EC-310 Microprocessor and Microcontroller Based Design

Chapter -11

PIC 18 Interrupt

Nazar Abbas Saqib
nazar.abbas@ceme.nust.edu.pk

Objectives

- Contrast and compare interrupts versus polling
- Explain the purpose of ISR (Interrupt Service Routine)
- List all the major interrupts of the PIC18
- Explain the purpose of interrupt vector table
- Enable or disable PIC18 interrupts
- Program the PIC18 timers using interrupts
- Define the interrupt priority of PIC18
- Program PIC interrupts in C

Introduction

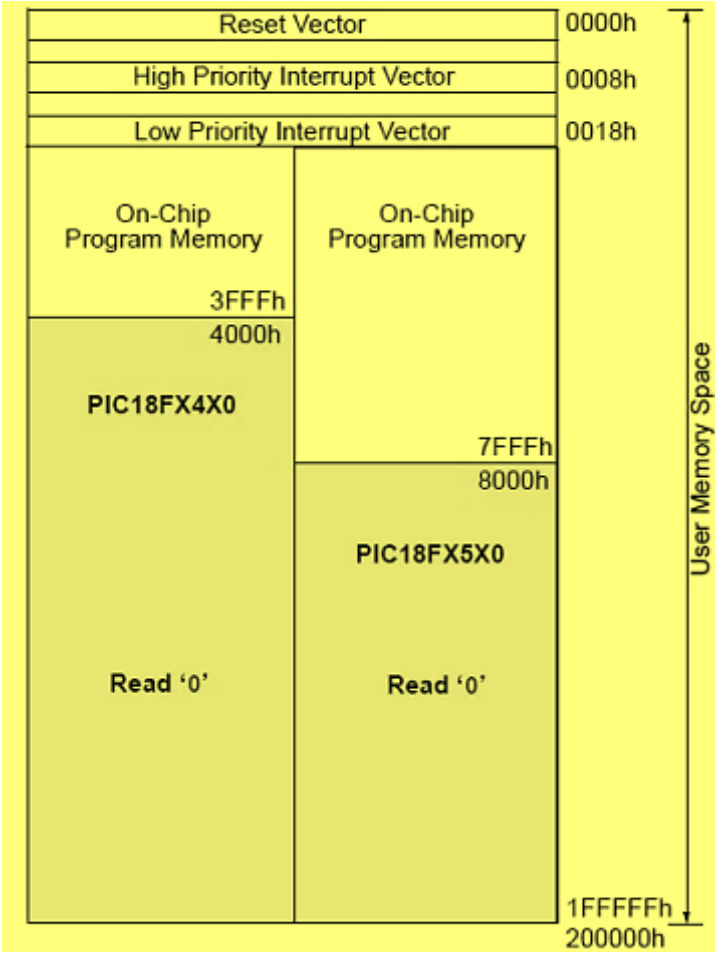
- Interrupts are mechanisms which enable instant response to events such as counter overflow, pin change, data received, etc.
- In normal mode, microcontroller executes the main program as long as there are no occurrences that would cause an interrupt.
- Upon interrupt, microcontroller stops the execution of main program and commences the special part of the program(ISR) which will analyze and handle the interrupt.

Polling vs Interrupts

- PIC can serve multiple devices using mechanisms of
  - Polling
    - PIC continuously monitors the status of each device
    - Each device get the attention of the CPU as the same level of priority
    - Wastes u-Controllers time by polling devices that do not need service.
  - Interrupt
    - Devices get the attention of the CPU only when it needs a service
    - Can service many devices with different level of priorities

Interrupt service routine (ISR)

- When an interrupt is invoked the uC runs the Interrupt Service Routine(ISR)
- Interrupt vector table holds the address of ISRs
  - Power-on Reset 0000h
  - High priority interrupt 0008h
  - Low priority interrupt 0018h



Steps in executing an interrupt

- Upon activation of interrupt the microcontroller
  - Finishes executing the current instruction
  - Pushes the PC of next instruction in the stack
  - Jumps to the interrupt vector table to get the address of ISR and jumps to it
  - Begin executing the ISR instructions to the last instruction of ISR (RETFIE)
  - Executes RETFIE
    - Pops the PC from the stack
    - Starts to execute from the address of that PC

Sources of interrupts in PIC18

- Interrupts for each of the Timers 0,1,2 and so on
  - Timer0 , Timer1 ,Timer2
- Three External hardware interrupts
  - Pins RB0(INT0),RB1(INT1),RB2(INT2)
- Two interrupts for Serial communication's UART
  - One for receive, One for transmit
- PORTB-Change Interrupt
- ADC (analog to digital converter)
- CCP (compare capture pulse width modulation, PWM)
- etc

INTCON REGISTER

INTCON register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

GIE	enable/disable unmasked interrupts
PEIE	enable/disable unmasked periferal interrupts
T0IE	enable/disable Tmr0 interrupts
INTE	enable/disable RB0/Int interrupt
RBIE	enable/disable RB port change interrupt
T0IF	Tmr0 register overflow
INTF	RB0/Int interrupt occurred
RBIF	one of the PORTB (rb4...rb7) has changed state

INTCON REGISTER

TMR0 Interrupt Enable
1= TMR0 interrupt enabled
0= TMR0 interrupt disabled

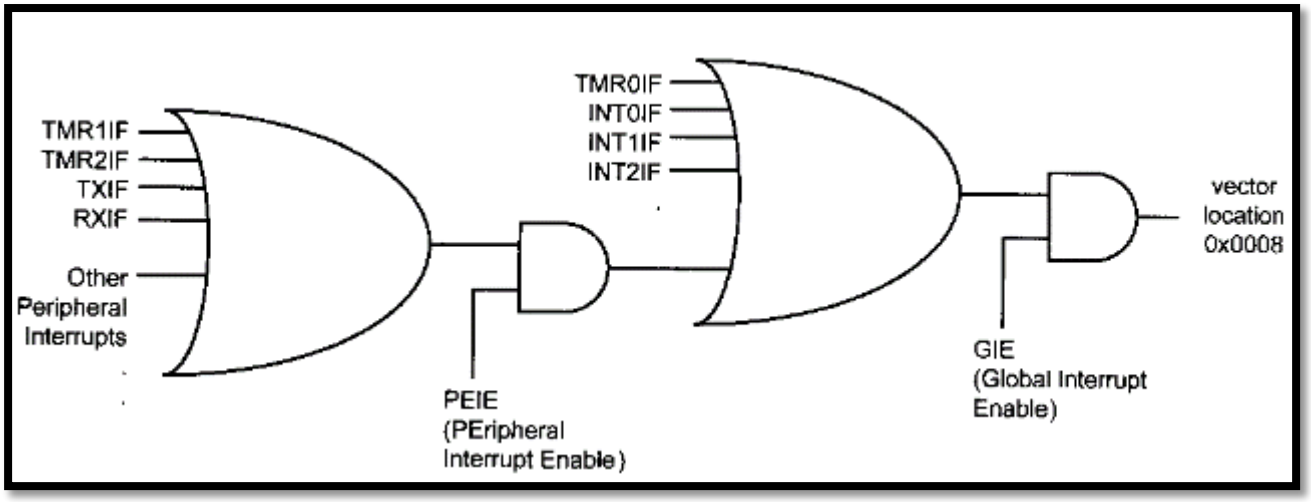
Timer0 Interrupt Flag
1= TMR0 has overflowed
0= TMR0 did not overflowed

GIE (Global Interrupt Enable)
0= Disable all interrupts
1= enable all interrupts

PEIE (PEripheral Interrupt Enable)
0= Disable many of periphers, timers 1,2
1= Enable many of periphers, timers 1,2

Steps in enabling an interrupt

- When CPU is powered up all interrupts are disabled (Masked)
- Interrupts must be enabled (unmasked) through software
- To enable any interrupt, Set the GIE bit from INTCON REG, BSF INTCON GIE
- Then Set the IE bit for that interrupt
- If the interrupt is one of the peripheral (timers 1,2 , serial, etc) set PEIE bit from INTCON register



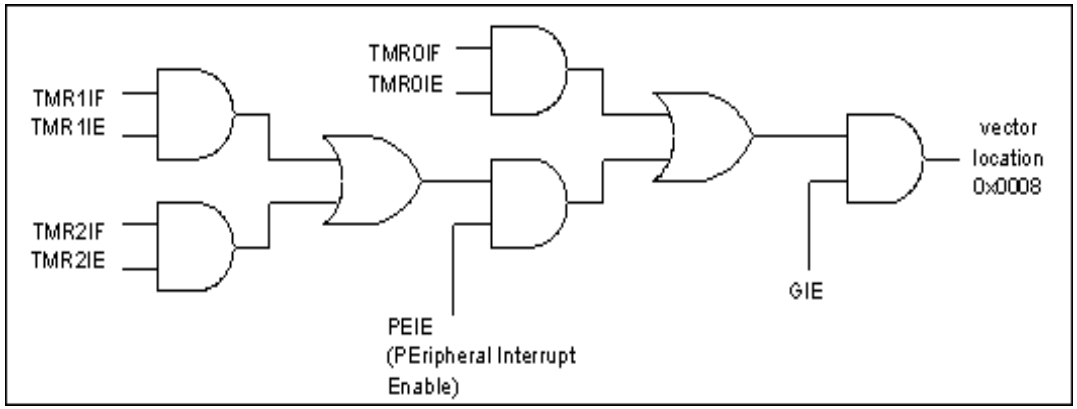
Example 11.1

a)

```
BSF INTCON,TMR0IE      ;enable Timer0 interrupt,
BSF INTCON,INT0IE      ;enable external interrupt (INT0)
BSF INTCON,GIE         ;allow interrupts to come in
```

Or

```
MOVLW B'10110000'
MOVWF INTCON
```



b)

```
BCF INTCON,TMR0IE      ;disable Timer0 interrupt
BCF INTCON,GIE         ;mak all interrupts globally
```

Program 11-4 Timer0 interrupt

The program uses Timer0 to generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD

```
        ORG 0000H
        GOTO MAIN                   ;bypass interrupt vector table

        ORG 0008H                   ; on default all interrupts land at address 00008
        BTFSS INTCON,TMR0IF         ; Monitor Timero interrupt
        RETFIE                      ; No, Then return to main
        GOTO INT0_ISR               ; yes, go to Timer0 ISR

        ORG 00100H
MAIN                                ; Main program for initialization and keeping CPU busy
        BCF TRISB,5                 ; make PB5 as an output
        CLRF TRISD                  ; make PORTD output
        SETF TRISC                  ; make PORTC input
        MOVLW 0X08                  ;Timer0, 16-bit, no prescale, internal clock
        MOVWF T0CON                 ; load T0CON reg
        MOVLW 0XFF                  ; TMR0L=FF, the high byte
        MOVWF TMR0H                 ; load Timer0 high byte
        MOVLW 0XF2                  ;TMR0L=F2, the low byte
        MOVWF TMR0L                 ; load Timer0 low byte
        BCF INTCON, TMR0IF          ;clear timer interrupt flag bit
        BSF T0CON,TMR0ON            ; start timer0
        BSF INTCON,TMR0IE           ; enable Timer0 interrupt
        BSF INTCON,GIE              ; enable interrupts globally
OVER    MOVFF PORTC,PORTD           ; send data from PORTC to PORTD
        BRA OVER                    ; stay in this loop for ever
INT0_ISR
```

…… Next page

Program 11-4 Timer0 interrupt

Continued from the previous slide …….

```
INT0_ISR
        ORG 200H
        MOVLW 0XFF              ; TMR0L=FF, the high byte
        MOVWF TMR0H             ; load Timer0 high byte
        MOVLW 0XF2              ;TMR0L=F2, the low byte
        MOVWF TMR0L             ; load Timer0 low byte
        BTG PORTB,5             ; toggle RB5
        BCF INTCON, TMR0IF      ; clear timer interrupt flag bit
        RETFIE                  ; return from interrupt
        END
```

Note: To start the timer again,
first there is a need to reload timer registers TMR0H & TMR0L
Second, to clear the interrupt flag bit.

Program 11-4 Timer0 interrupt

RETURN VS RETFIE

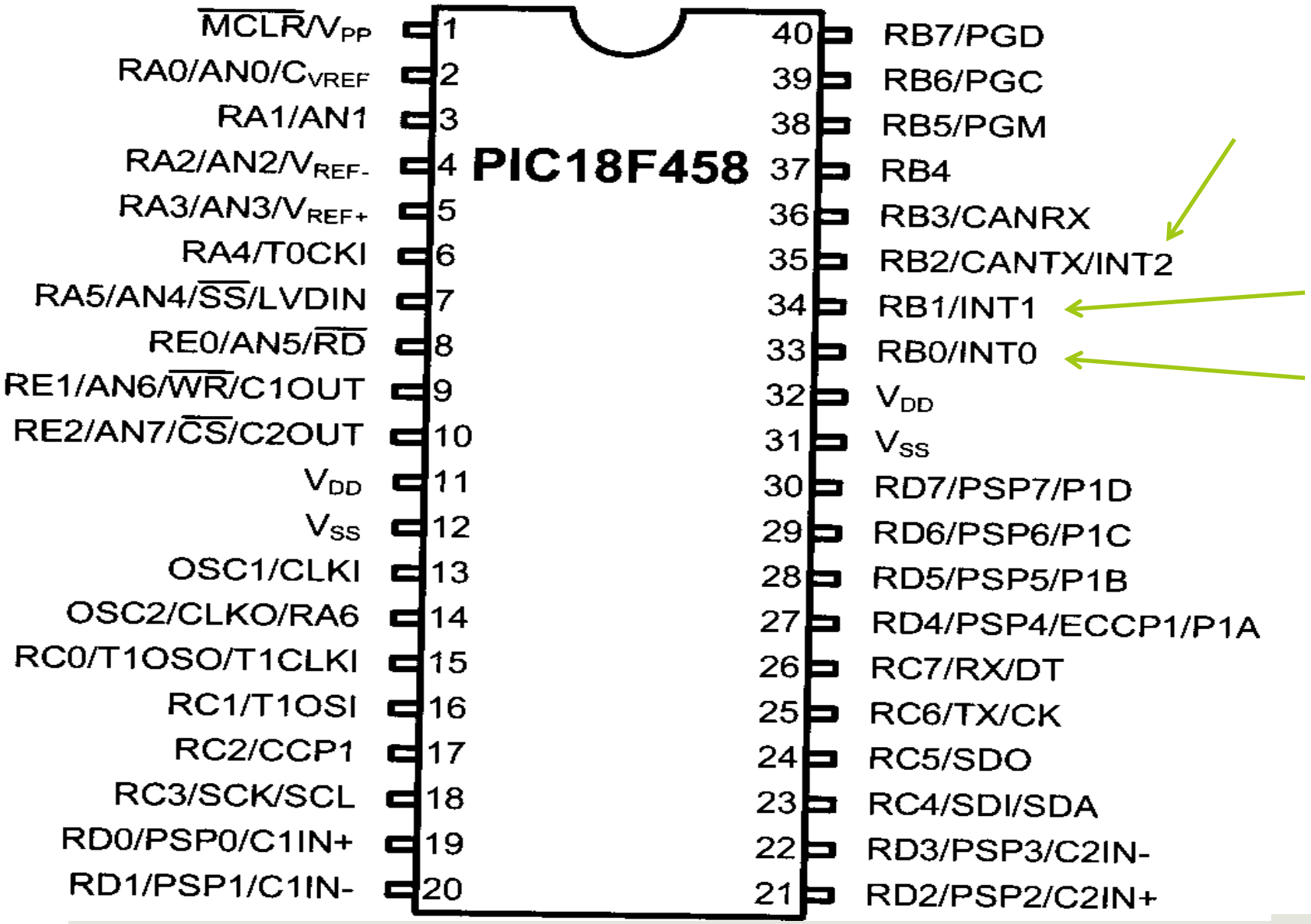
Example 11-2

What is the difference between the RETURN and RETFIE instructions? Explain why we cannot use RETURN instead of RETFIE as the last instruction of an ISR.

Solution:

Both perform the same actions of popping off the top bytes of the stack into the program counter, and making the PIC18 return to where it left off. However, RETFIE also performs the additional task of clearing the GIE flag, indicating that the servicing of the interrupt is over and the PIC18 now can accept a new interrupt. If you use RETURN instead of RETFIE as the last instruction of the interrupt service routine, you simply block any new interrupt after the first interrupt, because the GIE would indicate that the interrupt is still being serviced.

Program 11-4 External hardware interrupt

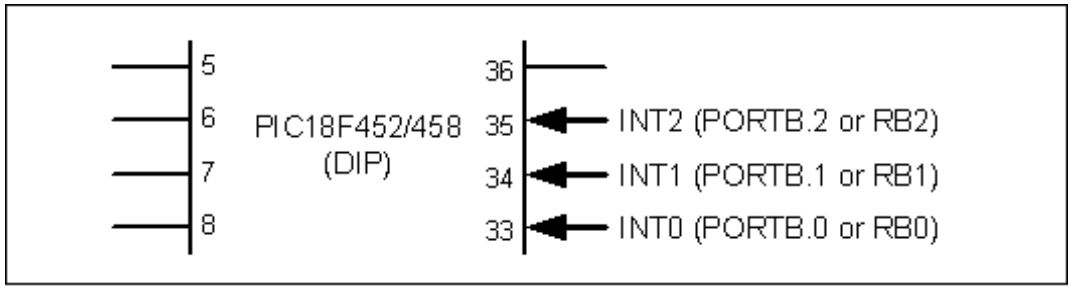


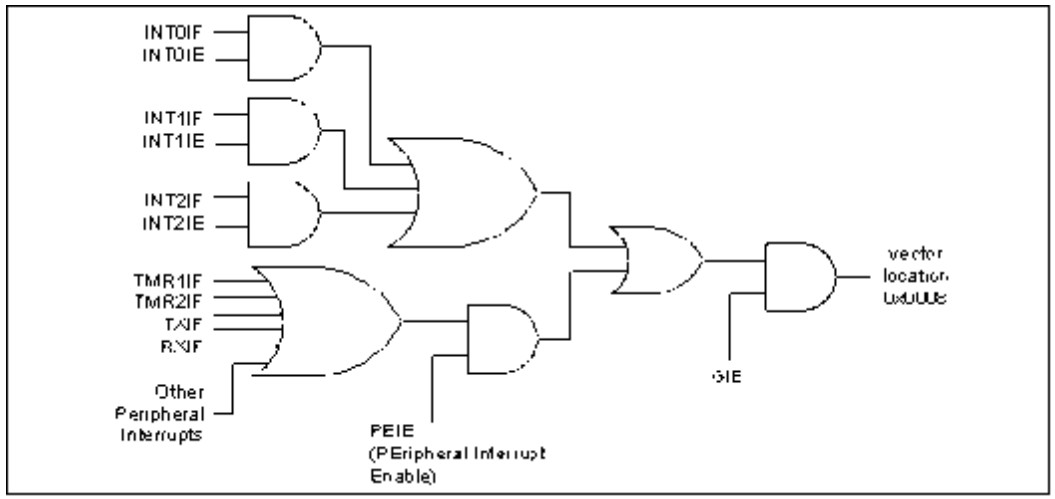
Program 11-4 External hardware interrupt

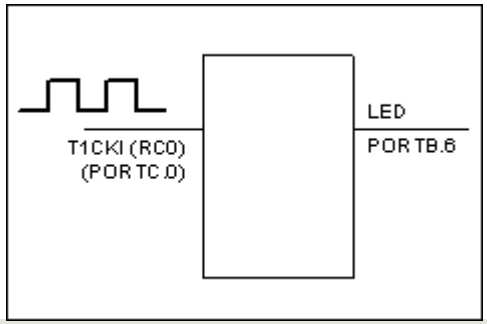
```
        ORG 0000H
        GOTO MAIN

        ORG 0008H
        BTFSS INTCON,INT0IF
        RETFIE
        GOTO INT0_ISR

        ORG 00100H
MAIN
        BCF TRISB,7
        BSF TRISB,INT0
        CLRF TRISD
        SETF TRISC
        BSF INTCON,INT0IE
        BSF INTCON,GIE
OVER    MOVFF PORTC,PORTD
        BRA OVER
INT0_ISR
        ORG 200H
        BTG PORTB,7
        BCF INTCON,INT0IF
        RETFIE
        END
```







PIC18 interrupt programming in C

```
#pragma code high_vector =0x0008  // High-priority interrupt location
void My_HiVect_Int (void)
{
_asm
GOTO my_isr
_endasm
}
#pragma code                      // End of code
```

- C18 compiler uses '#pragma code' to place a code at a specific ROM address
- C18 does not place an ISR at the interrupt vector table, we use assembly language GOTO statement to do that

- Now to redirect it from location 00008 to another program to find the source of the interrupt and finally to ISR
- Use of reserved keyword 'interrupt'

```
#pragma interrupt my_isr     //interrupt is reserved keyword
void my_isr (void)           //used for high-priority interrupt
{

//C18 places RETFIE here automatically due to
//interrupt keyword
}
```

PIC18 interrupt programming in C

```
//Program 11-2C (C version of Program 11-2)
#include <p18F458.h>
#define myPB1bit PORTBbits.RB1
#define myPB7bit PORTBbits.RB7
```

```
void T0_ISR(void);
 #pragma interrupt chk_isr  //used for high-priority
                            //interrupt only
 void chk_isr (void)
 {
    if (INTCONbits.TMR0IF==1)   //Timer0 causes interrupt?
         T0_ISR();              //Yes. Execute Timer0 ISR
 }
```

```
#pragma code My_HiPrio_Int=0x08//high-priority interrupt
void My_HiPrio_Int (void)
{
  _asm
    GOTO chk_isr
  _endasm
}
#pragma code
```

PIC18 interrupt programming in C

```
void main(void)
  {
     TRISBbits.TRISB6=0;    //RB6 = OUTPUT
     TRISCbits.TRISC0=1;    //PORTC0 = INPUT
     TRISD=0;
     T0CON=0x08;            //Timer0, 16-bit mode,
                            //no prescaler
     TMR0H=0;               //load Timer0 high byte
     TMR0L=0;               //load Timer0 low byte
    INTCONbits.TMR0IF=0;  //clear TF0
    INTCONbits.TMR0IE=1;  //enable Timer0 interrupt
    INTCONbits.GIE=1;     //enable all interrupts globally
    while(1);  //keep looping until interrupt comes
  }
  void T0_ISR(void)
    {
      myPB1bit=~myPB1bit;     //toggle PORTB.1
      TMR0H=0x35;             //load TH0
      TMR0L=0x00;             //load TL0
      INTCONbits.TMR0IF=0;    //clear TF0
    }
```