Handling Interrupts in PIC

Interrupts in PIC18F

- Only 2 Interrupt Vectors: High and low vectors
- Multiple Peripheral interrupts mapped to one vector
- Interrupt source identified by reading interrupt flag
  Set => triggered

Interrupt Sources:

- Timers
- PORTB
- External interrupts
- SPI
- UART
- USB
- Etc.

```
#pragma code
#pragma code _HIGH_INTERRUPT_VECTOR = 0x1008     //HIGH INTERRUPT VECTOR Location
void high_ISR (void)
{
    _asm goto chk_isr _endasm    //ISR for handling high priority interrupts
}

#pragma code
#pragma code _LOW_INTERRUPT_VECTOR = 0x1018  //LOW INTERRUPT VECTOR Location
void low_ISR (void)
{

}
#pragma code
```

Figure 1: Typical pseudo-code for handling interrupts

Timer Interrupts

Timer Overflow Interrupts

- Maximum timer count reached
- In Timers 0,1,2 and 3

Interrupt on match

- Timer count reaches a count match
- In Timer2

Initialization

- Enable global interrupts
- Enable Timer overflow/match interrupt

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Figure 2: INTCON register showing Interrupt Enable and Flag bits

Enable Timer0 Overflow Interrupt

- Set TMR0IE in Timer Initialization
- Clear TMR0IF
- Set GIE
- Set Priority(Optional)
- Put appropriate values in TMR0L,TMR0H
- Start the timer

Handle Timer0 Interrupt

- Clear TMR0IF
- Put appropriate values in TMR0L,TMR0H
- Perform any operation(if required)

Enable PORTB(RB7:RB4) Interrupt

- Set RBIE in PORTB Initialization
- Set bits for enabling input operation on RB7:RB4
- Clear RBIF
- Set GIE
- Set RBPU bit in INTCON2 register for enabling Weak Pull-up

Handle PORTB Interrupt

- Clear RBIF
- Perform any operation(if required)

Few important points:

- PORTB interrupt is triggered by change of input on any of RB7:RB4
  To allow triggering of interrupt using any of the pin set that pin as input, rest as output
  If all are set as inputs, identify change using PORTB register
  (Value will change from 0xF0)

- Along with Timer0 interrupts, if Timer0 register read is needed,
  First, read TMR0L then TMR0H (Assigning values to a variable)
  x = TMR0L
  y = (TMR0H<<8)
  Timer count = x + y