Experiment #7 (HW#2)
Embedded Systems: Keypads and DC Motors

1. Prerequisites

ENCS 538, C programming language, PICC compiler, Microchip PIC16F877A datasheet.

2. Objectives

- Getting familiar to keypads, how they work and how to program and implement an application using them.
- Getting familiar to DC drivers and motors.
- Getting introduced to embedded systems.

3. Background

3.1 Real-time Embedded Systems

Real-time embedded systems are fully-functional devices that can achieve mostly any program that can be written to serving some real-time purposes. This type of systems contain the most common peripherals that can be connected to a microcontroller.

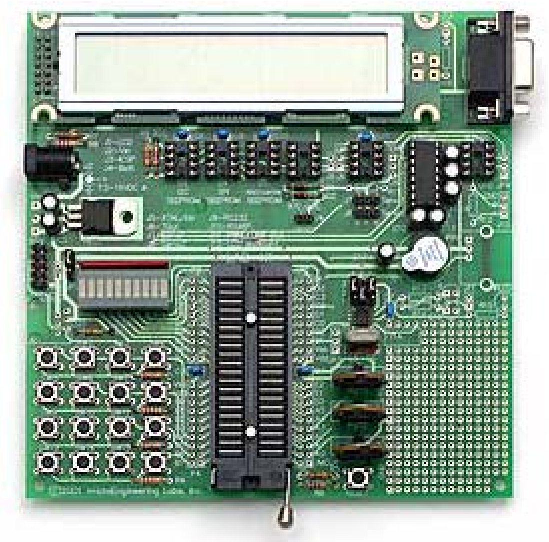

Figure 1 : LAB-X1 Experimenter Board

A good advantage to embedded systems is that they provide a built-it connected circuit where we don’t need to worry about hardware implementation; all what concerns us is to understand the applied connections between the MCU and the peripheral components and write a program that runs with these connections, download it on the MCU and run it within the system. On the other hand, a disadvantage to embedded systems is that their connections are fixed and cannot be changed, causing a constraint when writing the MCU program e.g. if we want to change the functionality of the LCD to be based on port B instead of D and E.

Embedded systems are not always separate devices. Most often they are physically built-in to the devices they control. The software written for embedded systems is often called firmware, and is stored in read-only memory or Flash memory chips rather than a disk drive. It often runs with limited computer hardware resources: small or no keyboard, screen, and little memory.

In this experiment, LAB-X1 Experimenter Board will be used as an embedded system to let us handle input/output units such as an LCD screens and keypads. Figure 1 shows the LAB-X1 board. It contains a keypad, LCD screen, serial connection, buzzer, LEDs and more. For more information about the LAB-X1 board, you can download its datasheet from the internet.

3.2 Keypad Interfacing

A keypad is a set of buttons that are arranged in a block "pad". The idea behind a keypad is assigning each button some certain value so that when it gets pressed, the MCU will be able to understand its meaning and apply a certain functionality according to what's written in the MCU program.

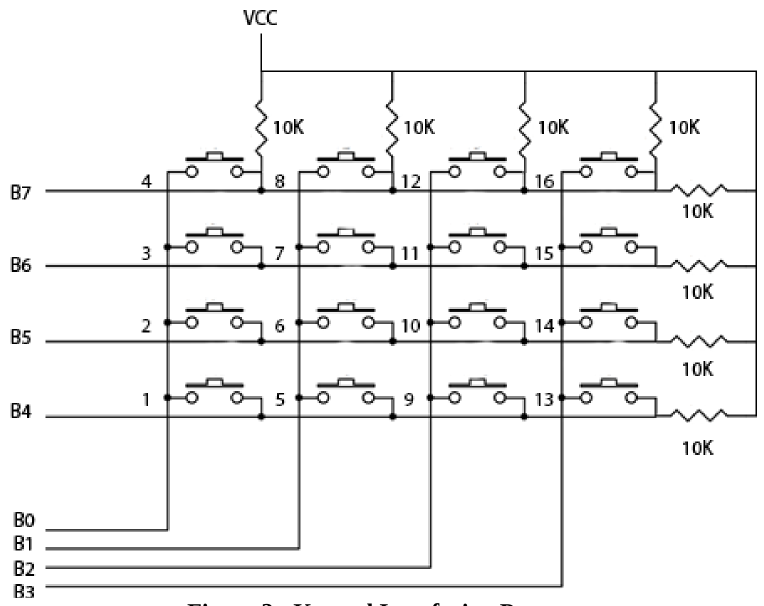

Figure 2 : Keypad Interfacing Ports

<u>The keypad functions according to the following algorithm:</u>

1. Define integer variables: **key** (a global variable to hold the pressed key value); **flag** (for key pressed breaking condition); **col** (for column loop indexing); **row** (for row loop indexing); **colVar** (to update direction registers for the columns); **rowVar** (to calculate the shift when converting from coordinate system to index system for the pressed key)
2. Assign the most significant nibble of port B as row inputs
3. Assign the least significant nibble of port B as column outputs
4. while the inputs on the rows are still not ones (connected to external high voltage source VCC, in other words: not pressed):
   a. assign the value zero to all port B bits
   b. assign the most significant nibble direction registers as inputs for the rows

   c. assign the least significant nibble direction registers as outputs for the columns

Now that we checked that none of the keys are pressed, we can wait a key press and save it

5. while true (forever waiting loop):
   a. if **flag** of key pressed is not set, then continue
   b. else, break because of a key press
   c. use **col** to loop on all the columns; set **colVar** to be within {1,2,4,8} and use it to set the one reached to zero through an XOR operation.
   d. if the rows nibble is not ones, then the row with zero value is found to be the row index and this is where we set the **flag** to 1 so that the forever loop breaks with the **row**, **col** and **colVar** of the pressed key are saved
   e. else, loop forever setting a column to zero on each iteration

Now that we obtained the pressed key in 2-dimention row/col coordinate system, we need to convert it into an indexed system where they has an integer value in the range [1-16]

6. set **rowVar** to the inverse of **row** by XORing it with F to get values in the set {1,2,4,8}
7. convert the values in the set {1,2,4,8} by mapping them to the new set {1,2,3,4}, this will be the shift variable
8. multiply **col** by 4 and add the **rowVar** to it, save it into **key**.

3.3 DC Motor Interfacing

The DC motor has two basic parts: the rotating part that is called the armature, and the stationary part that includes coils of wire called the field coils. The stationary part is also called the stator.

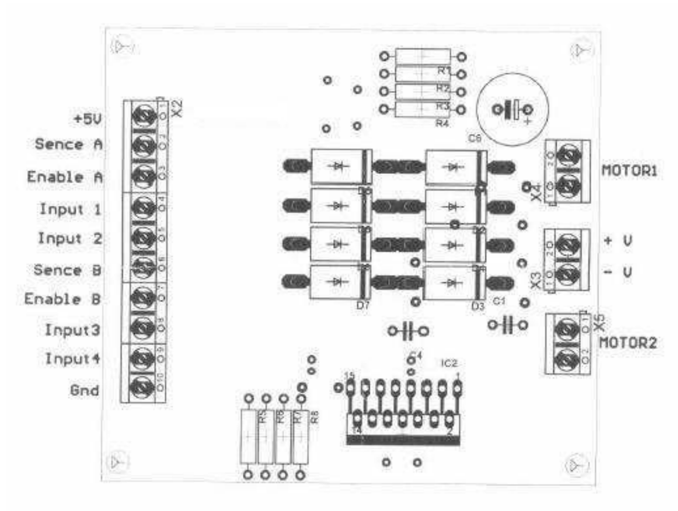

Figure 3: L298N DC motor driver

The armature and field in a DC motor can be wired in three different ways to provide varying amounts of torque or different types of speed control. The armature and field windings are designed slightly differently for different types of DC motors. The three basic types of DC motors are the series motor, the shunt motor, and the compound motor. The series motor is designed to move large loads with high starting torque in applications such as a crane motor. The shunt motor is designed slightly differently, since it is made for applications such as pumping fluids, where constant-speed characteristics are important. The compound motor is designed with some of the series motor's characteristics and some of the shunt motor's characteristics. This allows the compound motor to be used in applications where high starting torque and controlled operating speed are both required.

The DC motor we have is a shunt motor and requires a voltage between 12 VDC to 24 VDC. To drive the DC motor, we will use a driver that has been built specifically for that purpose (see Figure 3). The inputs to the driver are on the left side of the Figure while the outputs are on the right side.

As seen from the Figure, the driver can drive up to 2 DC motors (Motor A and Motor B). To rotate motor A in a clockwise direction, Input1 should be high while Input2 should be low. To rotate motor A in a counter-clockwise direction, Input2 should be high while Input1 should be low. The same argument goes to inputs Input3 and Input4 of motor B. You need to make sure that the 2 inputs are never high at the same time in order to have the motor rotate.

The inputs Enable A and Enable B should be high in order to have the motor rotate as required. Otherwise, the shaft of the motor would jam.

4. Procedure

4.1 Keypad Interfacing

1. Create a new project using **MPLAB.**
2. Add the codes: **delay.h, delay.c, lcd.h, lcd.c, key**.c.
3. Compile the project.
4. Create a new schematic using **Protues ISIS**.
5. Add the components: **PIC16F877A**, **CAP**, **CRYSTAL**, **RES**, **POT-HG or POT-LIN**, **LM016L**, and **KEYPAD-SMALLCAL**.
6. Apply the connections in **Figure 4**.
7. Simulate.
8. The **LCD** must display the value of the pressed key.
9. Download the program on the **PIC16f877A.**
10. Connect the **PIC16F877A** to the **LAB-X1 module.**
11. Compare the experimental results to the simulated ones.

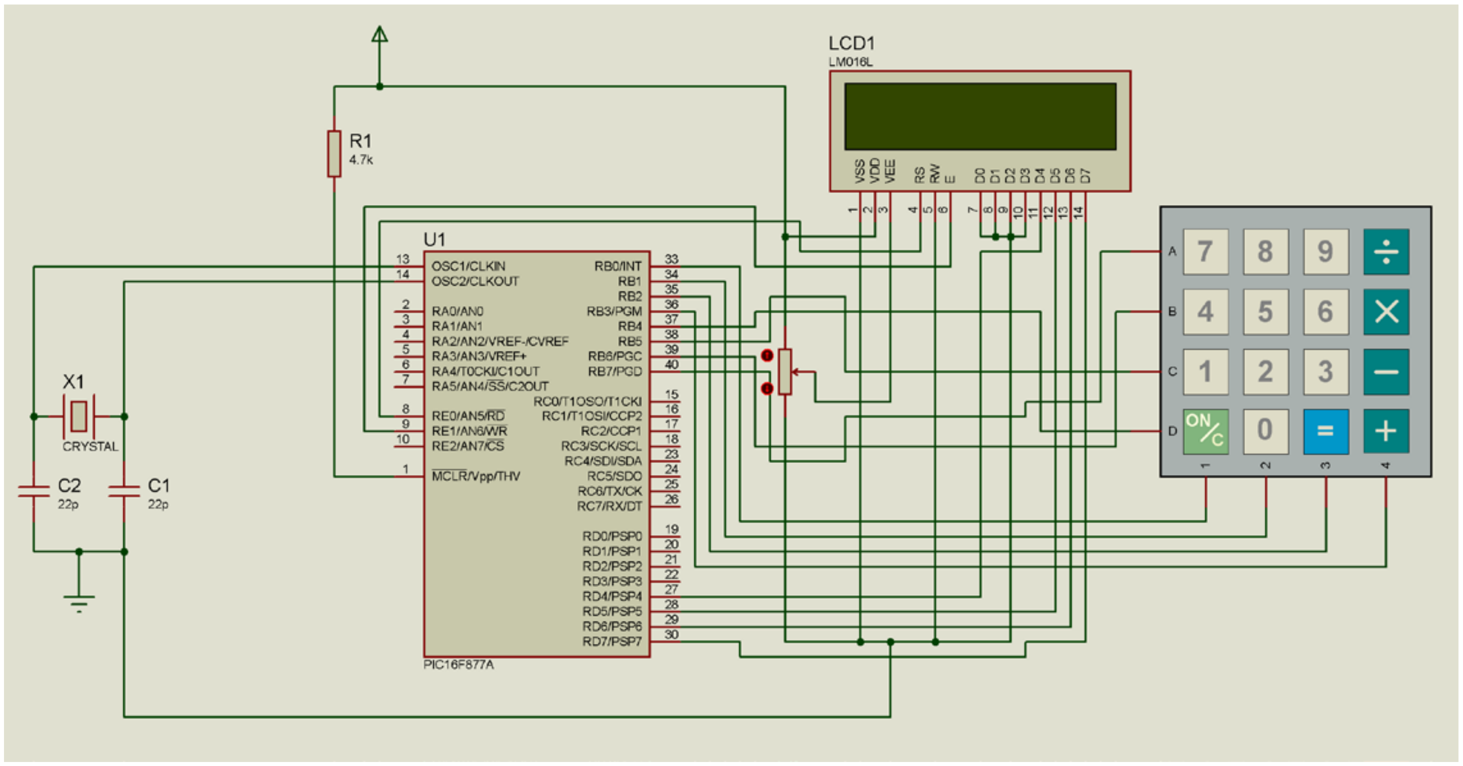

Figure 4: Keypad interfacing

4.2 DC Motor Interfacing

1. Create a new project using **MPLAB.**
2. Add the codes: **delay.h, delay.c, dcmtrl.c** to it.
3. Compile the project.
4. Create a new schematic using **Protues ISIS**.
5. Add the components: **PIC16F877A**, **CAP**, **CRYSTAL**, **RES**, **L298**, **MOTOR**, and **VSOURCE**.
6. Set the **VSOURCE** to **12 Volts**.
7. Apply the connections in **Figure 5**.
8. Simulate.
9. The **Motor** must rotate clockwise for 2 seconds then counterclockwise for another 2 seconds
10. Download the program on the **PIC16f877A**
11. Connect the **PIC16F877A** to the **LAB-X1 module**
12. Connect **pin0** of **PortB7 (pin40)** to **Input1** of the **driver**. Connect also **pin1** of **PortB6 (pin39)** to **Input2** and **pin2** of **PortB5 (pin38)** to Enable **A**.
13. Connect the **5VDC** to the **5V** pin on the **L298**.
14. Connect **12VDC** to the **+V** pin on the **L298**. Connect the **GND** and the **SenceA** pins of the **L298** to the **GND** of the same channel providing the **12VDC**.
15. Compare the experimental results to the simulated ones.

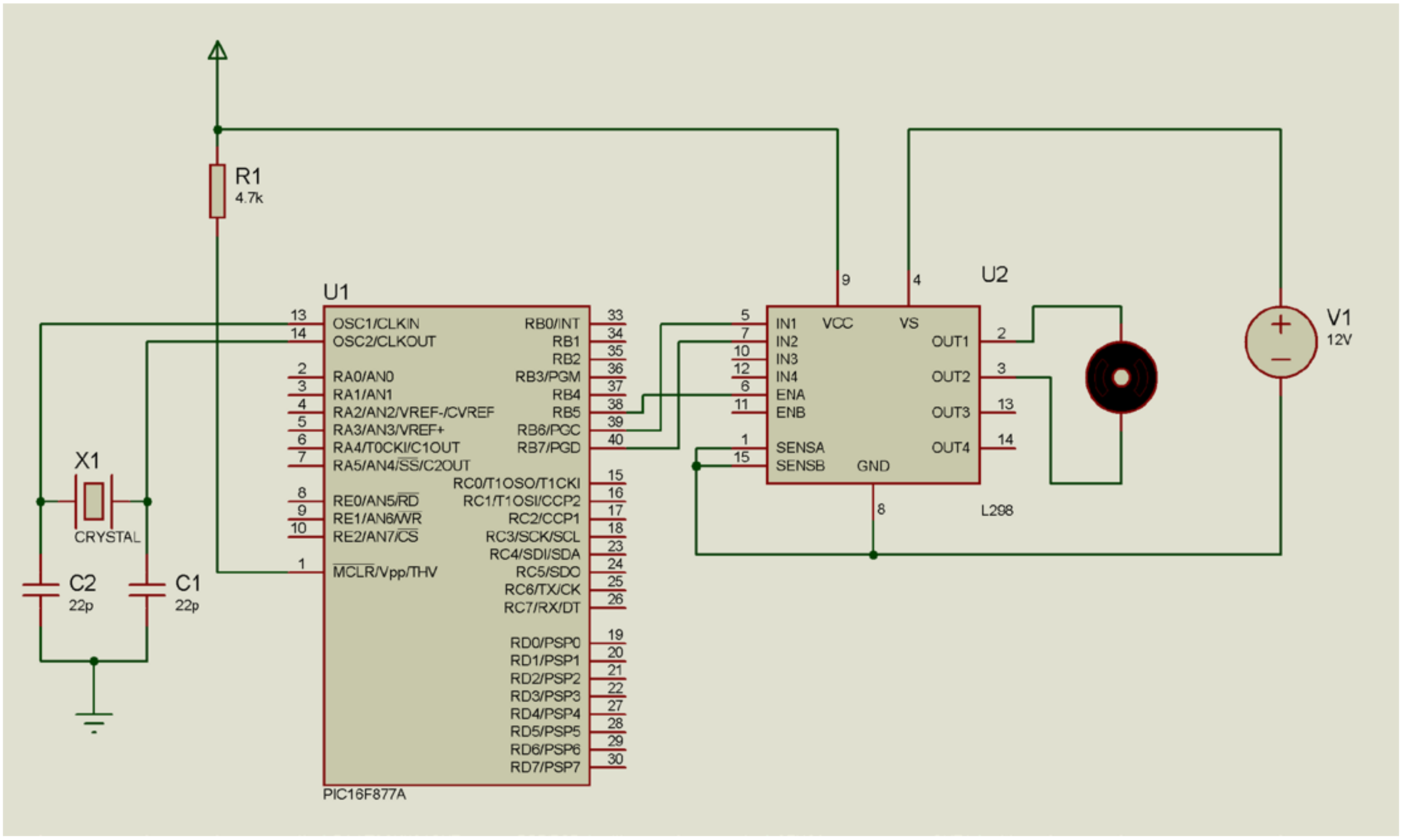

Figure 5: L298N and Motor interfacing

5. Experiment Codes

Delay.h

```
/*
 *    Delay functions for HI-TECH C on the PIC
 *
 *    Functions available:
 *          DelayUs(x)  Delay specified number of microseconds
 *          DelayMs(x)  Delay specified number of milliseconds
 *
 *    Note that there are range limits: x must not exceed 255 - for xtal
 *    frequencies > 12MHz the range for DelayUs is even smaller.
 *    To use DelayUs it is only necessary to include this file; to use
 *    DelayMs you must include delay.c in your project.
 *
 */

/*    Set the crystal frequency in the CPP predefined symbols list in
      HPDPIC, or on the PICC commmand line, e.g.
      picc -DXTAL_FREQ=4MHZ

      or
      picc -DXTAL_FREQ=100KHZ
      Note that this is the crystal frequency, the CPU clock is
      divided by 4.

 *    MAKE SURE this code is compiled with full optimization!!!

 */

#ifndef     XTAL_FREQ
#define     XTAL_FREQ   4MHZ        /* Crystal frequency in MHz */
#endif

#define     MHZ   *1000L                  /* number of kHz in a MHz */
#define     KHZ   *1                /* number of kHz in a kHz */

#if   XTAL_FREQ >= 12MHZ

#define     DelayUs(x)  { unsigned char _dcnt; \
                    _dcnt = (x)*((XTAL_FREQ)/(12MHZ)); \
                    while(--_dcnt != 0) \
                          continue; }
#else

#define     DelayUs(x)  { unsigned char _dcnt; \
                    _dcnt = (x)/((12MHZ)/(XTAL_FREQ))|1; \
                    while(--_dcnt != 0) \
                          continue; }
#endif

extern void DelayMs(unsigned char);
```

Delay.c

```
/*
 *    Delay functions
 *    See delay.h for details
 *
 *    Make sure this code is compiled with full optimization!!!
 */

#include "delay.h"

void DelayMs(unsigned char cnt)
{
#if   XTAL_FREQ <= 2MHZ
      do {
            DelayUs(996);
      } while(--cnt);
#endif

#if    XTAL_FREQ > 2MHZ
      unsigned char     i;
      do {
            i = 4;
            do {
                  DelayUs(250);
            } while(--i);
      } while(--cnt);
#endif
}
```

Lcd.h

```
extern void lcd_write(unsigned char);
extern void lcd_clear(void);
extern void lcd_puts(const char * s);
extern void lcd_goto(unsigned char pos);
extern void lcd_init(void);
extern void lcd_putch(char);
#define lcd_cursor(x) lcd_write(((x)&0x7F)|0x80)
```

Lcd.c

```
/*
*     the list of included files contains:
*     pic.h since we're going to use it with our pic microcontroller, duah
*     ldc.h which contains all the prototypes of the functions used for the
lcd
*     delay.h which contains the prototype of the melli-second delay and the
implementation
*     of the micro-second delay (which is used in the melli-second
implementation, so it has to be included)
*     delay.c contains the implementation of the delay_ms function, so it has
to be here
*/

#include <pic.h>
#include "lcd.h"
#include "delay.h"
#define LCD_STROBE ((RE1 = 1),(RE1=0))    //* The E bit on the lcd, where it
tells the lcd that we're writing data to it when it's set to 1*/
/*
*     the following write functions take a character inpute or 8 bits where
it takes the higher 4 bits and passes their values to the D port of the pic
*     then it shifts the character by 4 bits to the left in order to take the
values of the lower 4 bits and put them on the used D port bits. The strobe
*     is used indicate that the LCD is receving data so that the values are
guaranteed to be passed in order and without interference
*
*     As for the RS bit which is connected to RE0. It is used to tell the LCD
to accept the character as a command when it's set to 0, or as a character to
*     be displayed on the screen when it's set to 1
*/
void lcd_write(unsigned char c)
{
PORTD = (PORTD & 0x0F) | (c);
LCD_STROBE;
PORTD = (PORTD & 0x0F) | (c << 4);
LCD_STROBE;
DelayUs(40);
}
void lcd_clear(void)
{
RE0 = 0;
lcd_write(0x1);
DelayMs(2);
}
void lcd_puts(const char * s)
{
RE0 = 1;
while(*s)
lcd_write(*s++);
}
void lcd_putch(char c)
{
RE0 = 1;
PORTD = (PORTD & 0x0F) | (c);
LCD_STROBE;
PORTD = (PORTD & 0x0F) | (c << 4);
LCD_STROBE;
DelayUs(40);
}
void lcd_goto(unsigned char pos)
{
RE0 = 0;
lcd_write(0x80+pos);
}
void lcd_init(void)
{
RE0 = 0;
DelayMs(15); // power on delay
PORTD = (0x3 << 4);
LCD_STROBE;
DelayMs(5);
LCD_STROBE;
DelayUs(100);
LCD_STROBE;
DelayMs(5);
PORTD = (0x2 << 4);
LCD_STROBE;
DelayUs(40);
lcd_write(0x28); // 4 bit mode, 1/16 duty, 5x8 font
lcd_write(0x08); // display off
lcd_write(0x0F); // display on, blink curson on
lcd_write(0x06); // entry mode
}
```

Key.c

```
#include <pic.h>
#include <stdlib.h>
#include "lcd.h"
#include "delay.h"

__CONFIG(DEBUG_OFF & WDTE_OFF & LVP_OFF & FOSC_HS & BOREN_ON);

extern void getkey();
void putchar(char ch);
int col, row, pressed_key;
char Chkey;

void main(void)
{
      /* Define program variables */
      /* Enable PortB weak pullup resistors */
      nRBPU = 0;
      TRISE = 0;
      TRISC = 0;
      TRISD = 0;
      ADCON1 = 7;
      RE2 = 0;
      DelayMs(100);
      lcd_init();
      lcd_clear();

      /* get a key from the keypad, update its value and display it on the
LCD */
      while(1)
      {
            getkey();
            Chkey = (char) pressed_key;
            Chkey+=48;
            putchar((char)Chkey);

      }

}

void getkey()
{
      char colVar, rowVar;
      int flag=0;
      DelayMs(50);

      /* Wait for all keys up */
      do {
            PORTB = 0;
            TRISB = 0xf0;
      } while ( (PORTB >> 4) != 0xf );

      DelayMs(50);

      /* forever loop that assigns a zero to a column in turns until a key is
pressed */
      while (1)
      {
            if ( flag == 1 )
            break;
            /* Wait for keypress */

            for ( col = 0; col < 4; col++ ) { /* 4 columns in keypad */
                  PORTB = 0; /* Step(1): set all output bits to zero, this
step makes all the columns (outputs) set to zero until step (2) comes to
change 3 out of 4 bits to turn 1 */
                  /* All output pins low */
                  if ( col == 0 )
                        colVar = 1;
                  else if ( col == 1 )
                        colVar = 2;
                  else if ( col == 2 )
                        colVar = 4;
                  else
```

```
                        colVar = 8;

                  /* Use colVar to set one column pin to output */
                  TRISB = colVar ^ 0xff; /* Step(2): this makes the current
column picked to be set to zero, and the rest to be ones, since they changed
to inputs where VCC is connected */
                  row = PORTB >> 4; /* take the most significant nibble value
*/
                  /* if one value in the row nibble was zero, this means a
key was pressed to pass the column value to the row input, and hence, we need
to break the scanning loop */
                  if ( row != 0xf ) {
                        flag = 1;
                        break;
                  } /* end of if statement */
            } /* end of column scanning loop */
      } /* end of while true loop */
      /* Use rowVar to calculate the shift added to column index in order to
change the value of the pressed key from row/column coordinates to a value
from 1 to 16 */
      rowVar = row ^ 0xf; /* XOR to invert the value of the column nibble
within the set {1,2,4,8} */
      /* convert the shift by mapping the set {1,2,4,8} to {1,2,3,4} */
      if ( (rowVar / 8) == 1 )
            rowVar = 4;
      else if ( (rowVar / 4) == 1 )
            rowVar = 3;
      else if ( (rowVar / 2) == 1 )
            rowVar = 2;
      else if ( rowVar == 1 )
            rowVar = 1;
      else
            rowVar = 0;

      /* calculate the key index */
      pressed_key = (col * 4) + rowVar;
}

/* if you want to change values on the keypad to ones from your own choice,
you can implement a new function with the new mappings by changing the answer
to each if-statement */
void putchar(char ch)
{
      if(ch=='1')
      {
            lcd_clear();
      }
      else if(ch=='2')
      {
              lcd_putch('1');
      }
      else if(ch=='3')
      {
              lcd_putch('4');
      }
      else if(ch=='4')
      {
              lcd_putch('7');
      }
```

```
       else if(ch=='5')
       {
                lcd_putch('0');
       }
       else if(ch=='6')
       {
                lcd_putch('2');
       }
       else if(ch=='7')
       {
                lcd_putch('5');
       }
       else if(ch=='8')
       {
                lcd_putch('8');
       }
       else if(ch=='9')
       {
                lcd_putch('=');
       }
       else if(ch==':')
       {
                lcd_putch('3');
       }
       else if(ch==';')
       {
                lcd_putch('6');
       }
       else if(ch=='<')
       {
                lcd_putch('9');
       }
       else if(ch=='=')
       {
                lcd_putch('+');
       }
       else if(ch=='>')
       {
                lcd_putch('-');
       }
       else if(ch=='?')
       {
                lcd_putch('x');
       }
       else if(ch=='@')
       {
                lcd_putch('/');
       }
       else
       lcd_putch(ch);
}
```

Dcmtrl.c

```
#include <pic.h>
#include "lcd.h"
#include "delay.h"
```

```
__CONFIG(DEBUG_OFF & WDTE_OFF & LVP_OFF & FOSC_HS & BOREN_ON);

void pause(int d); /* a function to delay in terms of seconds */

void main(void)
{
      /*    RB5 controls enabling/disabling the motor
            RB6 and RB7 are used to control the direction of rotation */

      TRISB = 0;
      RB5=1; /* Eabling the Motor A */
      while (1) {
            RB7 = 1; /* Rotate Clockwise */
            RB6 = 0;
            pause(2);
            RB7 = 0; /* Rotate Counter-Clockwise */
            RB6 = 1;
            pause(2);
       }
}

void pause(int d){
      int i,j;
    for(i=0;i<4;i++)
            for(j=0;j<d;j++)
                  DelayMs(255);
}
```