Materials Required:

1. PIC16F877A Perf Board
2. 16x2 LCD Display
3. Push Button
4. Connecting Wires
5. Bread Board
6. PicKit 3

What are interrupts and where to use them:

Before getting into **how to program PIC microcontroller interrupts**, let us understand what an Interrupt actually is and where we would need to use them. Further, there are lots of types of interrupts in Microcontroller and PIC16F877A has about 15 of them. Let us not confuse them all into our head for now.

So! what is an interrupt in Microcontrollers?

As we all know microcontrollers are used to perform a set of pre-defined (programmed) activates which triggers the necessary outputs based on the input. But, while your Microcontroller is busy with executing one piece of code there might be an emergency situation where other piece of your code needs immediate attention. This other piece of code that needs immediate attention should be treated as an interrupt.

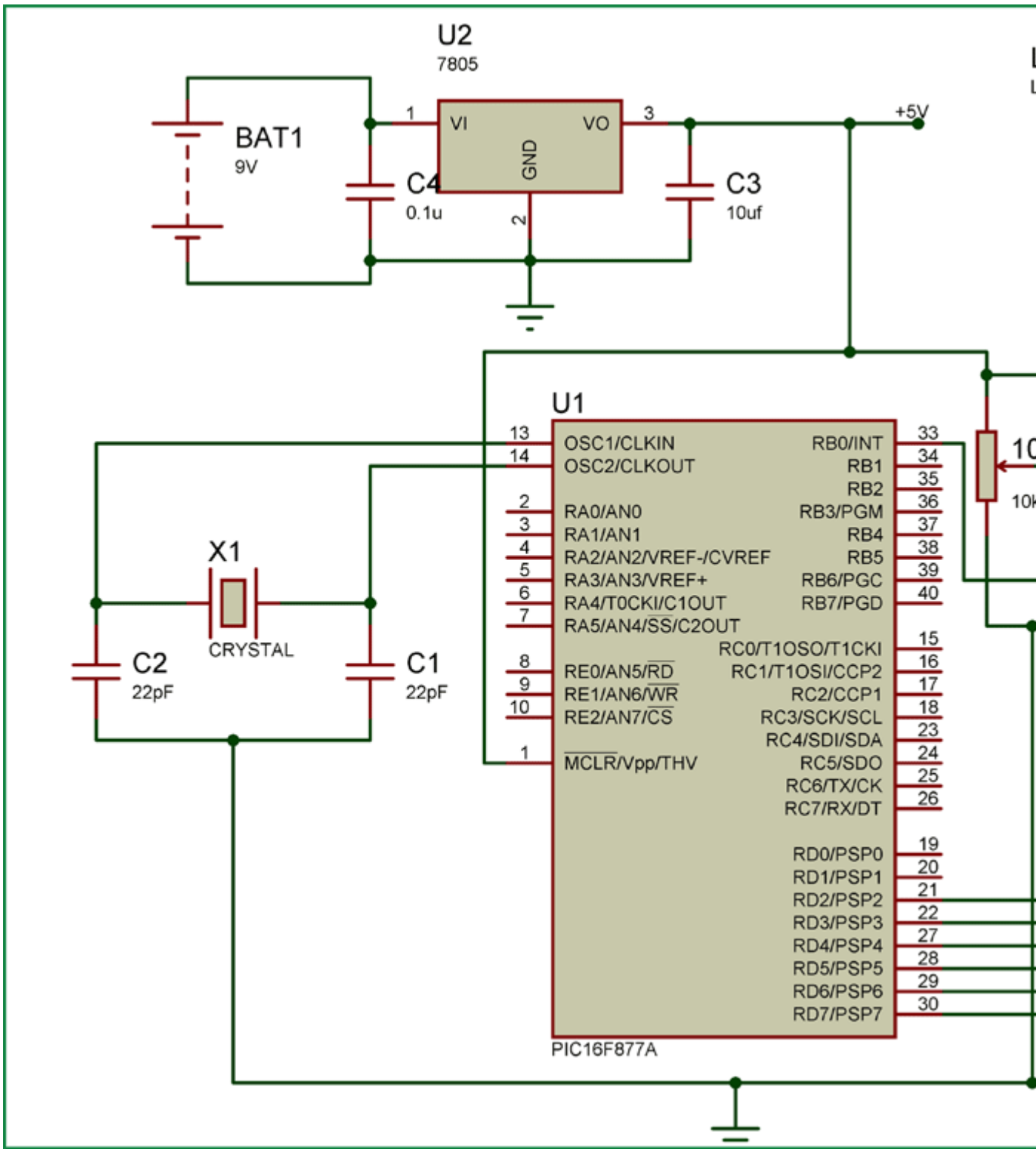
For example: Let us consider that you are playing your favourite game on your mobile and the controller (assumption) inside your phone is busy throwing all the graphics that is needed for you to enjoy the game. But, suddenly your girlfriend calls to your number. Now, the worst thing to happen is your mobiles controller to neglecting your girlfriends call since you are busy playing a game. To prevent this nightmare from happening we use something called interrupts.

These interrupts will always be active listing for some particular actions to happen and when they occur they execute a piece of code and then gets back to the normal function. This piece of code is called the **interrupt service routine (ISR)**. One practical project in which interrupt is mandatory is "Digital Speedometer and Odometer Circuit using PIC Microcontroller"

In Microcontrollers there are two main **types of interrupts**. They are External Interrupt and Internal Interrupt. The internal Interrupts occur inside the Microntroller for performing a task, for example Timer Interrupts, ADC Interrupts etc.. These interrupts are triggered by the software to complete the Timer operation or ADC operation respectively.

The external interrupt is the one that can get triggered by the user. In this program we will learn **how to use an External interrupt** by using a push button to trigger an interrupt. We will use an LCD to display numbers incrementing from 0 to 1000 and when an interrupt is triggered we should notify about it from the Interrupt Service Routine **ISR** and then continue back to incrementing the numbers.

Circuit Diagram and Explanation:



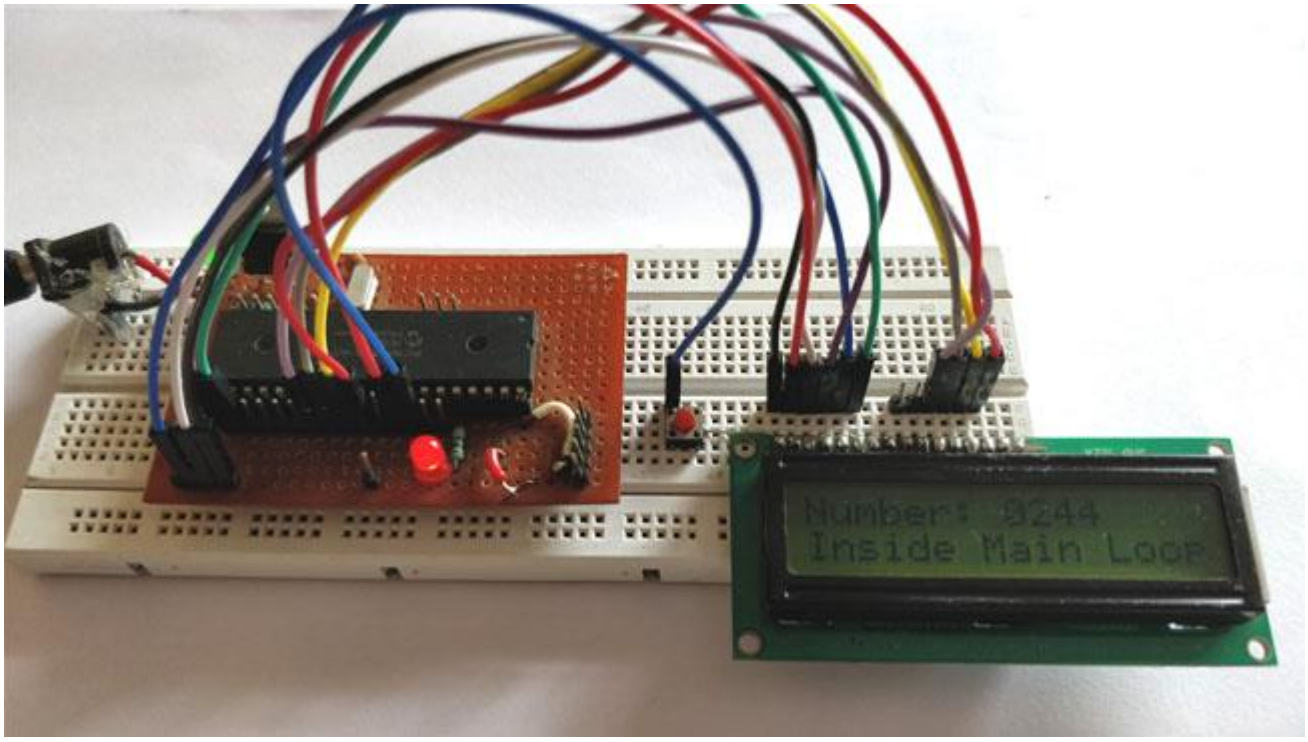
The circuit diagram for using **PIC16F877 interrupts** is given in the above image. You simply have to connect the LCD to the PIC as we did in interfacing LCD tutorial.

Now to connect the interrupt pin, we should look at the datasheet to know **which pin of the PIC is used for External interrupt**. In our case i**n PIC16F877A** the 33rd pin RBO/INT is used for external interrupt. You cannot use any other pin other than this pin. The Pin connection for this circuit diagram is shown in the table below.

S.No:	Pin Number	Pin Name	Connected to
1	21	RD2	RS of LCD
2	22	RD3	E of LCD
3	27	RD4	D4 of LCD
4	28	RD5	D5 of LCD
5	29	RD6	D6 of LCD
6	30	RD7	D7 of LCD
7	33	RBO/INT	Push button

We have enabled internal pull up resistors on PORT B, hence we can directly connect the RB0 pin to ground via a Push button. So whenever this pin gets LOW an interrupt will be triggered.

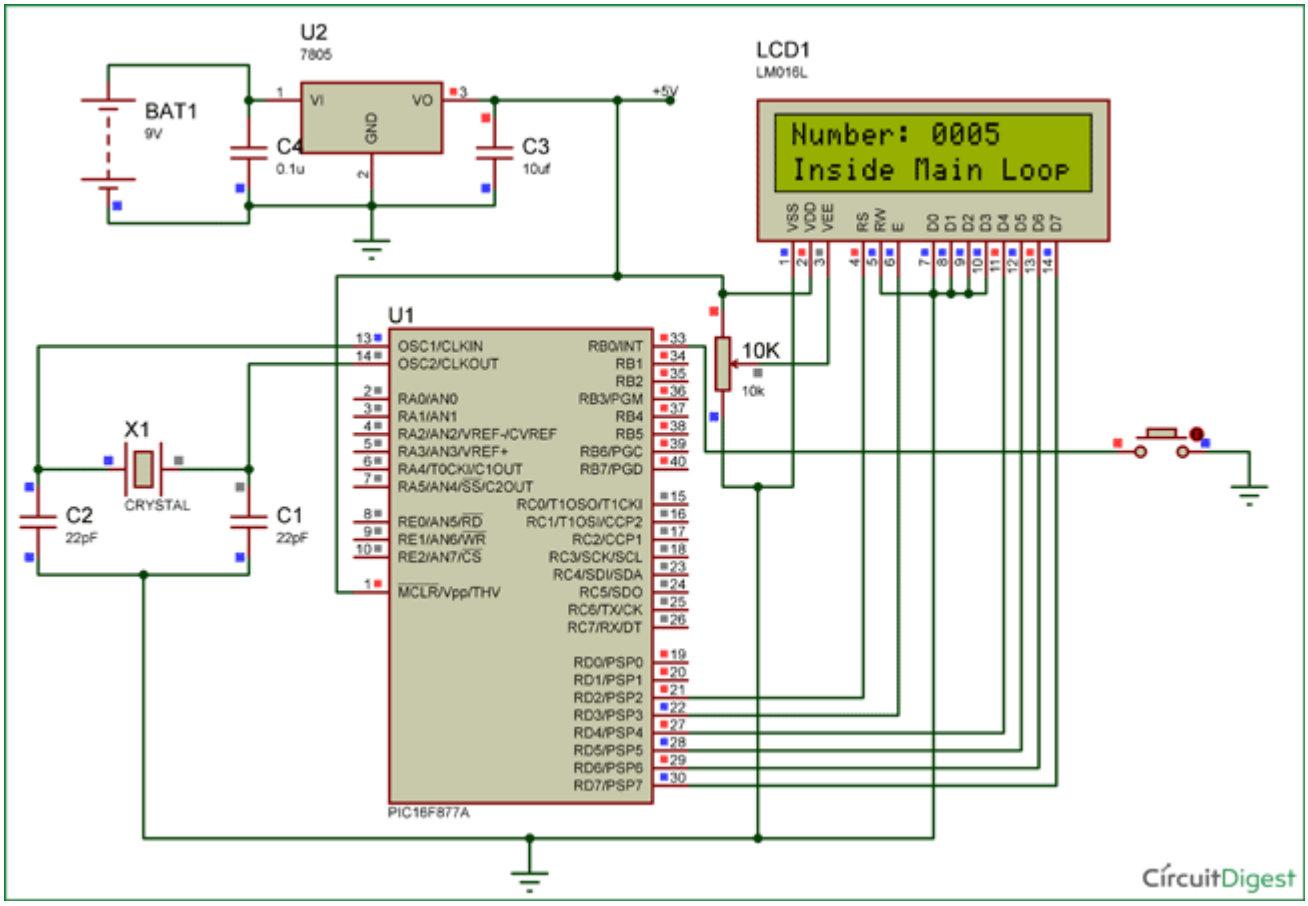
The connections can be made on a Bread board as shown below.



If you have been following our tutorials you should have got familiar with this Perf Board that I have used here. If not, you need not think much about it just simply follow the circuit diagram and you will get things working.

Simulation of Interrupts in PIC Microcontroller:

The Simulation for this project is made using Proteus.



When you simulate the project you should see a sequence of numbers being incremented on the LCD display. This happens inside the main loop and whenever the push button is pressed the LCD should display that it has entered into ISR. You can make your modifications in the code and try testing it here.

Code Explanation:

The complete code for this project can be found at the end of this tutorial. However the program is split into important chunks and explained below for your better understanding.

Like all programs we have to begin the code by defining the pin configuration for the pins that we use in our program. Also here we need to define that we are using RB0/INT as an external interrupt pin and not as a input or output pin. The below line of code enables the internal pull-up resistor on portB by making the 7th bit as 0.

```
OPTION_REG = 0b00000000;
```

REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

Then we enable the Global/Peripheral interrupts and declare that we are using RB0 as an external interrupt pin.

```
   GIE=1;          //Enable Global Interrupt

    PEIE=1;         //Enable the Peripheral Interrupt

    INTE = 1;       //Enable RB0 as external Interrupt pin
```

Once the RB0 pin is defined as an external interrupt pin, each time it gets low the external interrupt flag INTF will become 1 and the code inside the void interrupt function will get executed since the Interrupt Service Routine(ISR) will be called.

```
void interrupt ISR_example()

{

        if (INTF==1) //External Interrupt detected

        {

            Lcd_Clear();

            Lcd_Set_Cursor(1,1);

            Lcd_Print_String("  Entered ISR");
```

```
            INTF = 0;          // clear the interrupt flag after done with it

            __delay_ms(2000);

             Lcd_Clear();

        }

}
```

As you can see I have named the interrupt function as ISR_example. You can name it as per your wish. Inside the interrupt function we will check if the INTF flag is high and perform the needed actions. It is very important to **clear the interrupt flag** once you are done with the routine. Only then the program will return back to void main function. This clearing has to be done by software using the line

```
            INTF = 0;          // clear the interrupt flag after done with it
```

Inside the main function, we just increment a number for every 500 ms and display it on the LCD screen. We do not have any specific line to check the status of the RB0 pin. The interrupt will always stay active and whenever the push button is pressed it will jump out of void main and execute the lines in the ISR.

```
    Lcd_Set_Cursor(2,1);

        Lcd_Print_String("Inside Main Loop");

        Lcd_Set_Cursor(1,1);

        Lcd_Print_String("Number: ");

        Lcd_Print_Char(ch1+'0');

        Lcd_Print_Char(ch2+'0');
```

```
        Lcd_Print_Char(ch3+'0');

        Lcd_Print_Char(ch4+'0');



        __delay_ms(500);

        number++;
```

Working of PIC16F877A Interrupts:

Once you have understood **how the interrupt works** you can try it out on the hardware and fiddle around it. This program given here is a very basic example of external interrupt where it just changes the display of the LCD screen when an interrupt is detect.

The complete working of the project can be found in the video given below.Hope you understood about interrupts and where/how to use them. If you have any doubts you can reach me through the forums or through the comment section.

Code

```
#define _XTAL_FREQ 20000000

#define RS RD2

#define EN RD3

#define D4 RD4

#define D5 RD5

#define D6 RD6

#define D7 RD7



#include <xc.h>



#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)

#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)
```

```
#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial P
rogramming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming
)


#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data EE
PROM code protection off)


#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write p
rotection off; all program memory may be written to by EECON control)


#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code
protection off)









//LCD Functions Developed by Circuit Digest.


void Lcd_SetBit(char data_bit) //Based on the Hex value Set the Bits of the Data L
ines


{


if(data_bit& 1)


D4 = 1;


else


D4 = 0;

else


D4 = 0;







if(data_bit& 2)
```

```
D5 = 1;

else

D5 = 0;



if(data_bit& 4)

D6 = 1;

else

D6 = 0;



if(data_bit& 8)

D7 = 1;

else

D7 = 0;

}



void Lcd_Cmd(char a)

{

RS = 0;
```

```
Lcd_SetBit(a); //Incoming Hex value

EN  = 1;

        __delay_ms(4);

        EN  = 0;

}



void Lcd_Clear()

{

Lcd_Cmd(0); //Clear the LCD

Lcd_Cmd(1); //Move the curser to first position

}



void Lcd_Set_Cursor(char a, char b)

{

char temp,z,y;

if(a== 1)

{

 temp = 0x80 + b - 1; //80H is used to move the curser
```

```
z = temp>>4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd_Cmd(z); //Set Row

Lcd_Cmd(y); //Set Column

}

else if(a== 2)

{

temp = 0xC0 + b - 1;

z = temp>>4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd_Cmd(z); //Set Row

Lcd_Cmd(y); //Set Column

}

}




void Lcd_Start()

{

  Lcd_SetBit(0x00);
```

```
  for(int i=1065244; i<=0; i--)  NOP();

  Lcd_Cmd(0x03);

__delay_ms(5);

  Lcd_Cmd(0x03);

__delay_ms(11);

  Lcd_Cmd(0x03);

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes t
he LCD

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes t
he LCD

  Lcd_Cmd(0x08); //Select Row 1

  Lcd_Cmd(0x00); //Clear Row 1 Display

  Lcd_Cmd(0x0C); //Select Row 2

  Lcd_Cmd(0x00); //Clear Row 2 Display

  Lcd_Cmd(0x06);

}



void Lcd_Print_Char(char data)  //Send 8-bits through 4-bit mode

{
```

```
   char Lower_Nibble,Upper_Nibble;

   Lower_Nibble = data&0x0F;

   Upper_Nibble = data&0xF0;

   RS = 1;             // => RS = 1

   Lcd_SetBit(Upper_Nibble>>4);             //Send upper half by shifting by 4

   EN = 1;

   for(int i=2130483; i<=0; i--)  NOP();

   EN = 0;

   Lcd_SetBit(Lower_Nibble); //Send Lower half

   EN = 1;

   for(int i=2130483; i<=0; i--)  NOP();

   EN = 0;

}



void Lcd_Print_String(char *a)

{

int i;

for(i=0;a[i]!='\0';i++)
```

```
  Lcd_Print_Char(a[i]);  //Split the string using pointers and call the Char funct
ion


}


/*****End of LCD Functions*****/









/****Interrupt function ****/


void interrupt ISR_example()


{


        if (INTF==1) //External Interrupt detected


        {


            Lcd_Clear();


            Lcd_Set_Cursor(1,1);


            Lcd_Print_String("  Entered ISR");


           INTF = 0;          // clear the interrupt flag after done with it


           __delay_ms(2000);


            Lcd_Clear();


        }


}
```

```
/****End of Interrupt Function****/



int number =0;

char ch1,ch2,ch3,ch4;

int main()

{

    TRISD = 0x00; //PORTD declared as output for interfacing LCD

    TRISB0 = 1;        //DEfine the RB0 pin as input to use as interrupt pin

    OPTION_REG = 0b00000000;  // Enables PULL UPs

    GIE=1;          //Enable Global Interrupt

    PEIE=1;         //Enable the Peripheral Interrupt

    INTE = 1;       //Enable RB0 as external Interrupt pin

    Lcd_Start();



    while(1)

    {

        ch1 = (number/1000)%10;

        ch2 = (number/100)%10;
```

```
        ch3 = (number/10)%10;

        ch4 = (number/1)%10;



        Lcd_Set_Cursor(2,1);

        Lcd_Print_String("Inside Main Loop");

        Lcd_Set_Cursor(1,1);

        Lcd_Print_String("Number: ");

        Lcd_Print_Char(ch1+'0');

        Lcd_Print_Char(ch2+'0');

        Lcd_Print_Char(ch3+'0');

        Lcd_Print_Char(ch4+'0');



        __delay_ms(500);

        number++;



    }

    return 0;

}
```