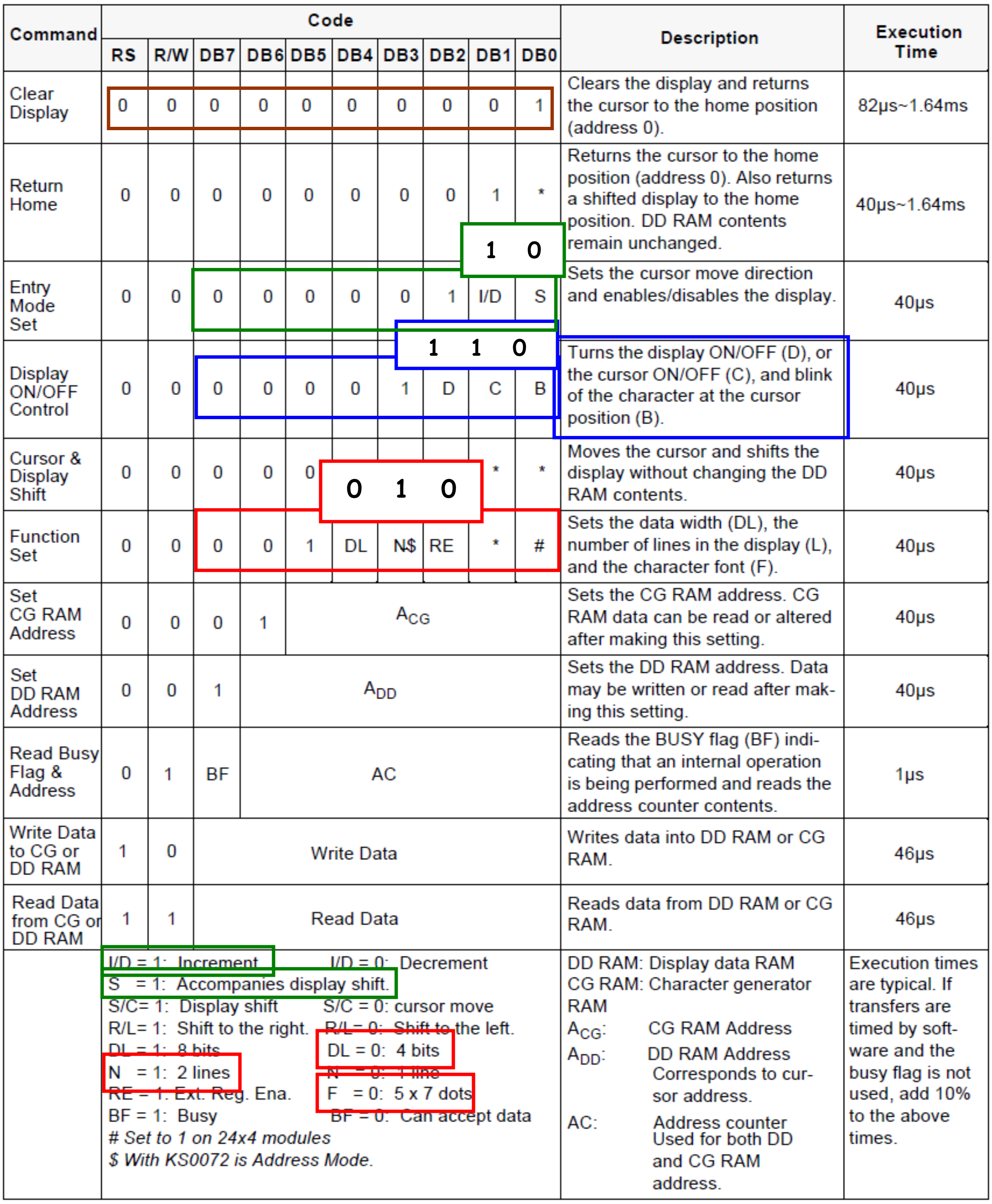
LCD pin	Connection	Remark / purpose
VEE	Variable resis	For contrast control.
RS	RD6	Register Select. Set RS = 0 to send "command" to LCD. Set RS = 1 to send "data" to LCD.
R/W	RD5	Set R/W = 0 to write to LCD. Set R/W = 1 to read from LCD.
E	RD4	Enable. Apply a falling edge (high to low transition) at E for LCD to latch on data / command at DB pins.
DB7-4	RD3-0	Use only DB7-4 in 4-bit mode, in which a byte of data/command is written as 2 nibbles.
DB3-0	Not connected	Use DB7-0 in 8-bit mode, in which a full 8-bit byte is written in one go.

An example of "command" is 0x01, which will clear the display.
An example of "data" is 0x41 – the character "A" to display on the LCD.

□ To make it easier for you to use the LCD, 4 functions have been written, based on the table above and the "commands for LCD module" on the next page. **You don't really have to understand the "fine prints" below or the table on the next page**.

void **lcd_write_cmd** (signed char cmd)	▪ A function for writing a command byte to the LCD in 4 bit mode. ▪ *If you look at the code, you will notice that RS is set to 0, and the command byte sent out as two nibbles.*
void **lcd_write_data** (char data)	▪ A function for writing a data byte to the LCD in 4 bit mode. ▪ *If you look at the code, you will notice that RS is set to 1, and the command byte sent out as two nibbles.*
*void **lcd_strobe** (void)*	▪ *A function for generating the strobe signal, i.e. a high to low transition at the Enable (E) pin.*
void **lcd_init** (void)	▪ A function for initialising the LCD. ▪ *The code configures Port D as an output port and set R/W to 0, so that data/command can be written to the LCD.* ▪ *The command lcd_write_cmd(0x28) or 0b001010xx puts the LCD into the 4-bit, 2 lines, 5x7 dots mode.* ▪ *The command lcd_write_cmd(0x0E) or 0b00001110 turns the display & cursor on.* ▪ *The command lcd_write_cmd(0x06) or 0b00000110 causes the cursor position to be incremented after every char.* ▪ *The command lcd_write_cmd(0x01) clears the display and returns the cursor to the home position.*

COMMANDS FOR LCD MODULE



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				*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N$	RE	*	#	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 RE = 1: Ext. Reg. Ena. 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.

- □ There is no need to start from scratch when you need to use LCD. You can modify an existing "main" function to suit your new application.

- □ In a typical "main" function (e.g. that of LCD2Lines.c below)
  - o The LCD is first initialised using **LCD_init().**
  - o Then, the cursor is move to the desired position
    - ▪ **lcd_write_cmd(0x80)** moves it to line 1 position 1 while
    - ▪ **lcd_write_cmd(0xC0)** moves it to line 2 position 1.
  - o The command **lcd_write_data (0x41)** write the letter "A" to the current cursor position etc.

```
// LCD2Lines.c
void main(void)
{
 lcd_init();                          // Initialise LCD module

 while(1)
 {
   lcd_write_cmd(0x80);               // Move cursor to line 1 position 1
   lcd_write_data(0x41);              // write "A" to LCD
.....
```

Binary patterns for different characters

LOWER 4 BITS / UPPER 4 BITS	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p		ー	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
1101	(6)	-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Q1: Fill in the blanks below to show how you can display "HELLO" on the first line of the LCD, and "WORLD" on the second line.

```
// Hello World.c
void main(void)
{
 ________________                    // Initialise LCD module

 while(1)
 {
  ________________                   // Move cursor to line 1 position 1
  lcd_write_data(0x_____);           // write "H" to LCD
  lcd_write_data(0x_____);           // write "E" to LCD
  lcd_write_data(0x_____);           // write "L" to LCD
  lcd_write_data(0x_____);           // write "L" to LCD
  lcd_write_data(0x_____);           // write "O" to LCD

  ________________                   // Move cursor to line 2 position 1
  lcd_write_data(0x_____);           // write "W" to LCD
  lcd_write_data(0x_____);           // write "O" to LCD
  lcd_write_data(0x_____);           // write "R" to LCD
  lcd_write_data(0x_____);           // write "L" to LCD
  lcd_write_data(0x_____);           // write "D" to LCD
  while(1);                          //stop here for now
 } // while
} // main
```

Q2: What do you think is achieved by the code below?

Your answer: ____

```
unsigned char K, outchar;
char Message [ ] = "Enter PIN number :  ";   // Defining a 20 char string

void main(void)
{
   lcd_init();                     // Initialise LCD module

   while(1)
   {
      lcd_write_cmd(0x80);         // Move cursor to line 1 position 1
      for (K = 0; K < 20; K++)     // for 20 char LCD module
      {
         outchar = Message[ K ];
         lcd_write_data(outchar);  // write character data to LCD
      }
...
```

Q3: What changes do you need to make to display "Welcome to SP"?

Your answer: ____

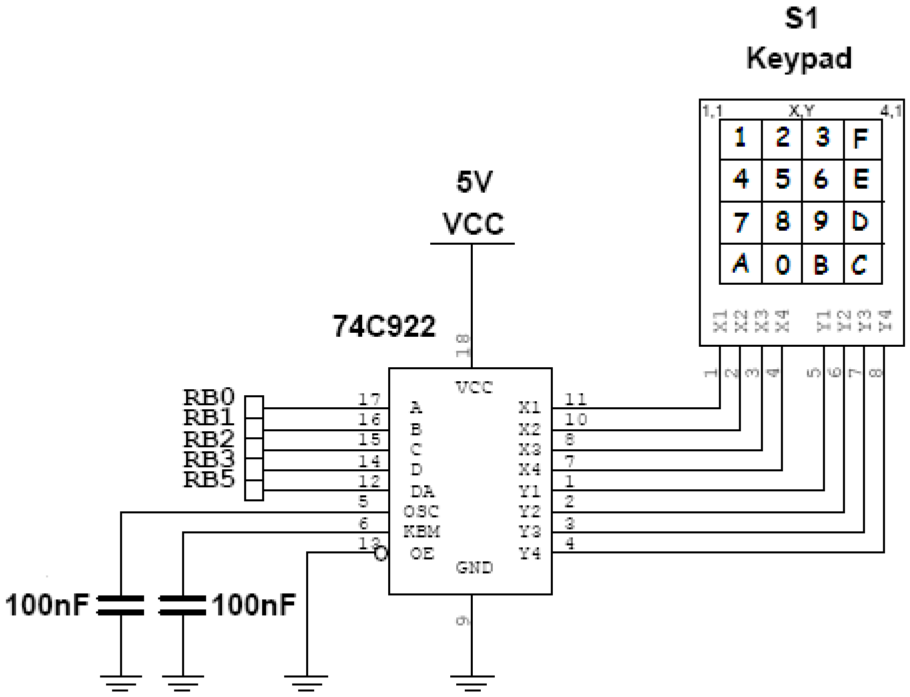
Keypad at Port B

□ In the second part of this experiment, you will be reading from a 4x4 keypad (with encoder) connected to Port B (pins 0, 1, 2, 3 and 5).

□ Examine the connection below. See how the 16 keys are labelled. The columns are numbered X1, X2, X3, X4 (from left to right) while the rows are numbered Y1, Y2, Y3, Y4 (from top to bottom). So, the key 2 is X2, Y1 while the key B is X3, Y4.

Q4: Which key corresponds to X2, Y3?

Your answer: ____



□ These 8 signals (X's and Y's) from the keypad are connected to a **keypad encoder 74C922** which has the **truth table** below. As a result of "encoding", 8 bits become 5 bits.

Keypad Encoder truth table

	Keys															
	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4	X1	X2	X3	X4
	Y1	Y1	Y1	Y1	Y2	Y2	Y2	Y2	Y3	Y3	Y3	Y3	Y4	Y4	Y4	Y4
D	0	**0**	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C	0	**0**	0	0	1	1	1	1	0	0	0	0	1	1	1	1
B	0	**0**	1	1	0	0	1	1	0	0	1	1	0	0	1	1
A	0	**1**	0	1	0	1	0	1	0	1	0	1	0	1	0	1
DA	1	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1

□ D (msb), C, B, A identify the key pressed. For instance, if key '2' is pressed, X2, Y1 cause DCBA = 0001. *[Note: This does not tell the key pressed is 2, as binary 0001 is not exactly decimal 2. Further interpretation is needed - see C code below.]* The DA (**data available**) signal will be set to logic '1', whenever a key is pressed.

Q5: What happen if key '6' is pressed?

Your answer: key '6' is X__ & Y__. So, DCBA = _______, DA = _______

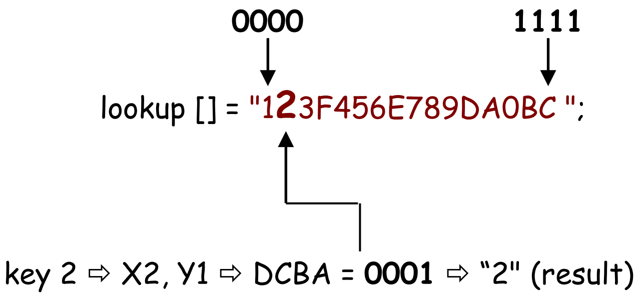
□ The function to read & interpret the key is this:

```
#define KEY_DA   PORTBbits.RB5  // 74922 DA output
#define KEY_PORT  PORTB  // RB3 to RB0 has keypad data

char getkey (void)
{  char keycode;
   const unsigned char lookup[] = "123F456E789DA0BC ";
   while (KEY_DA==0);  // wait for key to be pressed
   keycode=KEY_PORT & 0x0F;  // read from encoder at portB,
                             // mask off upper 4 bits

   while (KEY_DA==1);  // wait for key to be released
   return( lookup [keycode] ); // look up table to find
                               // key pressed
}
```

□ The function waits for DA to become 1 i.e. a key pressed. Then it reads from Port B and mask off the top 4 bits, i.e. only RB3 to RB0 (connected to the signals D, C, B and A) are retained. After that, it waits for the key to be released. Finally, it returns the key pressed by looking up the look-up-table.

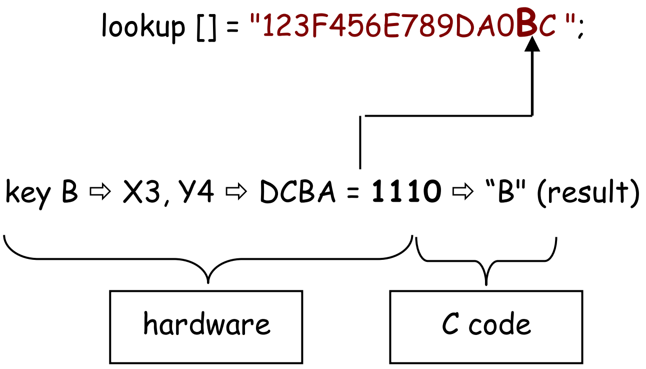


0000 **1111**

lookup [] = "1**2**3F456E789DA0BC ";

key 2 ⇨ X2, Y1 ⇨ DCBA = **0001** ⇨ “2" (result)

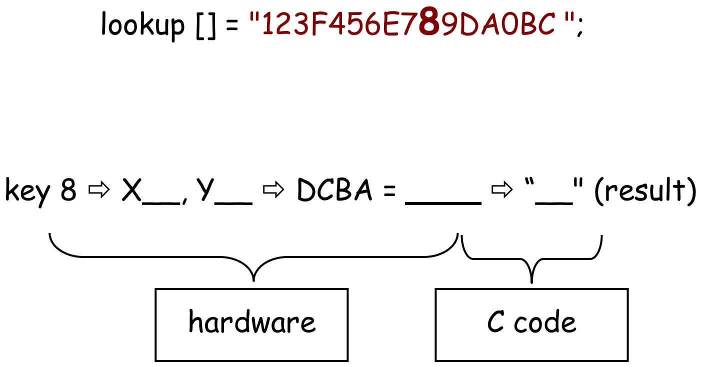
□ Likewise, if key B has been pressed, you get this



lookup [] = "123F456E789DA0**B**C ";

key B ⇨ X3, Y4 ⇨ DCBA = **1110** ⇨ “B" (result)

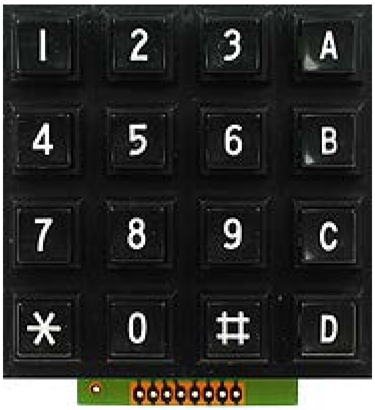
Q6: Try key 8…



lookup [] = "123F456E7**8**9DA0BC ";

key 8 ⇨ X__, Y__ ⇨ DCBA = _____ ⇨ “__" (result)

Q7: Assuming the hardware connections are unchanged, but the 4x4 keypad has been labelled differently, as follows:



What changes to the look up table is necessary for correct interpretation?

Your answer: lookup [] = "1___4____7____*____ ";

Q8. You will come across the following code in the last part of the experiment.

```
lcd_write_cmd(0xC0); // Move cursor to line 2 position 1

for ( i = 0; i < 20; i++) // for 20 number
{
  key=getkey(); // use "getkey" function to read/interpret key pressed
  lcd_write_data(key); // display on LCD
}
```

Describe what happens when the code is executed:

Your answer: ____

Activites:

Before you begin, ensure that the Micro-controller Board is connected to the LCD / Keypad Board.

Displaying an alphanumeric string on LCD

1. Launch the *MPLABX IDE* and create a new project called *Lab4*.

2. Add the file *LCD2Lines.c* to the *Lab4* project *Source File* folder. Make sure *Copy* is ticked before you click *Select*. If you have forgotten the steps, you will need to refer to the previous lab sheet.

   Note that the program uses the functions *lcd_init (), lcd_write_cmd (), lcd_write_data ()* from *lcd_utilities.c* and contains *#include "lcd.h"*. The files *lcd.h* and *lcd_utilities.c* need to be added to the Project.

Display msg on LCD ⇨

3. Study the code (the main function) and describe what this program will do:

   ____

4. Build, download and execute the program. Observe the result and see if it is as expected.

Changing the msg on LCD ⇨

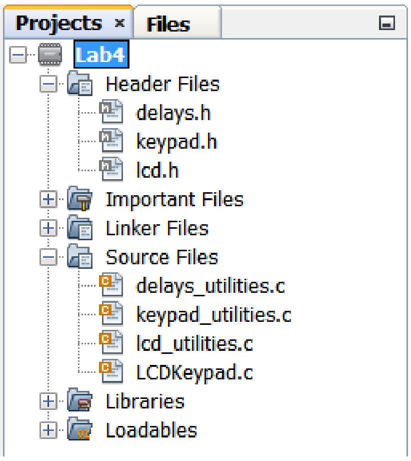
5. Modify the code to show the following on the LCD. Build, download and execute the program to verify your coding.

```
JOHN
9123456
```

Reading inputs from keypad and displaying them on LCD

Keypad entries on LCD ⇨

6. Replace *LCD2Lines.c* with *LCDKeypad.c*. The files *keypad.h* and *keypad_utilities.c* (which contains the *getkey ()* function) need to be added to the Project. Likewise, the files *delays.h* and *delays_utilities.c* need to be added.



Projects × Files

Lab4
- Header Files
  - delays.h
  - keypad.h
  - lcd.h
- Important Files
- Linker Files
- Source Files
  - delays_utilities.c
  - keypad_utilities.c
  - lcd_utilities.c
  - LCDKeypad.c
- Libraries
- Loadables

7. Study the code and describe what this program will do:

____

8. Build, download and execute the program. Observe the result and see if it is as expected.

4-key PIN no. on LCD

9. Modify the code to accept a 4-key PIN number (-- see Hint below). Build, download and execute the program to verify your coding.

<u>Hint</u>:

```
unsigned char P1, P2, P3, P4;  // variables to store a copy of the PIN number
                               // entered. Put this BEFORE any executable code in main

// put the following after the code to move cursor to line 2 position 1,
// replacing the 2nd for loop
key = getkey(); // get the first key
P1 = key; // save first key in P1
lcd_write_data(key); // display on LCD

key = getkey(); // get the second key
P2 = key; // save second key in P2
lcd_write_data(key); // display on LCD
…

while(1); // add this to prevent program from restarting after 4th key
```

Hiding the PIN number

10. Further modify the code so that it will not display the actual PIN number entered. Instead, * will be shown after each key.

    Enter PIN number: * * * _

<u>Optional:</u>

After 4 keys have been entered, the program can show the message:
"Processing……"

<u>Hint</u>: You need to have a char array: Message2 [] = " Processing….." and a loop to display this message.

11. Build, download and execute the program to verify your coding. Debug until the program can work.

<u>Password protected" access</u>

Password protected door

12. Replace *LCDKeypad.c* with *LCDKeypadPwd.c*.

13. This program will accept a 4-key password (or "PIN number"). If the correct password is entered, the LCD will display "*OPEN*". Otherwise, the LCD will display "*WRONG*"

14. What do you think is the password?

    Your answer: ___________

15. Build, download and execute the program. Observe the result and see if it is as expected.

```
// LCD2Lines.c
// Program to test LCD.
// The LCD display with two lines, 24 characters each.
// There are three control lines (RD4:RD6) and four data lines(RD3:RD0).
// RD6 - RS=0 Data represents Command, RS=1 Data represents Character
// RD5 - RW=0 Writing into the LCD module
// RD4 - E =1 Data is latched into LCD module during low to hight
transition

#include <xc.h>
#include "lcd.h"  // Include file is located in the project directory

void main(void)
{
      lcd_init();                   // Initialise LCD module

      while(1)
      {
      lcd_write_cmd(0x80);          // Move cursor to line 1 position 1
      lcd_write_data(0x41);         // write "A" to LCD
      lcd_write_data(0x42);         // write "B" to LCD
      lcd_write_data(0x43);         // write "C" to LCD

      lcd_write_cmd(0xC0);          // Move cursor to line 2 position 1
      lcd_write_data(0x31);         // write "1" to LCD
      lcd_write_data(0x32);         // write "2" to LCD
      lcd_write_data(0x33);         // write "3" to LCD

      while(1);                     //stop here for now

      }
}
```

```
// LCDKeypad.c
// Program to test LCD and keypad.
// For project using USB interface with Bootloader

#include "lcd.h"
#include <xc.h>
#include "keypad.h"
#include "delays.h"
unsigned char key,outchar;
char Message1 [ ] = "Enter PIN number :  ";        // Defining a 20 char string

// ---- Main Program --------------------------------------------------------------
void main(void)
{
        int i;
        lcd_init();                                 // Initialise LCD module

        while(1)
        {
                lcd_write_cmd(0x80);                // Move cursor to line 1 position 1
                for (i = 0; i < 20; i++)            //for 20 char LCD module
                {
                        outchar = Message1[i];
                        lcd_write_data(outchar); // write character data to LCD
                }


                lcd_write_cmd(0xC0);                // Move cursor to line 2 position 1
                for (i = 0; i < 20; i++)            //for 20 number
                {
                        key=getkey();               // waits and get ascii key number when pressed
                        lcd_write_data(key);        //display on LCD

                }

                delay_ms(1000);                     // wait 1 second
                lcd_write_cmd(0x01);                // 00000001 Clear Display instruction
        }
}
```

```
// LCDKeypad.c
// Program to test LCD and keypad.
// For project using USB interface with Bootloader

#include <xc.h>
#include "lcd.h"
#include "delays.h"
#include "keypad.h"

unsigned char key, outchar;
unsigned char p1, p2, p3, p4;
char Message1 [ ] = "Enter PIN number :  "; // Defining a 20 char string



void main(void) {
    int i;
    lcd_init(); // Initialise LCD module
    while (1) {

        lcd_write_cmd(0x80); // Move cursor to line 1 position 1
        for (i = 0; i < 20; i++) //for 20 char LCD module
        {
            outchar = Message1[i];
            lcd_write_data(outchar); // write character data to LCD
        }

        lcd_write_cmd(0xC0); // Move cursor to line 2 position 1

        key = getkey(); // waits and get an ascii key number when pressed
        p1 = key;
        lcd_write_data(key); //display on LCD

        key = getkey(); // waits and get an ascii key number when pressed
        p2 = key;
        lcd_write_data(key); //display on LCD

        key = getkey(); // waits and get an ascii key number when pressed
        p3 = key;
        lcd_write_data(key); //display on LCD

        key = getkey(); // waits and get an ascii key number when pressed
        p4 = key;
        lcd_write_data(key); //display on LCD

        if (p1 == '4' && p2 == '5' && p3 == '5' && p4 == '0')
        {
            lcd_write_data(0x20);
            lcd_write_data('O');
            lcd_write_data('P');
            lcd_write_data('E');
            lcd_write_data('N');
            lcd_write_data(0x20);
        }
        else
        {
            lcd_write_data(0x20);
            lcd_write_data('W');
            lcd_write_data('R');
            lcd_write_data('O');
            lcd_write_data('N');
            lcd_write_data('G');
        }

    }
}
```

```
/*  file : lcd.h
 *     LCD interface header file
 *     See lcd.c for more info
 */

 /* intialize the LCD - call before anything else */
extern void lcd_init(void);

/* write a byte to the LCD in 4 bit mode */
extern void lcd_write_cmd(unsigned char cmd);

//extern void lcd_write(unsigned char i);
extern void lcd_write_data(char data);
```

```
/*
 * File:   lcd utilities.c
 *
 * Created on 13 January, 2016, 10:28 AM
 */

//#include "LCD.H"  // Include file is located in the project directory

#include <xc.h>
#define _XTAL_FREQ 48000000
#define LCD_RS PORTDbits.RD6    //  Register Select on LCD
#define LCD_EN PORTDbits.RD4    //  Enable on LCD controller
#define LCD_WR PORTDbits.RD5    //  Write on LCD controller
void lcd_strobe(void);

//--- Function for writing a command byte to the LCD in 4 bit mode -------------

void lcd_write_cmd(unsigned char cmd)
{
    unsigned char temp2;
    LCD_RS = 0;                    // Select LCD for command mode
    __delay_ms(4);                 // 40us delay for LCD to settle down

    temp2 = cmd;
    temp2 = temp2 >> 4;    // Output upper 4 bits, by shifting out lower 4 bits
    PORTD = temp2 & 0x0F;          // Output to PORTD which is connected to LCD

    __delay_ms(8);                 // 10ms - Delay at least 1 ms before strobing
    lcd_strobe();
    __delay_ms(8);                 // 10ms - Delay at least 1 ms after strobing

    temp2 = cmd;                   // Re-initialise temp2
    PORTD = temp2 & 0x0F;          // Mask out upper 4 bits

    __delay_ms(8);                 // 10ms - Delay at least 1 ms before strobing
    lcd_strobe();
    __delay_ms(8);                 // 10ms - Delay at least 1 ms before strobing

}
```

```
//---- Function to write a character data to the LCD ---------------------------

void lcd_write_data(char data)
{
    char temp1;

    LCD_RS = 1;                  // Select LCD for data mode
    __delay_ms(4);               // 40us delay for LCD to settle down

    temp1 = data;
    temp1 = temp1 >> 4;
    PORTD = temp1 & 0x0F;

    __delay_ms(8);               //_-_ strobe data in
    lcd_strobe();
    __delay_ms(8);

    temp1 = data;
    PORTD = temp1 & 0x0F;

    __delay_ms(10);              //_-_ strobe data in
    lcd_strobe();
    __delay_ms(10);
}


//-- Function to generate the strobe signal for command and character----------

void lcd_strobe(void)            // Generate the E pulse
{
    LCD_EN = 1;                  // E = 1
    __delay_ms(8);               // 10ms delay for LCD_EN to settle
    LCD_EN = 0;                  // E = 0
    __delay_ms(8);               // 10ms delay for LCD_EN to settle
}
```

```
//---- Function to initialise LCD module ----------------------------------------
void lcd_init(void)
{
    int i;
    TRISD = 0x00;
    PORTD = 0x00;               // PORTD is connected to LCD data pin
    LCD_EN = 0;
    LCD_RS = 0;                 // Select LCD for command mode
    LCD_WR = 0;                 // Select LCD for write mode

    for(i=0;i<100;i++)
    __delay_ms(10);             // Delay a total of 1 s for LCD module to
                    // finish its own internal initialisation

    /* The datasheets warn that LCD module may fail to initialise properly when
       power is first applied. This is particularly likely if the Vdd
       supply does not rise to its correct operating voltage quickly enough.

       It is recommended that after power is applied, a command sequence of
       3 bytes of 30h be sent to the module. This will ensure that the module is
       in 8-bit mode and is properly initialised. Following this, the LCD module
       can be switched to 4-bit mode.
    */

    lcd_write_cmd(0x33);
    lcd_write_cmd(0x32);

    lcd_write_cmd(0x28);   // 001010xx – Function Set instruction
                           // DL=0 :4-bit interface,N=1 :2 lines,F=0 :5x7 dots

    lcd_write_cmd(0x0E);   // 00001110 – Display On/Off Control instruction
                           // D=1 :Display on,C=1 :Cursor on,B=0 :Cursor Blink on

    lcd_write_cmd(0x06);   // 00000110 – Entry Mode Set instruction
                           // I/D=1 :Increment Cursor position
                           // S=0 : No display shift

    lcd_write_cmd(0x01);   // 00000001 Clear Display instruction

    __delay_ms(10);        // 10 ms delay

}
```

```
// file : keypad.h

extern char getkey(void); // waits for a keypress and returns the ascii code


/*
 * File:   keypad utilities.c
 *
 * Created on 13 January, 2016, 10:46 AM
 */


#include <xc.h>


#define KEY_DA PORTBbits.RB5   //  74922 DA output
#define KEY_PORT PORTB           //     RB3 to RB0 has keypad data


//----- Function to obtained wait for key press and returns its ASCII value

char getkey(void)
{      char keycode;
    const unsigned char lookup[] = "123F456E789DA0BC ";

    while (KEY_DA==0);           //wait for key to be pressed
    keycode=KEY_PORT & 0x0F;     //read from encoder at portB,mask upper 4 bits

    while (KEY_DA==1);           //wait for key to be released
    return(lookup[keycode]);     //convert keycode to its ascii value for LCD
}
```